Spanish Natural Language Interface for a Relational Database Querying System *

Rodolfo A. Pazos R.[1], Alexander Gelbukh [2], J. Javier González B.[3], Erika Alarcón R.[3], Alejandro Mendoza M.[3], A. Patricia Domínguez S.[3]

[1] Centro Nacional de Investigación y Desarrollo Tecnológico, Mexico
[2] Computing Research Center (CIC), National Polytechnic Institute (IPN), Mexico
[3] Instituto Tecnológico de Ciudad Madero, Mexico
pazos@sd-cenidet.com.mx, gelbukh@cic.ipn.mx,
{jjgonzalezbarbosa,erika_2k,amejia_jnm}@hotmail.com,
patricia5@infosel.net.mx

Abstract. Fast growth of Internet is creating a society where the demand on information storage, organization, access, and analysis services is continuously growing. This constantly increases the number of inexperienced users that need to access databases in a simple way. Together with the emergence of voice interfaces, such a situation foretells a promising future for database query systems using natural language interfaces. We describe the architecture of a relational database querying system using a natural language (Spanish) interface, giving a brief explanation of the implementation of each of the constituent modules: lexical parser, syntax checker, and semantic analyzer.

1 Introduction

Providing simple access to database systems has become one of the problems of great interest for users and developers of database querying systems, especially since Internet became popular. Users are looking for systems that facilitate the use of databases, trying to reduce the time and effort required for learning how to use them [13]. To satisfying this demand, the developers design and implement friendlier interfaces. One of the most popular types of such interfaces is natural language interface.

Traditional natural language processing (NLP) has not yet provided database-querying systems with all the benefits that can be obtained from it (especially for the Spanish speaking community). Limited scope systems have been developed, mainly for a few languages such as English; some examples of these systems are California Restaurant Query, Expedia Hotels, GeoQuery [16], Hollywood [5], JobQuery [14], Masque/SQL [1], SQ-HAL [12], and SystemX [3].

Despite the long history of NLP and the numerous techniques developed, most of the work has been carried out for English; the situation with Spanish language is far behind as compared with other major languages. This is worrisome, especially taking

* Work done under partial support of CONACyT and SNI (Mexico) and RITOS-2 (CYTED).

into account the spread and large population of Spanish-speaking community around the world (390 million), and therefore the potential market for Spanish NLP tools.

Most of the NLP work done so far for Spanish is devoted to applications that have little or no relation with natural language interfaces to databases (NLIDBs), for example: text interpretation [6], spelling correctors [11], and processing tools [7, 9]. Only a few projects have dealt with NLIDBs, being the most important the following: GNBD [15], SISCO [10], and Sylvia-NLQ [4].

Most of NLIDB projects for Spanish have approached database querying from the AI perspective. Thus, they have focused mainly on deductive databases and overlooked the vast majority of existing database systems: relational databases, warehouses, and object-oriented databases. The aim of the project described in this paper is to design and implement a relational database querying system with a natural language interface for Spanish language.

The paper is organized as follows. First, the general architecture of the system is presented. Then, the implementation of the lexical parser, syntax checker, semantic analyzer, and the result presentation module is described. Then, domain customization issues are discussed. Finally, some concluding remarks are given.

2 Architecture

The architecture of the querying system consists of two main modules: Database Query module and Domain Customization module (Figure 1).

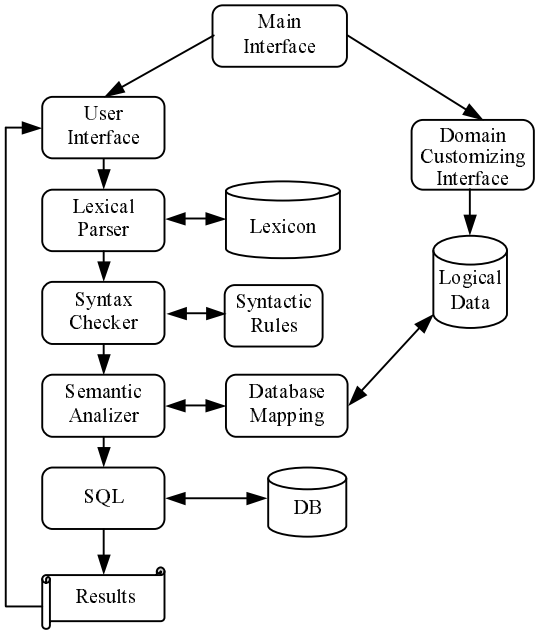

Fig. 1. Architecture of the querying system.

The *Database Query module* is the main part of the system. The user types in the query through the user interface submodule. It is processed by the lexical parser, syntax checker, and semantic analyzer, which translate it in an SQL expression. The results of the SQL query are passed back to the user again through the user interface submodule.

The *Domain Customization module* is used to customize and maintain the data and dictionaries used by the NLP modules of the system. It interacts with the system administrator through the domain customization interface submodule.

A brief description of the implementation of the main components of the query module is presented below; the customization module is discussed later.

3 Lexical Parsing

Lexical parsing is in charge of reading the input sentence (query) and dividing it into words, which are minimal meaningful units also known as tokens. Then, it identifies the tokens in the lexical dictionary (lexicon) and retrieves the information about each token. Finally, it passes the tokens with the associated information on to the syntax checker and semantic analyzer.

The lexical parser interacts with the lexicon, which contains all words that are expected to appear in user's queries. For each word, the dictionary includes its part of speech (noun, verb, article, etc.), gender, and number; the format of this dictionary is independent of the parser code. For this project, a database was used for storing each word, its type (part of speech) represented by an integer number according to Table 1, and subtype (the combination of gender and number, when applicable).

Table 1. Representation of the syntactic category.

Word	Type	Word	Type
Interrogative adjective	1	Preposition	8
Article	2	Conjunction	9
Noun	3	Pronoun	10
Adjective	4	Punctuation mark	11
Auxiliary verb	5	Relational operator	12
Verb	6	Error	*
Adverb.	7		

4 Syntax Checker

Wrongly understood queries can cause incorrect and confusing results to be delivered to the user, which is potentially dangerous since the user may rely on them to make important decisions. Thus, of crucial importance for reliability of natural language interfaces is verification of that the query has the expected pattern that the system can understand correctly.

The syntax checker verifies that the queries have the expected structure. It checks the relationship among words according to their role in the sentence to make sure that the input sentence satisfies a predefined order or pattern.

For this purpose, the checker verifies if the string of tokens received from the lexical parser can be generated by the grammar. It informs the user if any syntactic error is detected and prevents the suspicious query from further processing.

After evaluation of existing types of syntax checkers, a state transition matrix approach [6] has been considered the most convenient because of the following reasons: the implementation of the algorithm is simple since it handles uniformly all the grammar rules; it permits to modify and add new rules to the grammar without having to change the algorithm code. The use of a state transition matrix permits the expansion of a grammar just by adding new states or syntactic categories to the matrix.

The grammar we currently use has the form of a simple bigram check: each word of the sentence is checked to be compatible with the immediately following word. Table 2 shows a fragment of the state transition matrix currently used for syntax checking. Of course, in the future a more sophisticated grammar can be implemented using the same program code but a different matrix.

Table 2. Fragment of the state transition matrix.

State	I.P.	Art.	Noun	Adj.	Aux.V.	V.	Adv.
	1	2	3	4	5	6	7
0	1	*	*	*	*	6	*
1	*	*	3	*	5	6	*
2	*	*	3	*	*	*	*
3	*	*	3	4	5	6	7
4	*	*	*	*	5	6	*
5	*	2	*	*	*	6	*
6	*	2	3	*	*	*	7
7	*	*	*	4	*	*	7

* Error

Since each syntactic category is represented by an integer number, the interaction with the state transition matrix is realized through this number. Using the state transition matrix, the syntax checker determines if the first word of the sentence is an interrogative pronoun or a verb (state 0, first line of the table); if so, the current state is updated and the next iteration is performed. For example, considering the following question:

¿Cuántos créditos tiene Erika Alarcón? (1)

'How many units has Erika Alarcón?', the checker output is the following:

cuántos 'how many'	créditos 'units'	tiene 'has'	Erika Alarcón
1	3	6	3

If a syntactic error is found, i.e., the current word is not expected in the present state, the program will display a message indicating the error type and will stop the parsing process, since it will not be able to determine the next state to move into.

Additionally, the bigram method is used to verify the morphological compatibility of each pair of adjacent words. Lexical parser supplies with each input word its gender (feminine, masculine, neuter) and number (singular, plural, neuter); the value "neuter" us used when the gender is not defined (*estudiante* 'student') or not applicable (e.g., for adverbs), and similarly for the number. If two adjacent words have incompatible genders (feminine versus masculine) or numbers (singular versus plural), an error is reported.

5 Semantic Analyzer

Upon successful completion of syntax checking, the semantic analyzer extracts the meaning of the sentence and generates a logical structure. In this module, the sentence is analyzed again to identify the key words with which the system can interpret the query; in this way the analyzer obtains the meaning of the query.

This process is carried out using logical predicates [1] constituting the keystone of the semantic analyzer, since they permit to determine the words involved in the query for its successful execution. For this, each noun is looked up in a table where the logical predicates are stored. For example, consider the query (1) and the logical predicates shown in Table 3; the matching predicate for the question is *créditos* 'units', which needs a student *name* or *id number* as a parameter.

If a match occurs, the sentence is explored once more to obtain the data needed by the parameters of the logical predicate. When the parameter type is not explicit in the query, the analyzer looks for the parameter data in the database, in order to determine its type. This is done by finding the field name that corresponds to each of the possible parameters associated to the logical predicate.

Table 3. Fragment of the logical predicate information.

Predicate	Parameters	Predicate	Parameters
id number	name	*créditos* 'units'	name
especialidad 'major'	name	*créditos* 'units'	id number
especialidad 'major'	id number	*promedio* 'average mark'	name
nombre 'name'	id number	*promedio* 'average mark'	id number

In the example (1), the parameter data is *Erika Alarcón*. In this case the parameter type is determined by accessing first the word-to-field mapping information shown in Table 4, which indicates that the possible database fields associated to the parameter data are *STUD_ID* and *STUD_NAME*. Then the analyzer generates the following query to find *Erika Alarcón* in the field *STUD_NUM*:

```
SELECT COUNT(*)
FROM STDATA
WHERE STUD_ID = 'Erika Alarcón'
```

In this case the database manager will not find a student whose *id number* is *Erika Alarcón*, and thus the analyzer will issue a similar query to check if there is any

student with this name (using STUD_NAME instead of STUD_ID), which returns a positive answer.

Table 4. Fragment of lexicon words to database field mapping information.

Word	DB Field	Table	Database
id number	STUD_ID	STDATA	ACAD
major	STUD_MJR	STDATA	ACAD
name	STUD_NAME	STDATA	ACAD
units	STUD_UNITS	STDATA	ACAD
average mark	STUD_AVG	STDATA	ACAD

Each time a matching predicate is found and its corresponding parameter data is obtained, it is written in a structure called *logical data storage structure*. This structure holds the field names whose information is requested by the user, the database name, the table name, and the search conditions. The data structure for the example (1) is shown in Table 5. If no matching predicate is found, the system displays to the user a message asking for a reformulation of the query.

Table 5. Logical data storage structure.

Requested Data	Condition Parameter	Parameter Data	Table
STUD_UNITS	STUD_NAME	'Erika Alarcón'	STDATA

6 Results Module

The results module translates the natural language sentence into the target formal language. In this project, SQL (Structured Query Language) has been chosen as the target language for two reasons: (1) SQL is the most widely used query language for commercially available relational database engines and (2) this will enable the system to be used with many different databases.

This module accesses the logical data storage structure and extracts the field names whose data are requested and the search conditions. Using this information, the module constructs the SQL statement, which is executed with the selected database, and the result presented to the user. The SQL query generated for the example (1) is:

```
SELECT STUD_UNITS
FROM STDATA
WHERE STUD_NAME = 'Erika Alarcón'
```

7 Domain Customization

This module was implemented in order to make the querying system independent of the underlying database, thus enabling the system to issue queries to different databases or tables without any need to modify the system software.

The domain customization module was implemented as a Java servlet so that it can be used over the Internet or intranet. This module provides an interface that permits the system administrator to establish a relationship between the words of the lexicon and the database fields, since the field names are usually abbreviated (e.g., word: *major*, field: STUD_MJR). Therefore, with this information the semantic analyzer will know which field corresponds to a given word in the query. Additionally, the system administrator can add words to the lexicon and define their syntactic category, gender, and number.

The domain customization module can only be used by an advanced user (administrator); otherwise, if an unauthorized person uses this module and enters erroneous data, the system could fail or generate incorrect results. To prevent this situation, access to this module is restricted by a password.

8 Final Remarks

Since 1997 our group has been implementing software tools that facilitate inexperienced users to query databases. One of these tools is a QBE (Query by Example) graphical interface [17] for querying relational databases via Internet. This interface allows simultaneous access to two or more databases and the formulation of queries that involve a combination of information from all open databases [8]. In order to facilitate the formulation of queries involving joins of two or more database tables, a new interface (EzQ) is being developed [2]. This new interface will permit the users to formulate such queries even if they are not familiar with the concept of join (difficult to understand for users that are not information systems professionals).

A reflection about graphical interface limitations has revealed that it would be extremely difficult to further increase the ease of use of the QBE interfaces to meet the needs of the majority of the Internet users, which are not information systems professionals. Therefore, in order to achieve this goal we considered it necessary to include another technology: natural language interfaces.

The work described in this paper constitutes the first step for providing the current system with an interface, as friendly as possible, that will involve both graphical and natural language interfaces. Unlike most NLIDB projects for Spanish language, we are approaching NLIDB issues from the information systems perspective.

Our current version of Spanish grammar (syntax checker) handles 10 different simple and compound sentence patterns. The semantic analyzer has been tested with a specific domain (academic department database); however, it is implemented in such a way that it can be used for other domains just by changing the information supplied through the domain customization module.

References

1. Androutsopoulos, I. *Intefacing a Natural Language Front-End to a Relational Database*. M.Sc. dissertation, Dept. of Artificial Intelligence, Univ. of Edinburgh; www.dai.ed.ac.uk/papers/documents/mt92103.html.

2. Carreón V., G. *Herramienta para Consultas EzQ para Multibases de Datos en Internet*. M.Sc. dissertation (to be published), Computer Science Dept., National Center for Research and Technology Development (CENIDET), Cuernavaca, Mexico.
3. Cercone, N., P. McFetridge, F. Popowich, D. Fass, Ch. Groeneboer, G. Hall. *The SystemX Natural Languaje Interface: Design, Implementation and Evaluation*. Technical report, Centre for Systems Science, Simon Fraser University, British Columbia, Canada, 1993; www.cs.sfu.ca/research/groups/NLL/4.html#4.1.
4. Computational Linguistics Laboratory. *Project Sylvia-NQL*; www.lllf.uam.es/proyectos/sylvia.html.
5. ELF Software. *Hollywood Tutorial*. Last update 1998; www.elf-software.com/Docs/Hollywood.htm.
6. Flores V., J.M., J.M. Matadamas H. Sistema de Interpretación de Texto. *Proc. 7th. International Congress on Computer Science Research*, Technological Institute of Cd. Madero, Tampico, Mexico, Oct. 2000, pp. 73-81.
7. González, J.C., J.M. Goñi., A.F. Nieto. ARIES: a ready for use platform for engineering Spanish-processing tools. *Digest of the Second Language Engineering Convention*, London, U.K., 1995, pp. 219-226; wotan.mat.upm.es/~aries/papers.html.
8. May A., A., R.A. Pazos R., J. Pérez O., R. Ortega I. Intermediario para acceso a multibases de datos en Internet. *Proc. Simposio Español de Informática Distribuida*, Univesity of Vigo, Ourense, Spain, 2000, pp. 259-267.
9. Monedero, J., J.C. González, J.M. Goñi, C.A. Iglesias, A.F. Nieto. Obtención automática de marcos de subcategorización verbal a partir de texto etiquetado: el sistema SOAMAS. *Proc. XI Congreso de la Sociedad Española para el Procesamiento del Lenguaje Natural (SEPLN'95)*, Bilbao, Spain, 1995, pp. 241-254; wotan.mat.upm.es/~aries/ papers.html.
10. Palomar, M., L. Moreno, A. Molina: SISCO: Sistema de interrogación en lenguaje natural a una base de datos geográfica. *Procesamiento del Lenguaje Natural*, No. 14, 1993.
11. Rodríguez S., J. Carretero. Corrector ortográfico de libre distribución basado en reglas de derivación. *Primer Encuentro del Grupo de Usuarios de TeX Hispanohablantes (EGUTH'99)*, 1999, pp. 44-52; www.datsi.fi.upm.es/~coes/publications.html.
12. Ruwanpura, S. SQ-HAL: *Natural language to SQL translator*; www.csse.monash.edu.au/hons/projects/2000/Supun.Ruwanpura.
13. Sethi, V.: *Natural language interfaces to databases: MIS impact, and a survey of their use and importance*. Technical report, Graduate School of Business, University of Pittsburgh, Pittsburgh, E.U.A.
14. Thompson, C.A., R.J. Mooney, Lappoon, R. Tang. Learning to parse natural language database queries into logical form. *Proc. ML-97 Workshop on Automata Induction, Grammatical Inference, and Language Acquisition*, Nashville, U.S.A., 1997.
15. Validation and Business Applications Group. *PASO – PC315 PROJECT, Generator of natural language databases interfaces*; www.vai.dia.fi.upm.es/ing/projects/paso.htm.
16. Zelle, J.M., R.J. Mooney. Learning to parse database queries using inductive logic programming. *Proc. Thirteenth National Conference on Aritificial Intelligence*, Portland, U.S.A., 1996, pp. 1050-1055.
17. Zloof M.M. Query by Example: a database language. *IBM Sys. Journal*, Vol. 16, No. 4, 1977, pp. 137-152.