Yugoslav Journal of Operations Research
2 (1992), Number 2, 207–218

PROVIDING HELP IN A NATURAL LANGUAGE QUERY INTERFACE TO RELATIONAL DATABASES

Sanja PETROVIĆ

Mihajlo Pupin Institute,
P. O. Box 15, 11000 Belgrade, Yugoslavia

Abstract. The aim of this paper is to present a possible help prototype for IBM SAA LanguageAccess — a natural language query interface to relational databases. The role of help is to assist the user in formulating queries. It is done by extracting and presenting information, syntactic and semantic, about natural language terms and the concepts behind them from a conceptual schema which links the natural language to tables in a particular database. The presented help functions have been prototyped using IBM Prolog.

Key words and phrases: natural language interface, relational databases

1. INTRODUCTION

The recent years have seen an extraordinary acceleration of interest in the application of natural language processing to relational databases inquiry. Natural language queries achieve a high degree of acceptance, particularly among users who are not prepared to learn the syntax of formal query languages. Moreover, in order to obtain information from relational databases, the user must be familiar with database structure. Using the natural language interface in a database system frees the user from learning the syntax of formal query languages and details of the database structure.

As database systems are of wide-spread use, research on the natural language interface has flourished during the past years [**1**]. An example is the known IBM SAA LanguageAccess — a multilingual natural language query interface to IBM relational databases DB2 and SQL/DS [**2**]. The product is developed at IBM Nordic Laboratories in cooperation with IBM research laboratories all over the world. LanguageAccess translates a natural language query into SQL statements which are then interpreted in the usual way.

However, the receiving advantage to have the natural or human language query mechanism must be paid off. Words of a natural language are inherently vague and refuse formal definition. The phrases in a natural language are often ambiguous and clear understanding depends on a large amount of contextual information. For this reason, some authors have claimed that a real natural language query interface to a relational database requires an almost complete array of the capabilities of human intelligence to succeed [3].

The aim of the paper is to brighten one specific aspect of the natural language interface problem, to portray a Help function which can assist the user in forming correct natural language queries. A prototype of this Help function for extracting and presenting information about natural language terms that can be used in queries and natural language fragments based on these terms, is presented. The paper is organized in the following way. In the next section an overview of LanguageAccess features will be given briefly. Section three describes the customization process which is the task of linking the vocabulary that will be used in natural language queries with database tables. The conceptual schema, as the output of the customization process, will be described. Section four is devoted to the Help functions. The role of Help will be demonstrated using a number of examples. Particular attention is paid to the specification and design of Help functions.

2. AN OVERVIEW DESCRIPTION OF NATURAL LANGUAGE INTERFACE TO RELATIONAL DATABASE IN LanguageAccess

With LanguageAccess the user can retrieve database information in his own language. The user does not need to learn either the syntax of SQL statements or database structure details. For example, consider a case of a relational database covering employees in a firm. In order to retrieve information about managers who manage more than ten employees by means of SQL, the user of database system has to look at the structure of the database EMPLOYEE and construct the following SQL statement

```
SELECT NAME, SALARY
FROM EMPLOYEE
WHERE NUMBER =
        SELECT MGR
        FROM EMPLOYEE
        GROUP BY MGR
        HAVING COUNT(*) > 10
```

However, by using LanguageAccess the user can get the same information in a more friendly way, such as: *List the name and salary of the managers who manage more than ten employees.*

To interpret such a sentence a very complex system is required. LanguageAccess has three related parts: a customization tool, a natural language engine and a query interface. We shall briefly explain each of them [5].

1) **The customization tool.** Each application must be customized before the user can use LanguageAccess. The task of customization is to link natural language words and phrases to a predefined set of database tables which appear in the application domain. Customization is done from the user-friendly graphical interface in the customization tool [6].

2) **The natural language engine.** It is the kernel of the system which interprets questions using a linguistic analysis component and produces the SQL statements with information about the requested form of the answer: chart, report, etc.

The processing stages of the natural language engine are presented in Figure 1. Their tasks are:

— The input sentence is parsed in the syntactic analysis stage. Till now, analysis grammars for a number of languages have been prototyped: English, German, French, Spanish, Italian and Swedish. LanguageAccess is commercially available for English and German.

— Semantic analysis produces an interpretation of the input question which still has no references to database organization but is determined by facts asserted during the customization process.

— Generation of the SQL statement

— Generation of natural language paraphrases for the interpretations of the question. If the input question is ambiguous, one paraphrase for each interpretation is produced. The user selects the paraphrase which represents the desired meaning of the question. For example, the question: *Which senior managers work at the head office*? is paraphrased as: *Find senior managers that work at departments named head office.* Each interpretation is converted to one or more SQL statements. After selecting a paraphrase, the corresponding SQL statement is executed.

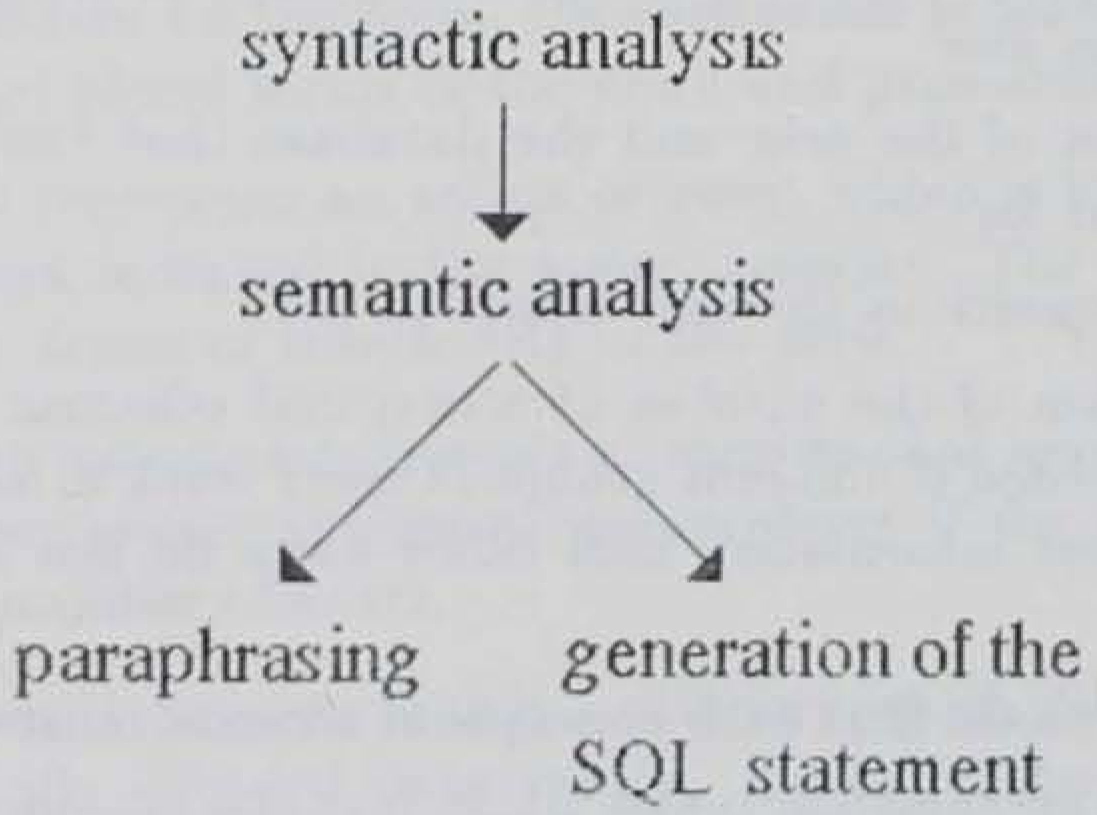

Figure 1. Processing stages in the natural language engine

Looking at the engine structure, two parts can be distinguished: the language dependent part, such as analysis grammar, generation grammar used in paraphraser, and a language independent part. This gives the multilingual capability of the engine.

3) **The query interface** lets users enter questions in natural language, confirming interpretations and viewing the answer. A very interesting and useful feature of LanguageAccess is resolution of pronouns, both inter- and intra-sententially.

The latter enables the user to use a pronoun instead of the noun referred to in the previous sentence. For example, the user can ask: *List employees of the Computer Science Department.* After getting the answer the user can continue with the question: *Who is their manager?* relating *their* to the subject of the preceding question.

Besides the questions concerning the database content, the user can ask, through natural language, about the database tables themselves. For example, the user can ask: which database tables exist, who is their creator about words and their part of speech, etc. This meta-knowledge is stored automatically in relational database tables during the customization process and becomes part of LanguageAccess application. .

3. CUSTOMIZATION PROCESS AND CONCEPTUAL SCHEMA

A conceptual schema is a collection of facts created as a result of the customization process. The schema defines the words that can be used in natural language questions in order to retrieve particular data from the database.

' The conceptual schema is used by various parts of the engine to analyze questions in natural language, interpret them and generate paraphrases and SQL statements. The conceptual schema is used by the help function, also. A description of the main stages of the conceptual schema will be presented.

The customization process is accomplished in an interactive way through three stages [6]. The first stage describes the planning steps of the customization process. These planning steps are:

— Identification of the user and the database that contains the information the user is interested for.

— Listing the questions the user wants to ask.

— Determination of the number of conceptual schemas. More than one conceptual schema is needed if different groups of users want to ask their own questions or some users request information that other users do not have the authority to access.

— Listing the words that each conceptual schema must contain.

— Analysis of the database tables which includes specification of: tables' keys, inclusion dependencies and join paths between tables and exclusion of the columns containing data the user will not want to ask questions about.

— Analysis for the entities that the conceptual schema will contain. An entity is representation of an object in the domain of the application. To each entity one or more terms are assigned. A term is the word or phrase in the natural language which will be used in the questions.

The objective in the second stage of the customization process is to download information from the database, to select the tables and columns that the conceptual schema will use and define dependencies among columns.

In the third stage of the customization process the conceptual schema is developed. The main steps in developing the conceptual schema are:

1. NAMING entities.

2. CLASSIFYING each entity as a subclass or instance of at least one other class. A classified entity is added to the hierarchy of classes which contains both built-in and customizer defined classes.

3. CREATING TERMS for each entity that users want to refer to. The first term associated with the entity is considered as the primary term which will appear in the paraphrase of the question. The other terms are synonyms to the primary term and may be used in questions, too.

Each term belongs to one of the four possible categories: **noun**, **verb**, **adjective** or **proper name**.

a) **Noun.** Each table or column used in the conceptual schema is an entity which belongs to the category noun, because it represents people or things. Other nouns in the conceptual schema represent:

— composite entities consisting of two or more entities. For example, the noun *interview* consists of two entities: *first name* and *last name*. Constituent entities specify the order of the corresponding data values which can be used in questions or given in paraphrases if the question concerns the composite entity.

— An SQL expression performed on one or more columns. For example, the noun *duration* represents *end date* minus *start date*.

After specifying terms for the noun, the customizer is asked to define the noun grammar: singular and plural forms of the noun and pronouns which refer to it.

b) **Verb.** A verb represents an action or event which is linked to a noun. For example, the verb work is linked to the noun *employee*. The customizer specifies the verb grammar i.e. forms of transitivity of the verb.

c) **Adjective.** An adjective represents a category that applies to a noun entity. For example, adjective *senior* applies to an *employee* if the age of experience is greater than a given number of years.

d) **Proper name.** A proper name is a category which designates a particular being or thing, typically encoded data values. *Head-Quarters* (*HQ*), instance of department with a predefined number, is an example of a proper name.

4. Specifying SYNTAX for entities. Syntax prescribes the usage of terms in a natural language question. It is obligatory for verbs and optional for other categories. We shall briefly discuss verb syntax.

The syntax for the verb determines how the verb will be used in questions. In English, a verb phrase can be used without a direct object, with one or two direct objects or with one direct and one indirect object. For some verb phrases it is also possible to select prepositions which will be used with the verb. For example, suppose that the verb *work* has to be used in the context: *employee works for* *manager*. The verb complement window contains four alternative phrases with the verb: *who works*, *who works what*, *who works what to whom* and *who works whom whom*. As the verb *work* has no direct object, the customizer selects the first option. After defining the verb complement, the list of possible prepositions is proposed to the customizer. The following preposition complement form should be selected from the list proposed: *somebody works for something*. Verb syntax specifies how the prepositions selected fit into a phrase. If there are two prepositions defined, syntax definition specifies whether they can be used interchangeably or not. Finally, the following verb syntax should be selected: *who works for something*.

5. Specifying RELATIONSHIPS between entities and prepositions for relationships. The proposed list of possible relationships between two entities depends upon the class of each entity, type of terms of each entity and their syntax. Relationships determine the context in which terms defined can be used. Let us continue the example with the verb work. Assume that the customizer has selected the relationship: *Who works?* between the verb work and its subject *employee* and relationship: *What does someone work for?* between *work* and *manager* (Figure 2). In that case, the user can ask questions such as: *Who works for Brown?*, *List employees who work for Brown* etc.

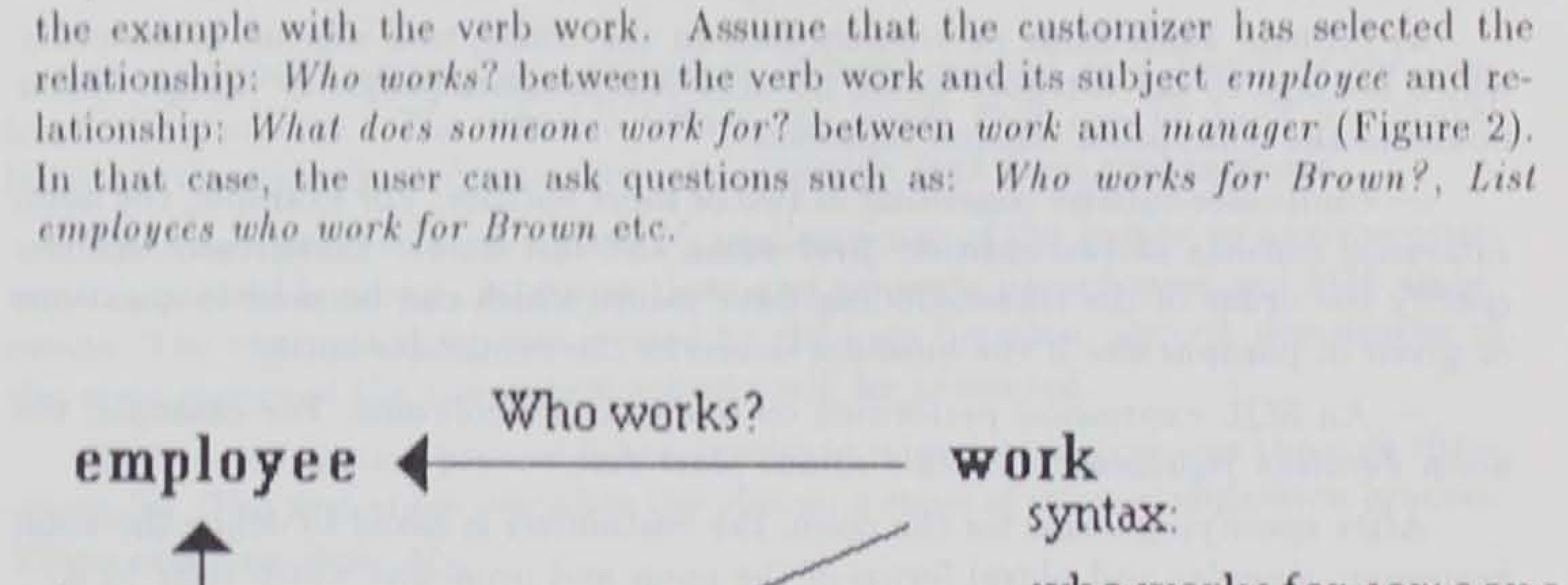

Figure 2. Syntax, prepositions and relationships defined for the verb work

A subclass inherits relationships associated with its class. For example, if *manager* is defined as a subclass of *employee*, *manager* inherits the relationship between *employee* and *work* (Figure 2). So, the user is allowed to express: *manager works*, although it is not stated explicitly in the conceptual schema.

The conceptual schema is presented as a collection of Prolog facts. For example, nominative role of the noun *employee* to verb *work* is presented with a predicate: *nom(work,employee)*.

A simple conceptual schema with a few terms from each of the four possible categories is presented in Figure 3. A brief explanation of entities and relationships among them is following. *Senior* is defined as an adjective with one meaning which is applicable to noun *employee*. This implies that *senior* is applicable to all subclasses of *employee*: *clerk* and *manager*. Verb *manage* is connected with two nouns: *manager* and *department*. *Manager* and *department* have nominative and accusative role, respectively, to verb *manage*. The accusative role of *department* is inherited by its instance *HQ*. Relationships between *work*, *employee* and *manager* are already explained in the paragraph about verb syntax. *Employee* is connected to *salary* via relationship *measured by*. There is relationship *possesses* between *employee* and *department*, and vice versa.

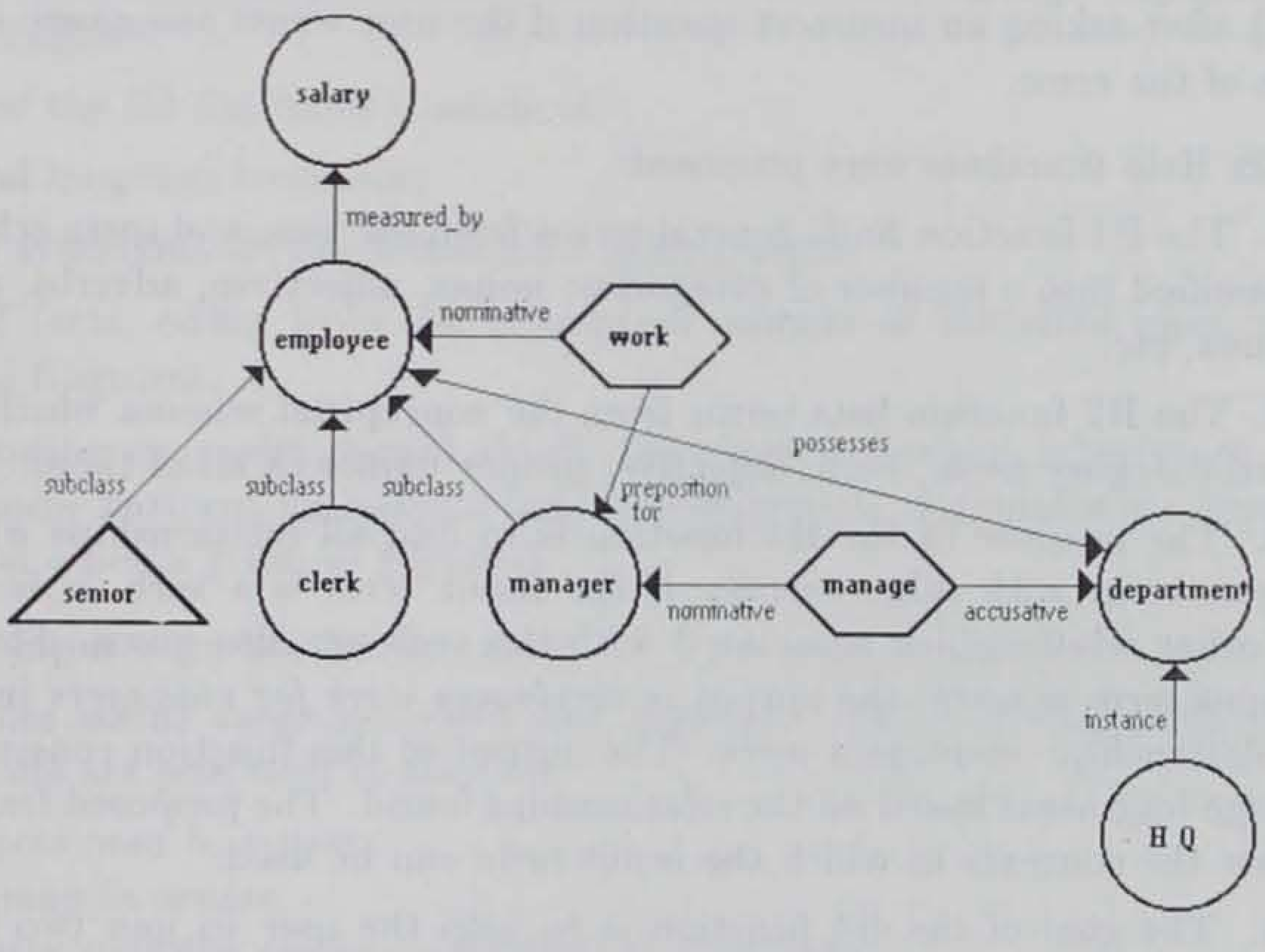

Figure 3. An example of conceptual schema

The conceptual schema contains word specific to one application. However, there are words which may be useful in all applications regardless of their domains, like preambles and polite words, measure words, pronouns, etc. As an example, preambles and polite words like: *can you*, *please*, *would you* etc. are suitable to be used in many questions. The entities representing such words are already defined and kept in the so called base schema. Similarly, the meta schema contains words which can be used to obtain the meta-knowledge about databases themselves. The two latter schemas are an incorporated part of the engine.

4. THE HELP FUNCTIONS

The user is not completely free in formulating questions in natural language. The conceptual schema determines the forms of the questions the user can ask. Only terms associated with entities in the conceptual schema and general terms from the base and meta schema are allowed to be used in questions in the context which is specified by the definition of entities and relationships between them. This is not a strong restriction, because a proper customization results in a wide variety of possible questions. The required answer can usually be retrieved through syntactically different questions.

In order to ask correct questions, the user often needs help. The purpose of the help is to inform the user about the contents of the conceptual schema in a form of a natural language report. This help is based on a sophisticated search through the facts of conceptual schema guided by the meaning of these facts.

Help can be invoked by the user in two cases:

1) before asking the question if the user needs help in asking a correct question.

2) after asking an incorrect question if the user wants assistance to detect the source of the error.

Six Help functions were proposed:

1. The H1 function finds general terms from the base and meta schema. Terms are classified into a number of categories: nouns, adjectives, adverbs, prepositions, pronouns, etc.

2. The H2 function lists terms from the conceptual schema which belong to a selected category noun, verb, adjective, proper namer or all of them.

3. The purpose of the H3 function is to find all relationships which connect an input term with other terms. If the input term is a verb or is related to a verb, other relationships associated with the verb are also given. For example, if the input term is *work*, the output is *employees work for managers* instead of just one relationship: *employees work*. The output of this function consists of natural language fragments based on the relationships found. The proposed fragments show the user the contexts in which the input term can be used.

4. The goal of the H4 function is to help the user to join two terms in the queries. The output of the function is a list of natural language fragments which contains two input terms.

If there are no relationships which connect both terms, the user can ask for the lists of relationships associated with each term separately. These lists are actually the output of the H3 function.

5. The H5 function offers the user sample queries based on the input natural language fragment produced by the H3 function. Sample queries are based on patterns which present different types of possible queries. These queries can be processed by the analysis grammar.

6. The H6 function helps the user in the case the query cannot be processed. A list of query terms which belong to the conceptual schema is given to the user. By invoking the H3 function the user will be informed about the context in which these terms can be used.

These proposed functions are described in more details below.

H1 function. Base and meta vocabulary terms are described by base and meta schema facts. The H1 function examines these facts and generate the base and meta terms.

• **H2 function.** Function H2 collects all terms which belong to the input category from the conceptual schema facts.

• **H3 function.** Function H3 searches through conceptual schema facts in order to find all entities which are connected through relationships with the entity related to the input term. Besides relationships explicitly presented by conceptual schema facts, all inherited relationships have to be taken into account. Relationships are picked up by using the Prolog's backtracking facility. Collected relationships are presented to the user in the form of natural language fragments. For each set of relationships, syntactic patterns of the natural language fragments are defined by a linguist.

Output of the H3 functions consists of:

1) natural language fragment,

2) list of synonyms for the terms used in fragments,

3) list of facts, either from the conceptual schema or inherited ones, which determine the fragment.

The following examples should clarify the most important inheritance rules and present some patterns for natural language fragments. Examples are based on the conceptual schema given in Figure 3.

EXAMPLE 1. Input term is adjective *senior*.

Senior modifies nouns *employee*, *clerk* and *manager*. The following natural language fragments are proposed to the user:

- *employees may be senior*,
- *clerks may be senior*,
  (*senior* modifies clerks — subclass of *employee*),
- *managers may be senior*,
  (*senior* modifies *manager* — subclass of *employee*).

EXAMPLE 2. Input term is verb *manage*.

Verb patterns join two or more relationships assigned to the verb to form a meaningful natural language fragment.

In our example, the user can use the following fragments in this questions:

- *managers manage departments*,
- *managers manage HQ*,
  (*HQ* is successor of *department* and inherits relationships assigned to it),
- *departments are managed*,
- *HQ is managed.*

EXAMPLE 3. Input term is noun *manager*.

Searching for the relationships assigned to the input noun is organized in two parts. In the first part from the relationships which connect the noun *manager* to other nouns, the following fragments are generated:

- *managers have employees*,
- *employees have managers*,
- *managers have departments*,
  (relationship inherited from *employee* — of *manager*),
- *departments have managers*,
- *managers have salaries*
  (relationship inherited from *employee*).

In the second part the H3 function searches for the verbs related to the input noun. Noun *manager* can be used with the verbs *work* and *manage* in the following context:

- *managers manage departments*,
- *managers manage HQ*,
- *employees work for managers*,
- *clerks work for managers*
  (*clerks* inherits relationships assigned to its ancestor — *employee*).

• **H4 function.** The H4 function proposes natural language fragments which contain two input terms. Fragments are formed in two ways:

1) as the output of H3 function,

2) by concatenating two fragments produced as the output of H3 function.

The following example should demonstrate the usefulness of H4 function. Let us assume that the following query is asked: *List all departments which code in Prolog* (conceptual schema entities are given in Figure 4). The query is composed of correct terms but cannot be processed because there are no relationships between entities related to *department* and *code*. Additional relationships associated to the entities related to the terms in the query must be included so that the query can be processed. The user should invoke the H4 function with *department* and *code* as input terms. The H4 function will concatenate two fragments associated with *department* and *code*:

- *departments have programmers* and
- *programmers code in Prolog*

and will form the resulting fragment:

- *departments have programmers that code in Prolog.*

• **H5 function.** A set of several queries patterns is assigned to each pattern of natural language fragment.

For example, for the natural language fragment:

- *managers manage departments*

the output is the following list of queries:

- *Which managers manage departments?*
- *What department was managed by manager?*
- *Tell me which managers manage departments.*

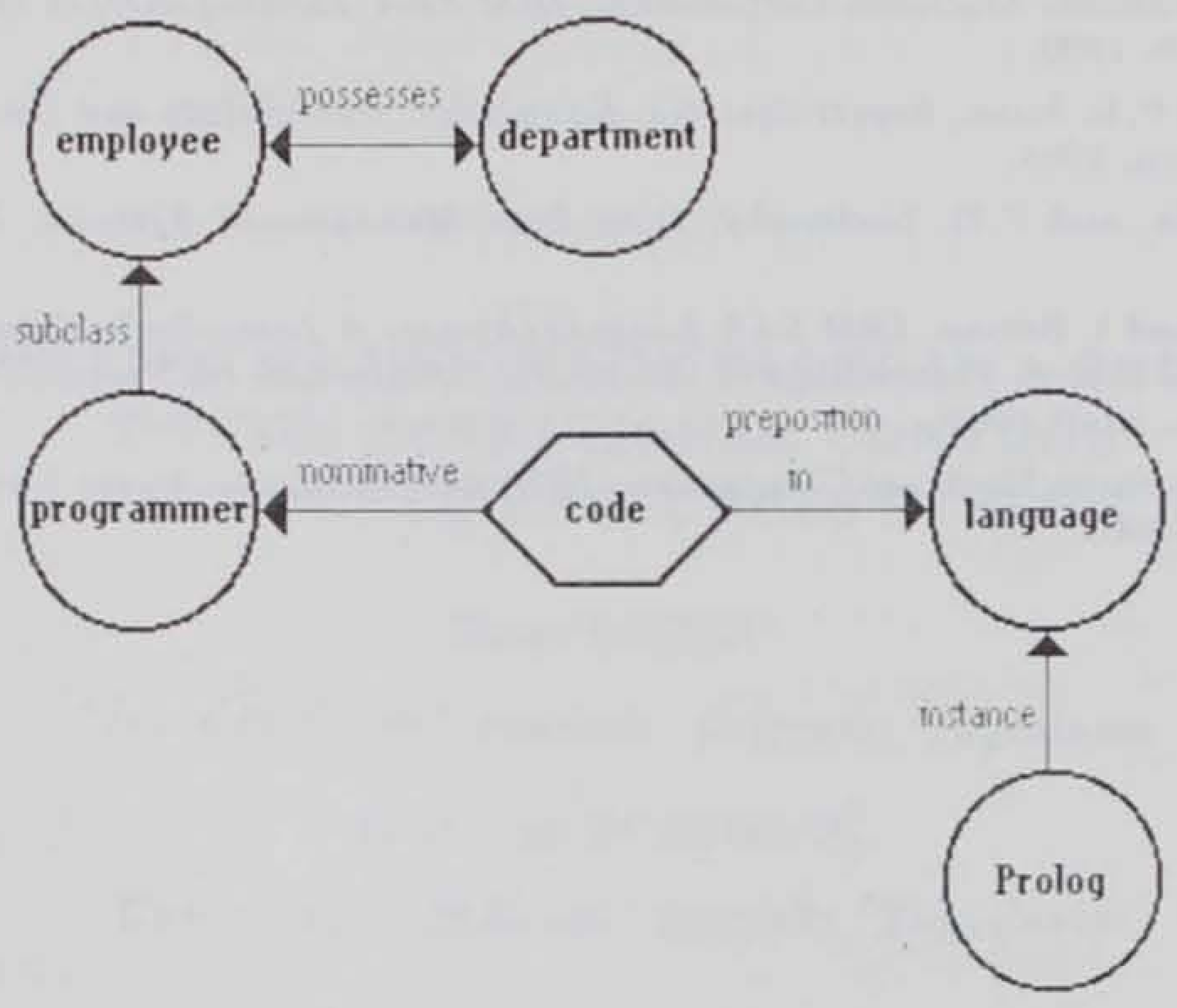

Figure 4. Correct question based on the conceptual schema is: *List all departments which have programmers that code in Prolog.*

• **H6 function.** The H6 function uses the Prolog facility to manipulate strings and lists. The input string which presents a query is transformed into a list of words. In order to find terms, composed of one or more words, which belong to the conceptual schema, the list is examined in a recursive manner.

5. CONCLUSION

This paper presented a Help function for a natural language interface to relational databases. The modular design of the proposed functions and the choice of Prolog as a programming language give many advantages like: rapid prototyping, additional tuning and robustness of code.

The design and prototyping was made for the English language. It seems that very similar Help can be made for other natural languages by changing the patterns of natural language fragments.

Acknowledgement. This work was performed while the author was with IBM Nordic laboratory in Lidingö, Stockholm, arranged through IAESTE — the International Association for the Exchange of Students for Technical Experience. Special thanks for helpful advice and valuable suggestions go to Dr. Mohammad Sanamrad — Project Manager in Natural Language Department, Brian White and Jane Brown — who provided the linguistic help.

REFERENCES

[1] A. Barr, E. A. and Feigenbaum, eds., *The Handbook of Artificial Intelligence, Volume I*, Pitman Books Limited, 1981.

[2] International Business Machines Corporation: *IBM SAA LanguageAccess General Information*, GH19-6680, 1990.

[3] I. Graham, and P. L. Jones, *Expert Systems, Knowledge, Uncertainty and Decision*, Chapman and Hall, London, 1988.

[4] D.C. Tsichritzis, and F.H. Lochovsky, *Data Base Management Systems*, Academic Press, 1977.

[5] M. Sanamrad and I. Bretan, *IBM SAA LanguageAccess. A Large-Scale Commercial Product Implemented in Prolog*, Proceedings of the 1st Int. Conference on Practical Applications of Prolog, London, April 1992.

[6] International Business Machines Corporation: *IBM SAA Language-Access Customization Tool User's Guide*, 1990.