© International Journal of Combinatorial Optimization Problems and Informatics, Vol. 6, No. 2, May-Aug 2015, pp. 28-42. ISSN: 2007-1558.

Semantic representations for knowledge modelling of a Natural Language Interface to Databases using ontologies

Juan J. González, Rogelio Florencia-Juarez, Héctor J. Fraire, Rodolfo A. Pazos, Laura Cruz-Reyes, Claudia Gómez
Instituto Tecnológico de Ciudad Madero
jjgonzalezbarbosa@hotmail.com, rogelio.florencia@live.com.mx, automatas2002@yahoo.com.mx, r_pazos_r@yahoo.com.mx, lauracruzreyes@itcm.edu.mx, claudia.gomez@itcm.edu.mx

Abstract. Despite the large number of Natural Language Interfaces to Databases (NLIDB) that have been implemented, they do not guarantee to provide a correct response in 100% of the queries. In this paper, we present a way of semantic modelling the elements that integrate the knowledge of a NLIDB with the aim of increasing the number of correctly-answered queries. We design semantic representations in order to: *a*) model any relational database schema and its relationship with the natural language and *b*) add metadata to natural language words to enable our NLIDB to interpret natural language queries that contain superlatives. We configured our NLIDB in a relational database that we migrated from Geobase and used the Geoquery250 corpus to evaluate its performance. We compare its performance with the interfaces ELF, Freya and NLP-Reduce. The results indicate that our proposal allowed our NLIDB to obtain the best performance.

Keywords: Natural Language Interface to Databases, Semantic Representations, Knowledge, Ontology.

1 Introduction

One of the major sources of information is databases, which are collections of related information, stored in a systematic way for modelling a part of the world [1]. Sometimes, it is necessary to use a database query language to access the information stored in a database. To use database query languages like Structured Query Language (SQL), people require expertise which most of them do not have. Therefore, it became necessary to design software applications to translate from natural language queries into SQL queries, allowing users to easily access the information contained in a database. Such applications appeared in the 60s [2] and were called Natural Language Interface to Databases (NLIDB).

To translate from natural language queries into SQL queries, a NLIDB should adequately model the necessary knowledge in order to correctly interpret natural language queries. With this in mind, different architectures of NLIDBs have been proposed, and Androutsopoulos [2] classifies them into four types: *Pattern-matching* employs a simple technique based on patterns or rules that apply to natural language queries. *Syntax-based* systems syntactically parse natural language queries and the resulting parse tree is directly mapped to an expression in some database query language. *Semantic grammar* systems are similar to syntax-based systems. The difference is that the grammars categories do not necessarily correspond to syntactic concepts. *Intermediate representation language* systems first transform from natural language queries into intermediate logical queries expressed in some internal meaning representation language, and subsequently, this representation is translated into a SQL query. To the present day, despite the number of NLIDBs implemented, they do not guarantee 100% of correctly-answered queries [3], so NLIDBs are still an open research topic.

In this paper, we present a way of semantic modelling the elements that integrate the knowledge of a NLIDB, with the aim of increasing the number of correctly-answered queries. To model the knowledge, we designed semantic representations in the Ontology Web Language (OWL) in order to: *a*) model any relational database schema and its relationship with the natural language, which it allows interpreting natural language queries and generating the corresponding SQL queries and *b*) add metadata to natural language words to enable our NLIDB to interpret natural language queries that integrate superlatives, which allows to increase the linguistic coverage of our NLIDB. In order to make the knowledge customization in our NLIDB easy for the user, we implemented these semantic representations in a Customization Module (CM). To assess the performance of our

Received Jun 2, 2015 / Accepted Jun 11, 2015

NLIDB, we configured it in a relational database that we migrated from Geobase and we used the Geoquery250 corpus [4, 5]. We compared its performance with the interfaces, ELF, Freya and NLP-Reduce. The results indicate that our proposal allowed our NLIDB to obtain the best performance.

This paper is structured as follows: Section 2 presents related works; section 3 describes the design of the semantic representations used to model the knowledge of our NLIDB; section 4 shows the description of the CM; section 5 provides an example of the use of the semantic representations; section 6 presents the experimental results and section 7 presents the conclusions drawn from this research.

2 Related Works

The NLIDBs research topic began to develop around the 60s. The first interfaces were designed to be used in a particular database. This feature made them difficult to be used with other databases. One of the pioneer systems was LUNAR, implemented by Woods [6]. It is a system that answered questions about samples of rocks brought back from the moon. Proposed by Codd, RANDEZVOUS [7] used clarification dialogues to help users to generate queries. Hendrix [8] implemented LADDER using a semantic grammar technique that combines syntactic and semantic processing to interpret queries.

In the 80s, researchers focused on achieving the portability of the NLIDBs, by developing interfaces such as CHAT-80 [9], TEAM [10], PARLANCE [11], JANUS [12], among others. The techniques used in most NLIDBs reported in the literature are pattern matching, syntax-based systems, semantic grammar systems, and intermediate-representation language systems. For more information on these techniques see [13].

There are interfaces that seek to automate the configuration of a NLIDB in a database by using techniques such as Learning Semantic Parsing, processing a natural language sentence into a complete formal meaning representation or logical form. Examples of these interfaces are CHILL [4], KRISP [14], SCISSOR [15], and WASP [16]. Although these interfaces demonstrated good results, they are not related to our research since those interfaces require a training set, generally designed by an expert. Unlike those interfaces, our approach is based on knowledge. Next, we mention other interfaces that follow the same approach.

MASQUE/SQL [17] uses a lexical dictionary and a semantic dictionary for storing its knowledge. The lexical dictionary contains the possible forms of the words that the user could use to formulate a query. The semantic dictionary stores the meaning of each word in terms of predicate logic, where each predicate is related to a table or view from the database or a SQL statement. This interface also has a domain editor that helps users to describe entities in the database, using an "is-a" hierarchy. The translating process of MASQUE/SQL transforms users' queries into expressions in Logical Query Language (LQL), this LQL expression is translated into a SQL query. The MASQUE/SQL configuration is semi-automated, since dictionaries are created automatically by interacting with the domain editor.

CoBASE [18] uses a semantic graph to represent its knowledge. This graph is generated semi-automatically and models database objects, allowing the user to define their relationships. CoBASE has a query formulator that allows the user to enter query data. Subsequently, candidate queries are formulated using a graph search technique based on the input data. In case of ambiguity, the candidate queries are categorized according to probabilistic information, allowing the user to select the correct query.

ELF [19] is a commercial interface developed by ELF Software Co. This interface automatically generates a lexicon from the database schema. The user can exclude tables and/or relationships that he/she thinks are not necessary before the interface generates the lexicon. The user can also choose the columns whose data will be stored in the lexicon to identify values in the user's query. Since these stored data are a copy, ELF is no able to identify the new data introduced to the database as a value, after creating the lexicon. ELF creates a semantic dictionary of words that compose the table and column names. Synonyms of these words are also added.

PRECISE [20] stores its knowledge in a lexicon, which is generated automatically by extracting values, attributes, and names of the relations of the database. The lexicon can be added manually with synonyms, and prepositions and also allow specifying which elements of the database can be related to a preposition or verb. The translation process of PRECISE is based on semantic information stored in the lexicon and on the syntactic parsing information of the query, performed by the Charniak parser, which was trained with a set of 150 queries, whose words were labeled by their POS tag. The strength of the translation process is

based on the ability to match the keywords in a sentence to the corresponding database structures. The translation process reduces the potential semantic interpretations to a graph matching problem solved by using the MaxFlow algorithm. The weakness of the interface is that it reaches a low recall rate.

Like the NLIDBs described above, we also extract information about the relational database schema. Unlike MASQUE/SQL, ELF, and PRECISE which stores this information in a lexicon and CoBASE that stores it in a semantic graph, we store this information in an ontology by using the proposed semantic representations. In order to generate the SQL query, unlike MASQUE/SQL that uses predicate logic, we use the mappings between words and the database schema elements stored in the ontology. Unlike CoBASE that has a query formulator which helps users enter the query, our interface allows users to enter the query as free text. To identify elements in the user's query, we search values in the relational database in a similar way to ELF. The difference is that ELF searches for values in a copy of the data stored in its lexicon, and we search for values directly on the database. The disadvantage of searching for values in a copy makes it impossible for ELF to identify values introduced to the database after creating its lexicon. Another difference between these NLIDBs is that we can easily increase the language coverage of our interface by using the metadata modelled by using the proposed semantic representations.

The popularity of NLIDBs has been changing towards a new type of interfaces which find answers on ontologies [21]. Ontologies are used in the Semantic Web as a suitable representation of knowledge available on the Web [22]. These interfaces are easily portable to different domains, they can access and combine resources on the web and they are able to reason about structured data. Among the interfaces of this type we can mention the following:

GINSENG (Guided Input Natural Language Search Engine) [23] loads all knowledge bases located in a predefined search and generates grammar rules from each knowledge base loaded. These dynamic rules allow GINSENG to suggest alternative words to users when they are formulating a query, reducing in this manner the number of incorrect entries. Static grammar rules provide the basic structures for queries, and they are also used to transform a query into the RDQL query language. This interface also allows adding synonyms to labels used in the ontologies.

ORAKEL [24] has a lexicon composed by a general lexicon and a domain specific lexicon. The general lexicon specifies the meaning of the words that could be used in different domains, such as prepositions, determiners, and pronouns. The domain specific lexicon is composed by an ontological lexicon and a lexicon for mapping ontology relations of words. The ontological lexicon is generated automatically from the domain ontology. The lexicon for mapping ontology relations for words is created manually and contains mappings of sub-categorization frames to ontology relations. Sub-categorization frames are essentially linguistic argument structures (e.g. verbs with their arguments, nouns with their arguments) created by the domain experts. ORAKEL transforms the user's query in First-Order-Logic formulas, which are transformed into an OWL or F-Logic query.

NLP-REDUCE [25] creates automatically a lexicon by extracting from the domain ontology explicit and inferred information. This extracted information is added with synonyms extracted from WordNet. NLP-REDUCE does not perform a complex processing of natural language, it just tries to relate words to expressions stored in an ontology with the aim of formulating a SPARQL query.

PANTO (Portable Natural Language Interface to Ontologies) [26] has a Lexicon Builder that automatically extracts entities out of the ontology selected as the underlying knowledge base in order to build a Lexicon. The Lexicon is mainly designed for making sense of words in natural language queries and mapping them to entities in the ontology. WordNet is utilized to find synonyms to enlarge the vocabulary of the ontology. Once the user inputs a natural language query, PANTO invokes the Parser to produce the parse tree, which is then transferred to the Translator. The Translator transforms the parse tree into SPARQL.

FREyA (Feedback, Refinement and Extended Vocabulary Aggregation) [27] combines the syntactic parsing with the knowledge contained in ontologies in order to interpret a query. FREyA uses heuristic rules in the syntactic information of the query for identifying possible ontology concepts as well as some other elements. The possible ontology concepts identified are related to ontology concepts. If it is not possible, FREyA engages the user in dialogues for disambiguation. If the ambiguity prevails, FREyA engages the user in dialogues for relating a concept to some of the concepts suggested by the interface. The related concepts are used to form a SPARQL query.

A difference between these interfaces to ontologies and our NLIDB is that we do not use ontologies as a database. We use ontologies to model the knowledge of a NLIDB in order to interpret natural language queries and to generate the corresponding SQL queries.

3 Semantic representations

We believe that, to obtain a good performance, a NLIDB must have appropriate and sufficient knowledge in order to interpret natural language queries and to generate the corresponding SQL queries. To generate SQL queries, the NLIDB must have knowledge about the schema of the database to be queried. To interpret natural language queries, the NLIDB must know the relationship between the natural language derived from the domain of the database and the knowledge about the database schema. However, since databases can have different schemas among them and since the natural language depends on the domain of that database, a dynamic and flexible mechanism for knowledge representation is required. For this purpose, in order to semantically model these elements, we designed semantic representations by using the Ontology Web Language (OWL). In our approach, an ontology is a way to explicitly model a semantic representation of any database schema and its relationship with the natural language. This way, our NLIDB has knowledge of both natural language (used to interpret user queries) and database schema (used to generate the appropriate SQL queries).

In the following sub-sections, we describe the design of the semantic representations used to model the elements that integrate the knowledge of our NLIDB: *a*) the database schema (sub-section 3.1), *b*) the natural language and its relationship with the database schema (sub-section 3.2) and *c*) the added metadata to natural language words to interpret natural language queries that integrate superlatives (sub-section 3.3).

3.1 Semantic representation of the database schema

We named *Semantic Database Schema* (SDbS) the semantic representation of any database schema. The SDbS is dynamically generated by the CM in two steps based on the database schema. In the first step, the structure of all tables is modelled and in the second step, table relationships are modelled. In order to model the structure of a table, we defined in OWL the classes and properties described in Table 1.

Table 1. Classes and properties defined for modelling of the structure of the tables

Name	OWL Component	Description
CTable	Class	Instantiates each table name as an individual
CColumn	Class	Instantiates each column name as an individual
hasColumn	Object Property	Relates CTable individuals to CColumn individuals
isColumnOf	Object Property	Inverse object property of hasColumn
DataType	Datatype property	Stores the data type name of each column

Each table name is modeled as *CTable* individual. The columns of each table are modeled as *CColumn* individuals. Subsequently, these *CColumn* individuals related to their corresponding CTable individuals by using the *hasColumn* object property. The data type name of each column is added to its owner CColumn individual by using the *DataType* data property. Figure 1 shows an example of how the components described in Table 1 were used to model the State table of Geobase.

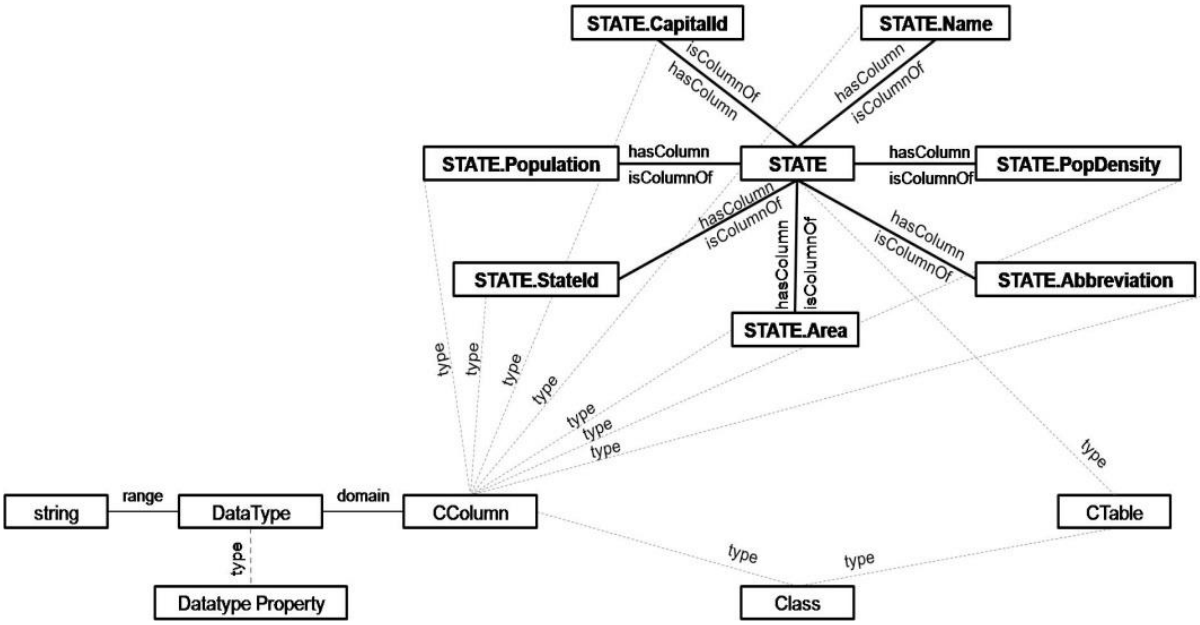

Fig. 1. Semantic representation of the State table.

After modeling the structure of all tables, table relationships must be modeled. In order to model table relationships, we defined in OWL the *relatesColumnToTable* object property. This object property relates a CColumn individual with a CTable individual. CColumn individual must be the primary key of one table and the CTable individual must be another table and vice versa. A proper modeling of table relationships is key to perform table joins when forming SQL queries. Figure 2 shows an example of how to relate the *State* individual to the *Mountain* individual through the *State.StateId* and *Mountain.StateId* CColumn individuals.

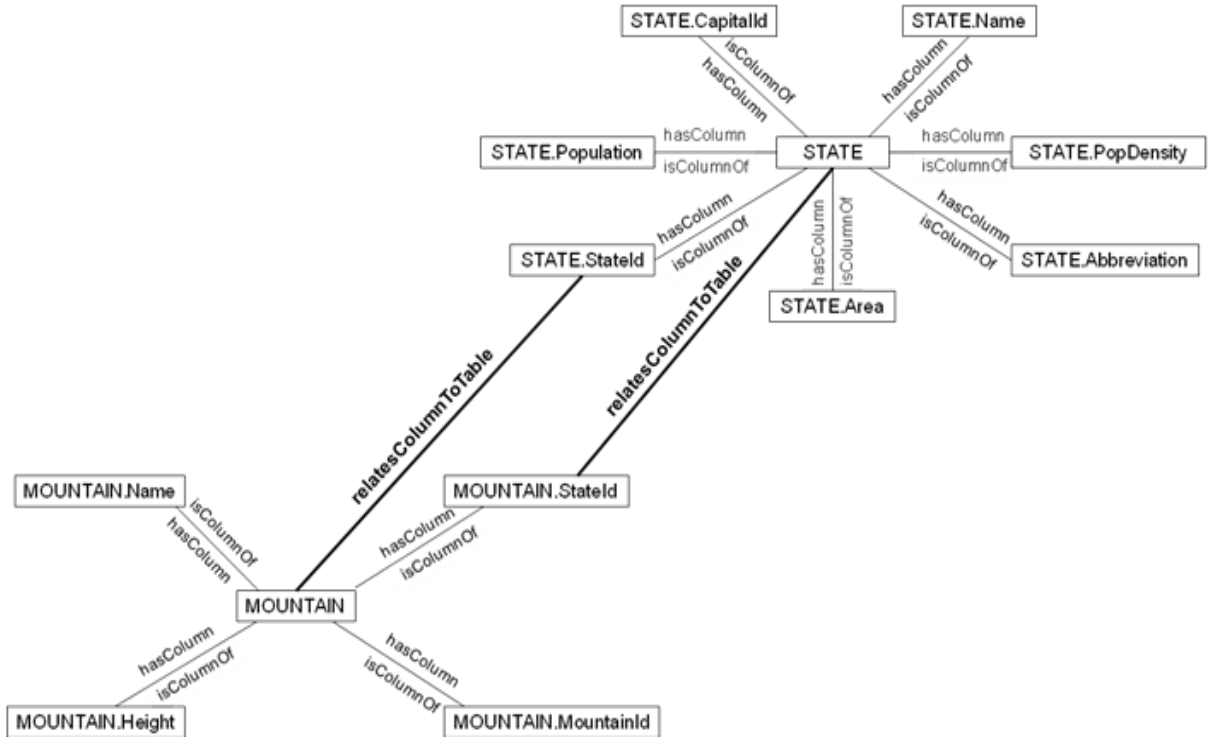

Fig. 2. Semantic representation to relate the State individual with the Mountain individual.

After modeling the table relationships, the information modeled in the SDbS is saved as an OWL file. It is noteworthy that all the processes described in this sub-section are dynamically generated by the CM whose functioning is described in section 4.

3.2 Relationship between the Semantic Database Schema and the natural language

After modeling the SDbS, the user must define the relationship between the SDbS and the natural language. For this purpose, the user must identify tables and/or columns of the database that store information that could be queried by other users and (s)he must specify a set of natural language words that describe them. Subsequently, (s) he must relate each of these words with the corresponding CTable and CColumn individuals in the SDbS. We have named this relation, *mapping*. Mappings are used to identify tables and columns explicitly mentioned in users' queries which, subsequently, are used to form the SQL query. In order to model mappings, we defined in OWL the classes and properties described in Table 2.

Table 2. Classes and properties defined for modeling of mappings

Name	OWL Component	Description
CToken	Class	Instantiates each word introduced by the user as individual
mappingObject	Class	Relates CToken individuals to CTable/CColumn individuals

Each natural language word specified by the user is represented as a *CToken* individual. In order to relate a CToken individual with CTable and/or CColumn individuals we defined in OWL the *mappingObject* object property. This object property relates a CToken individual with CTable and CColumn individuals. Figure 3 shows an example of how mappgins are modeled. In the example, it can be seen that the *density* and *population* words are mapping to the State.PopDensity individual; the *citizens*, *people*, *inhabitants*, *population*, *populations* are mapping to the State.Population individual, etc.

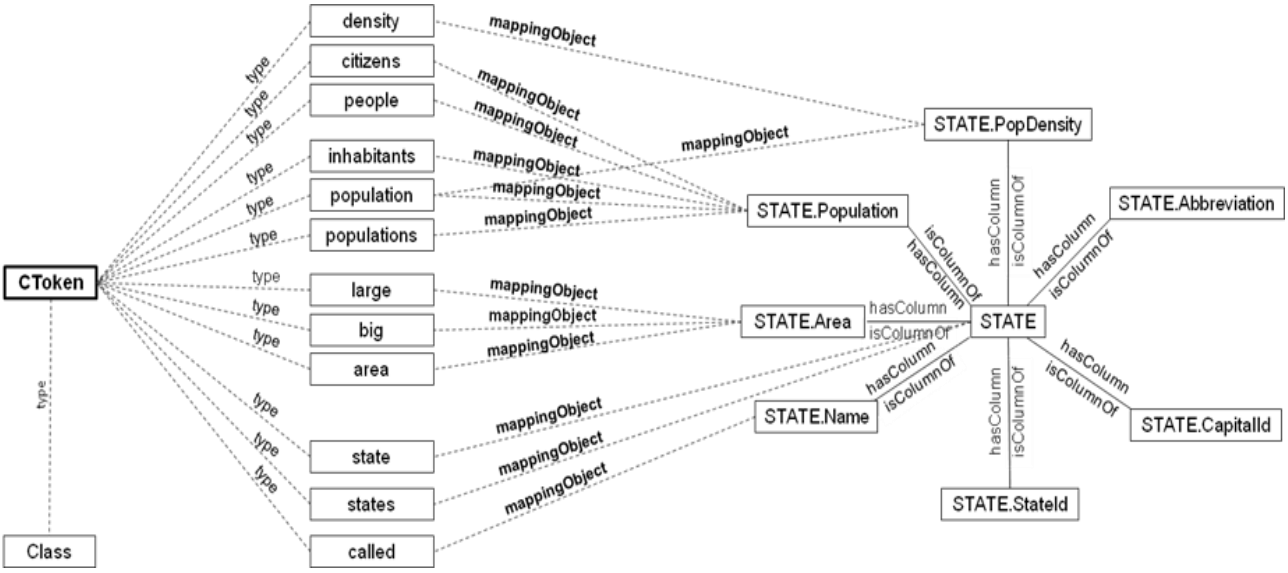

Fig. 3. Semantic representation to model the relationship between natural language and database schema (SDbS).

3.3 Superlatives

In the context of NLIDBs, superlatives are words used to refer to the best or worst value stored in a column of the database. For example in the GeoQuery250 corpus, to answer the query "*What is the longest river in Mississippi*?" the NLIDB must know that the *longest* word is a superlative that refers to the maximum value stored in the *River.Length* column for the indicated state. The information of the superlatives must be defined by the user when configuring the NLIDB by specifying the columns that could be related to the superlative (River.Length in this case) and by indicating whether the superlative refers to a maximum or

minimum value (maximum in this case). In order to model superlatives, we defined classes and properties in OWL (see Table 3) with the aim of adding metadata to natural language words (longest in this case) to indicate to our NLIDB how to process a superlative

Table 3. Classes and properties defined for modeling of superlatives

Name	OWL Component	Description
CSuperlativeToken	Class	Instantiates each superlative word, introduced by the user, as individual. This class is subclass of CToken class
SuperlativeColumn	Datatype property	Stores the column names to whom the superlative can be applied
SuperlativeFunction	Datatype property	Stores if the superlative refers to a maximum or minimum value

Each superlative is represented as an individual from the *CSuperlativeToken* class, subclass of the CToken class. The *SuperlativeColumn* and *SuperlativeFunction* data properties serve as metadata that indicates to the NLIDB how the superlative must be interpreted. SuperlativeColumn defines the columns to which the superlative can be applied and SuperlativeFunction defines whether the superlative refers to a maximum or minimum value. Figure 4 shows an example of how superlatives are modeled.

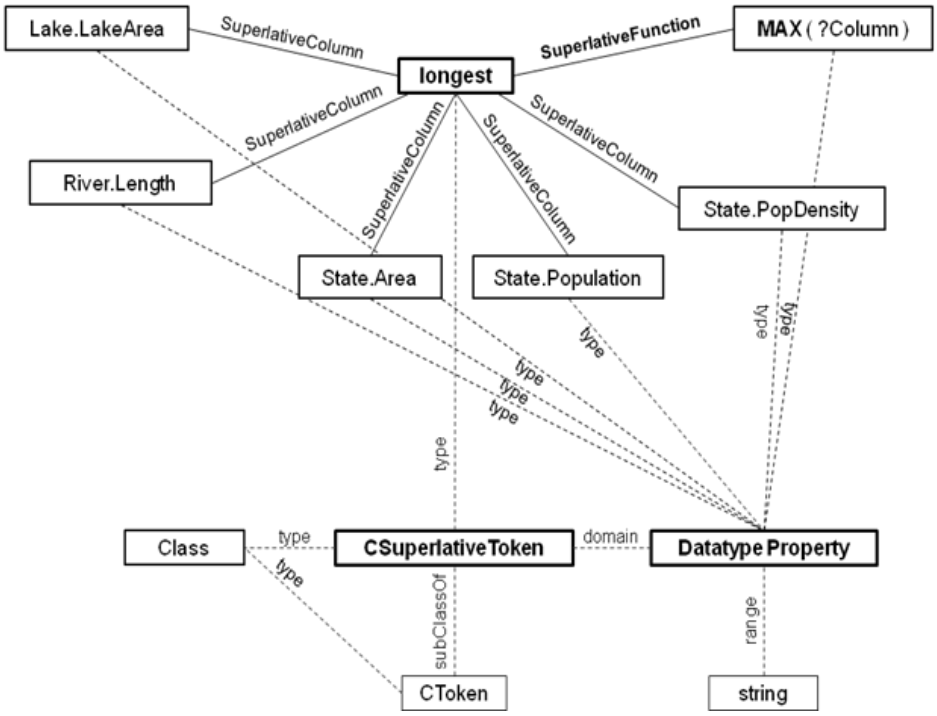

Fig 4. Semantic representation to model superlatives.

It is noteworthy that the knowledge described in sub-section 3.2 y 3.3 must be defined by the user by using the CM which is described in the following section. We also want to emphasize that, like superlatives, we designed semantic representations to add metadata to interpret natural language words to enable our NLIDB to interpret natural language queries that integrate *time intervals* and *constraints* when extracting information from the database. The descriptions of these semantic representations were not considered as part of this paper.

4 Customization Module

We implemented the CM to integrate the semantic representations described in section 3 with the aim of facilitating the user the configuration of our NLIDB. The NLIDB is configured in two steps: *a*) in the first step, the CM automatically generates the SDbS and *b*) in the second step, the CM uses the information modeled in the SDbS for the user can customize the NLIDB. Figure 5 shows the architecture of the CM.

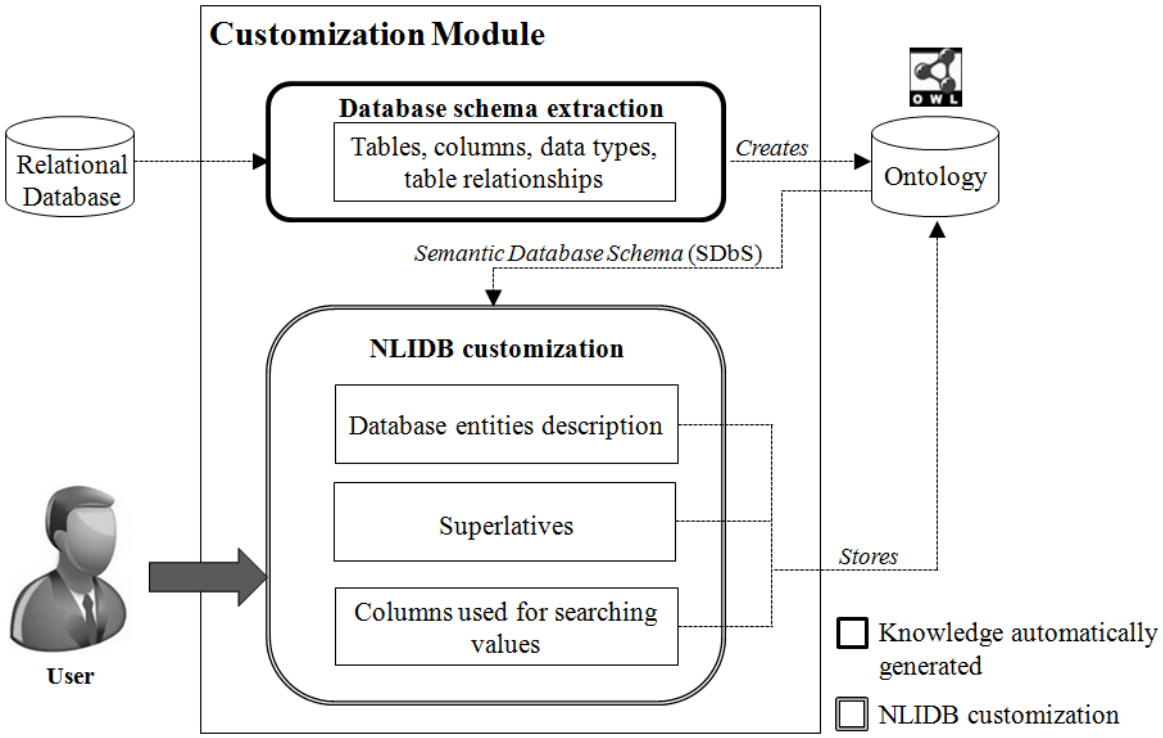

Fig. 5. Customization Module.

To generate the SDbS, the user must indicate to the CM the database in which the NLIDB is going to be configured. The CM analyzes the database schema in order to extract the names of tables, columns, data types and the table relationships. Subsequently, the semantic representations described in sub-section 3.1 are used to model the information extracted from the database schema. Finally, the information modeled in the SDbS is stored in an ontology, which is dynamically created by the CM. It is noteworthy that these steps are performed automatically by the CM, since the user only has to specify the database in which the NLIDB will be configured. Once the SDbS has been generated, the user can customize the NLIDB in three steps.

In the first step, the user must introduce into the CM a set of natural language words to describe to each of the entities of the SDbS (CTable and CColumn individuals) what information could be queried by other users. When the user introduces a natural language word, (s)he must map it to those entities of the SDbS which it describes. For example, if the user maps the *state* word to the *State* individual; the *population* word to both *State.Population* and *State.PopDensity* individuals and the *density* word to the *State.PopDensity* individual, the NLIDB is able to interpret that the query "*What state has the largest population density*?" is about the *State* and the *PopDensity* individuals. This relationship is modeled as described in sub-section 3.2.

In the second step, the user can configure superlative words, if required. For this, the user must introduce into the CM the superlative word to be configured and must specify: *a*) if the superlative refers to a maximum or minimum value and *b*) the columns to which the superlative can be applied. This information indicates the NLIDB how to process that superlative. Superlatives are modeled as described in sub-section 3.3.

In the third step, the user must indicate the CM which columns of the database could store information that could be used by users as a *value* in their queries. We consider a word or sequence of words stored in any column in the database as a value and is used by the user to formulate queries. For example, in the query "*How large is Alaska*", *Alaska* is a value stored in the State.Name column. In our approach, the identification of values is an important part in the interpretation of the query, since it

allows our NLIDB to solve some problems of ellipsis. Ellipsis is the absence of information in a query, for example, the previous query does not specify that Alaska is a state; however, this problem can be solved by identifying that Alaska is a value stored in the State table.

The knowledge defined by the user during the NLIDB customization is also stored in the ontology generated earlier. It is noteworthy that a different ontology is generated by each database in which the NLIDB is configured, thus, the user only has to indicate the NLIDB, the ontology with which s(he) wants to work. As it can be seen, the user neither needs to be a database expert nor a knowledge engineer, nor is (s)he required to have knowledge about ontologies for configuring our NLIDB, since all knowledge is managed by the CM.

5 Example usage of the semantic representations

In order to show how the semantic representations are used by our NLIDB in order to interpret natural language queries and generate the appropriate SQL query, consider the following knowledge, generated and stored in the ontology by the CM.

```
<rdf:Description rdf:about="http://www.semanticweb.org/ontologies/OEBD.OWL#name">
  <j.0:mappingObject rdf:resource="http://www.semanticweb.org/ontologies/OEBD.OWL#LAKE.Name"/>
  <j.0:mappingObject rdf:resource="http://www.semanticweb.org/ontologies/OEBD.OWL#ROAD.Name"/>
  <j.0:mappingObject rdf:resource="http://www.semanticweb.org/ontologies/OEBD.OWL#RIVER.Name"/>
  <j.0:mappingObject rdf:resource="http://www.semanticweb.org/ontologies/OEBD.OWL#STATEHILOPOINT.Name"/>
  <j.0:mappingObject rdf:resource="http://www.semanticweb.org/ontologies/OEBD.OWL#CITY.Name"/>
  <j.0:mappingObject rdf:resource="http://www.semanticweb.org/ontologies/OEBD.OWL#MOUNTAIN.Name"/>
  <j.0:mappingObject rdf:resource="http://www.semanticweb.org/ontologies/OEBD.OWL#STATE.Name"/>
  <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/OEBD.OWL#CToken"/>
  <j.0:mappingObject rdf:resource="http://www.semanticweb.org/ontologies/OEBD.OWL#CAPITAL.Name"/>
</rdf:Description>

<rdf:Description rdf:about="http://www.semanticweb.org/ontologies/OEBD.OWL#river">
  <j.0:mappingObject rdf:resource="http://www.semanticweb.org/ontologies/OEBD.OWL#RIVER.Name"/>
  <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/OEBD.OWL#CToken"/>
  <j.0:mappingObject rdf:resource="http://www.semanticweb.org/ontologies/OEBD.OWL#RIVER"/>
</rdf:Description>

<rdf:Description rdf:about="http://www.semanticweb.org/ontologies/OEBD.OWL#population">
  <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/OEBD.OWL#CToken"/>
  <j.0:mappingObject rdf:resource="http://www.semanticweb.org/ontologies/OEBD.OWL#CAPITAL.Population"/>
  <j.0:mappingObject rdf:resource="http://www.semanticweb.org/ontologies/OEBD.OWL#CITY.Population"/>
  <j.0:mappingObject rdf:resource="http://www.semanticweb.org/ontologies/OEBD.OWL#STATE.Population"/>
  <j.0:mappingObject rdf:resource="http://www.semanticweb.org/ontologies/OEBD.OWL#STATE.PopDensity"/>
</rdf:Description>
```

In the knowledge above, we present the mappings associated to the natural language words *name*, *river* and *population*. It can be seen that the Lake.Name, Road.Name, River.Name, StateHiLoPoint.Name, City.Name, Mountain.Name, State.Name, and Capital.Name columns are mapping to the natural language word *name*. The River.Name column and the River table are mapping to the natural language word *river*. The Capital.Population, City.Population, State.Population and State.PopDesnity columns are mappings to the natural language word *population*. In the following knowledge, we show the metadata defined to the *longest* superlative.

```
<rdf:Description rdf:about="http://www.semanticweb.org/ontologies/OEBD.OWL#longest">
  <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/OEBD.OWL#CSuperlativeToken"/>
  <j.0:SuperlativeToColumn rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://www.semanticweb.org/ontologies/OEBD.OWL#LAKE.LakeArea</j.0:SuperlativeToColumn>
  <j.0:SuperlativeToColumn rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://www.semanticweb.org/ontologies/OEBD.OWL#RIVER.Length</j.0:SuperlativeToColumn>
  <j.0:SuperlativeToColumn rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://www.semanticweb.org/ontologies/OEBD.OWL#STATE.Area</j.0:SuperlativeToColumn>
  <j.0:SuperlativeToColumn rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://www.semanticweb.org/ontologies/OEBD.OWL#STATE.Population</j.0:SuperlativeToColumn>
  <j.0:SuperlativeToColumn rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://www.semanticweb.org/ontologies/OEBD.OWL#STATE.PopDensity</j.0:SuperlativeToColumn>
  <j.0:SuperlativeFunction rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Max( ?Column ) </j.0:SuperlativeFunction>
</rdf:Description>
```

As it can be seen, the metadata defined for the longest superlative indicate that this superlative is associated to a MAX function and it can be applied to the Lake.LakeArea, River.Length, State.Area, State.PopDensity and State.Population columns. Table 4 summarizes the mappings described above.

Table 4. Some mappings defined by the user by using the CM

CToken individuals	Elements modeled in the SDbS
name	Lake.Name, Road.Name, River.Name, StateHiLoPoint.Name, City.Name, Mountain.Name, State.Name, Capital.Name
river	River.Name, River
population	Capital.Population, City.Population, State.Population, State.PopDensity
longest	MAX (Lake.LakeArea), MAX (River.Length), MAX (State.Area), MAX (State.Population), MAX (State.PopDensity)

Based on the mappings summarized in Table 4, the NLIDB is able to answer queries such as: "*What is the longest river in Mississippi*?".

The first step to answering the above query is to interpret the query. For this, the NLIDB identifies by querying the ontology the words that are mapping to some components of the SDbS and also identifies the superlative words, as shown in Figure 6. In this case, the NLIDB identifies that the *river* word is mapped to the *River* table and to the *River.Name* column and that the *longest* word is a superlative, so it extracts, from the ontology, the columns to which the superlative can be applied and if it refers to a maximum or minimum value.

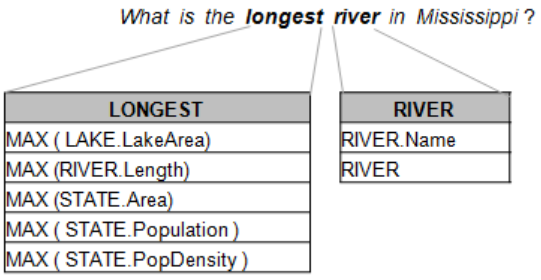

Fig. 6. Identification of components in the query.

After this identification process, the next step is to identify words that are semantically related to combine them and form a compound word. We defined that a word is semantically related to another if: *a*) both contain at least one mapping to a same column, *b*) both contain at least one mapping to the same table, or *c*) if one of both words has a mapping to a column, whose table is a mapping in the other word. In this case, the longest word has a mapping to the River.Length column whose table (River) is a mapping in the other word (river word). For this reason, both words are combined in a compound word and the remaining mappings of the longest word are filtered as shown in Figure 7. The aim of this process is to simplify the number of mappings of each word, allowing in this manner, to generate a SQL query more accurately.

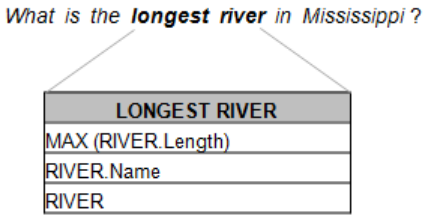

Fig. 7. Example of a compound word.

The next step is to identify values. For this purpose, the NLIDB extracts from the ontology the columns defined by the user to search values. In this case, the *Mississippi* word is identified as a value which is stored in the State.Name and River.Name columns, as shown in Figure 8.

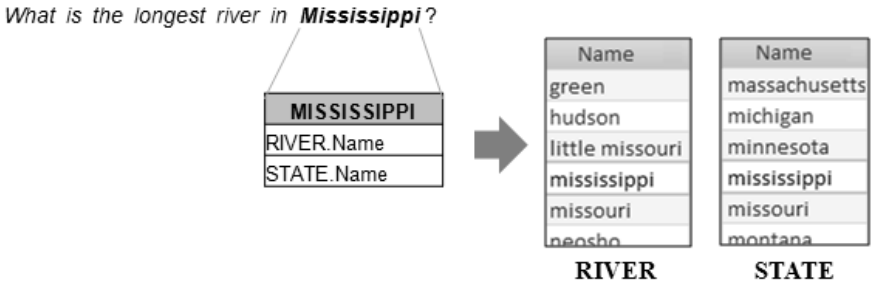

Fig. 8. Identification of values.

Since the Mississippi value is stored in two columns, the NLIDB shows the user a clarification dialog box for (s)he to specify which column must be related to this value (see Figure 9). In this case, the value stored in the State.Name must be used.

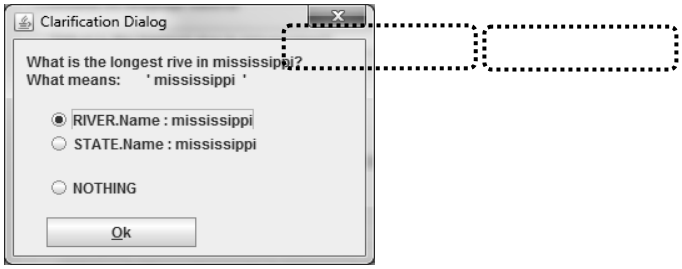

Fig. 9. Clarification dialog box.

In this way the query is filtered as shown in Figure 10, so the NLIDB is now able to generate the appropriate SQL query.

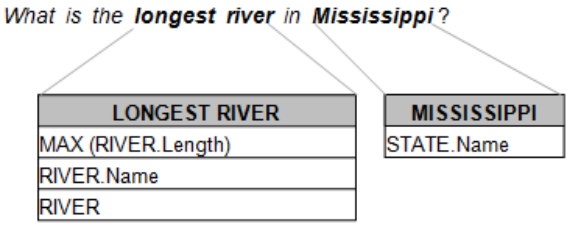

Fig. 10. Components identified in the user query.

To generate the SQL query, first the components that must be part of the *Select*, *From* and *Where* clauses are identified. The *Select* clause is composed by the columns that are not related to any value (River.Name). The *Where* clause is formed by the values identified and their columns where they are stored (State.Name = 'mississippi'). The *From* clause consists in the following components:

- Tables explicitly identified. In this case, the combined word *longest river* maps explicitly to the *River* table.
- The tables to which the columns of the *Select* clause belong.
- The tables to which the columns corresponding to the values identified belong. In this case, the *State* table is identified from the *mississippi* value which is stored in the *State.Name* column.

The metadata of superlatives, extracted from the ontology, are added to a sub-query in order to obtain the maximum or minimum value, as appropriate. Finally, the join of the tables of the *From* clause, is performed.

To perform the table joins, the NLIDB queries the ontology in order to identify whether it is necessary to add tables to the *From* clause to perform the join of each pair of tables from the *From* clause. Based on Figure 11, the NLIDB identifies that to perform the join of tables *River* and *State* it is necessary to add the *RiverRunsThrough* table to the *From* clause. In addition, the *primary* and *foreign* keys used to join both tables must be added to the *Where* clause. It is noteworthy that our NLIDB adds prefixes to the tables in the *From* clause (From River a, b State, RiverRunsThrough c) because this is important to generate more complex SQL queries. The SQL query generated by our NLIDB, based on the descriptions defined in Table 4, is shown below:

```
SELECT DISTINCT A.Name
FROM RIVER A, STATE B, RIVERRUNSTHROUGH C
WHERE B.Name='mississippi' AND A.RiverId=C.RiverId AND
                              C.StateId=B.StateId AND
      B.StateId=C.StateId AND
      A.Length=( SELECT Max( A.Length )
                   FROM RIVER A, STATE B, RIVERRUNSTHROUGH C
                  WHERE B.Name='mississippi' AND A.RiverId=C.RiverId AND
                        C.StateId=B.StateId AND B.StateId=C.StateId )
```

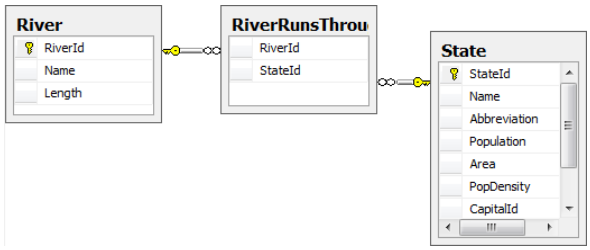

Fig. 21. Fragment of the Entity-Relationship diagram of Geobase

It is noteworthy that incorrect table joins could cause to display erroneous or additional information to the user. For this reason, we believe that table joins is key to generate the SQL query.

6 Experimental results

We are interested in evaluating if semantic representations described in this paper, allow our NLIDB to obtain a good performance. To evaluate the performance, we configured our NLIDB in the version of a relational database that we migrated from Geobase. Geobase was included in the commercial Prolog for PCs (Turbo Prolog 2.0, Borland International 1988). As metric, we considered the percentage of correctly-answered queries, which we measured using the *recall* formula expressed as follows:

$$\text{Recall} = \frac{\#\text{ number of correctly answered queries}}{\#\text{ total number of queries}} * 100$$

We compared the performance obtained by our NLIDB with the interfaces ELF, FREyA, and NLP-Reduce. ELF is a commercial interface to relational databases while FREyA and NLP-Reduce are interfaces to ontologies, as stated in section 2. We use these interfaces because we have a version of each for testing. In this way, we make sure to apply the same criteria to determine whether the answers of the interfaces were correct.

For the evaluation we used the GeoQuery250 corpus of the Raymond Mooney's group at the University of Texas, Austin (website: http://www.cs.utexas.edu/users/ml/nldata.html). This corpus is based on a series of queries carried out by a group of college students about a database of the geography of the U.S. The data to answer these queries were originally formalized in Prolog under the name of GeoBase. There is also a version of Geobase formalized in an OWL version (Web site: http://www.ifi.uzh.ch/ddis/research/talking/OWL-Test-Data.html). We used this version to evaluate the performance of FREyA and NLP-Reduce, since these interfaces are designed for querying ontologies. To test the performance of our NLIDB and ELF, we manually export the Geobase OWL version to a relational database schema.

We decided to use the GeoQuery250 corpus because most of its queries have ellipsis problems or require the use of superlatives in order to be correctly answered.

The evaluation consisted on introducing to each interface all queries of the GeoQuery250 corpus. The results obtained from the experiments are shown in Table 5.

Table 8 shows the number of correctly-answered queries, incorrectly-answered queries, and the percentage of recall obtained for each interface evaluated.

Table 5. Results obtained from the evaluation

NLIDB	Correctly Answered	Incorrectly Answered	Recall %
ELF	61	189	24.40
NLP-Reduce	49	201	19.60
FREyA	202	48	80.80
Our NLIDB	213	37	85.20

The percentages of recall indicate that our NLIDB obtained the best performance by responding correctly 85.20% of the 250 queries against 80.80% of FREyA, 24.40% of ELF, and 19.60% of NLP-Reduce.

Another aspect observed in the experimentation is the number of clarification dialogs used by the interfaces. Figure 12 and Figure 13 show the number of clarification dialog boxes used by FREyA and our NLIDB. ELF and NLP-Reduce are not designed to use clarification dialog boxes.

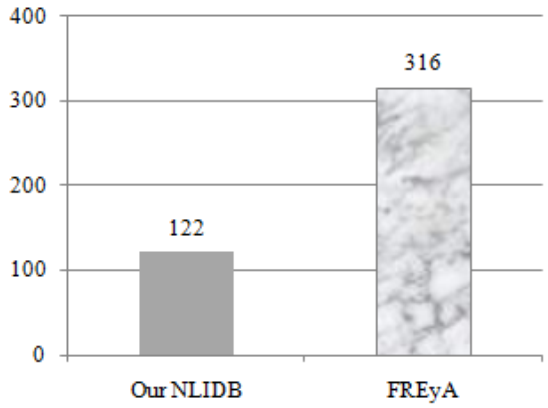

Fig. 12. Number of dialog boxes used by our NLIDB and FREyA for the 250 queries

.

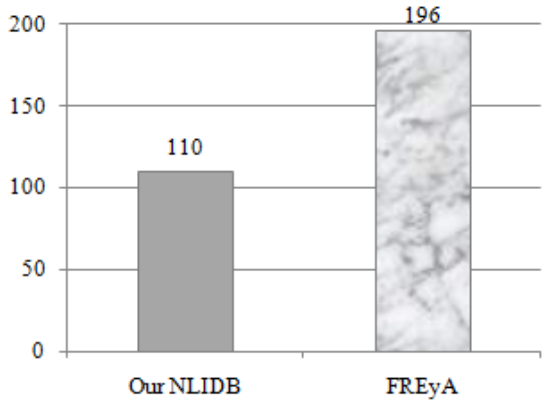

Fig. 13. Number of queries that used at least one dialog box.

Figure 12 and Figure 13 show that our NLIDB used 122 clarification dialogs in 110 queries and FREyA used 316 in 196 queries. We can conclude that our NLIDB has better interpreted the queries of the corpus used.

7 Conclusions

NLIDBs are software applications that facilitate inexperienced users to access the information stored in a database, since the interaction is through natural language. Despite the great advances in this area, NLIDBs have not achieved the performance expected by users.

In this paper we present a way of semantically modeling the elements that integrate the knowledge of a NLIDB, with the aim of increasing the number of correctly-answered queries.

To model the knowledge, we designed semantic representations in order to: *a*) model any relational database schema and its relationship with natural language, which allows interpreting natural language queries and generating the corresponding SQL queries and *b*) add metadata to natural language words to enable our NLIDB to interpret natural language queries that contain superlatives, which allows to increase the linguistic coverage of our NLIDB.

We implemented these semantic representations in a Customization Module to make the customization and management of the knowledge in our NLIDB easy for the user.

To evaluate its performance, we configured our NLIDB in the version of relational database that we migrated from Geobase, and we used the GeoQuery250 corpus. We compared the results obtained by our NLIDB with the commercial interface to databases ELF and the interfaces to ontologies FREyA and NLP-Reduce. The results indicate that our NLIDB obtained the best performance.

As final conclusions, we can say that the flexibility of ontologies to model any domain of knowledge allows us:

- To model the relationship between natural language and any database schema as a single unit of knowledge. This facilitated the interpretation of queries and helped generate the SQL queries, which translates into good performance.
- To model metadata that enable our NLIBD to interpret superlative words, increasing in this way its linguistic coverage. In the same way that it was done with superlatives, new features can be added in order to provide intelligent services to users.
- To implement a configuration tool that facilitates the portability of our NLIDB to different databases.

Based on the results, we can conclude that using semantic representations for knowledge modeling of a NLIDB is a good alternative to increase the number of correctly-answered queries.

As immediate work, we pretend that the mappings between natural language and database schema be automatically generated by the CM in order to reduce the user intervention to configure our NLIDB. In addition, we plan to add knowledge to the NLIDB to enable it to resolve the ambiguities and to semantically relate words more accurately, thus reducing the use of clarification dialogues to the user. As future work, in the long term, we will integrate new features to our NLIDB. We are working on integrating discourse features by designing semantic representations to dynamically model entities mentioned in users' queries. Discourse is an issue that has not been widely addressed. Discourse allows a NLIDB to answer complex queries interactively with the user. This facilitates the information extraction since humans do not get information through isolated queries, but through queries performed interactively. Discourse represents an advantage over query languages to databases and graphical interfaces.

8 Acknowledgements

We acknowledge the financial support from CONACYT Mexico and Tecnológico Nacional de México, since this project was possible thanks to their kind support.

References

1. Chandra, Y.: Natural Language Interfaces to Databases. PhD Thesis, University of North Texas (2006).
2. Androutsopoulos, I., Ritchie, G.D., Thanish, P.: Natural language interfaces to databases - An introduction. Journal of Natural Language Engineering, Vol. 1, pp. 29-81 (1995).
3. Nihalani, N., Motwani, M., Silaka, S.: Natural language interface to database using semantic matching. International Journal of Computer Applications, Vol. 31, pp. 29-34 (2011).
4. Zelle, J.M., Mooney, R.J.: Learning to parse database queries using inductive logic programming. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence, Vol. 2, pp. 1050-1055 (1996)
5. Tang, L.R., Mooney, R.J.: Using multiple clause constructors in inductive logic programming for semantic parsing. In Machine Learning: ECML 2001, Lecture Notes in Computer Science, Vol. 2167, pp. 466–477, Springer Berlin Heidelberg (2001).
6. Woods, W., Kaplan, R. Webber, B.: The Lunar Sciences Natural Language Information System. Final Report, Bolt Beranek and Newman Inc., Cambridge, Massachusetts. B. B. N. Report No 2378. (1972).
7. Codd, E.F.: Seven Steps to Rendezvous with the Casual User. In: IFIP Working Conference Data Base Management, pp. 179–200 (1974)
8. Hendrix, G.G., Sacerdoti, E.D., Sagalowicz, D., Slocum, J.: Developing a Natural Language Interface to Complex Data. ACM Transactions on Database Systems, Vol. 3, pp. 105–147 (1978).
9. Warren, D.H.: Efficient processing of interactive relational database queries expressed in logic. In Proceedings of the seventh international conference on Very Large Data Bases, Vol. 7, pp. 272–281 (1981).
10. Grosz, B.J., Appelt, D.E., Martin, P.A., Pereira, F.C.: TEAM: an experiment in the design of transportable natural-language interfaces. Artificial Intelligence, Vol. 32, pp. 173–243 (1987).
11. Bates, M.: Rapid Porting of the PARLANCE Natural Language Interface. In: Proceedings of the workshop on Speech and Natural Language. Association for Computational Linguistics, pp. 83–88 (1989).
12. Bobrow, R.J, Resnik, P., Weischedel, R.M.: Multiple underlying systems: translating user request into programs to produce answers. In: Proceedings of the 28th Annual meeting of ACL, Pittsburgh, pp. 227–234 (1990).
13. Pazos, R.A., González, J.J., Aguirre, M.A., Martínez, J.A., Fraire, H.J.: Natural Language Interfaces to Databases: An Analysis of the State of the Art. Recent Advances on Hybrid Intelligent Systems, Vol. 451, pp. 463–480 (2013).
14. Kate, R.J., Mooney, R.J.: Using string-kernels for learning semantic parsers. In: Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Sydney, Australia, pp. 913–920 (2006).
15. Ge, R., Mooney, R. J.: A statistical semantic parser that integrates syntax and semantics. In: Proceedings of the Ninth Conference on Computational Natural Language Learning, pp. 9–16 (2005)
16. Wong, Y.W., Mooney, R.J.: Learning for semantic parsing with statistical machine translation. In: Proceedings of Human Language Technology Conference / North American Chapter of the Association for Computational Linguistics Annual Meeting, pp. 439–446 (2006).
17. Androutsopoulus, I., Ritchie, G.D., Thanish, P.: MASQUE/SQL, An Efficient and Portables Language Query Interface for Relational Databases. In: Proceedings of the 6th international conference on Industrial and engineering applications of artificial intelligence and expert systems, pp. 327–330 (1993).
18. Zhang, G., Chu W.W., Meng F., Kong G.: Query Formulation from High-Level Concepts for Relational Databases. In: Proceedings of the 1999 User Interfaces to Data Intensive Systems, pp. 64–74 (1999)
19. ELF software (English Language Frontend).: Demo Gallery. http://www.elfsoft.com/help/accelf/Overview.htm. Accessed 23 April 2015.
20. Popescu, A.M., Armanasu, A., Etzioni, O., Ko, D., Yates, A.: Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In: Proceedings of the 20th international conference on Computational Linguistics, Article No. 141 (2004)
21. Damljanovic, D.: Natural Language Interfaces to Conceptual Models. PhD Thesis, The University of Sheffield (2011).
22. Buraga, S.C., Cojocaru, L., Nichifor, O.C.: Survey on Web Ontology Editing Tools. Transactions on Automatic Control and Computer Science, Romania, pp. 1–6 (2006).
23. Bernstein, A., Kaufmann, E., Kaiser, C.: Querying the Semantic Web with Ginseng: A Guided Input Natural Language Search Engine. In: 15th Workshop on Information Technologies and Systems, pp. 112–126 (2005)
24. Cimiano, P., Haase, P., Heizmann, J.: Porting natural language interfaces between domains: an experimental user study with the ORAKEL system. In: Proceedings of the 12th international conference on Intelligent User Interfaces, Honolulu, Hawaii, USA, pp. 180–189 (2007).
25. Kaufmann, E., Bernstein, A., Fischer, L.: NLP-Reduce: A "naïve" but Domain-independent Natural Language Interface for Querying Ontologies. In: 4th European Semantic Web Conference (ESWC 2007), Innsbruck, pp. 1–2 (2007).
26. Wang, C., Xiong, M., Zhou, Q., Yu, Y.: Panto: A portable natural language interface to ontologies. In: The Semantic Web: Research and Applications, Vol. 4519, pp. 473–487 (2007).
27. Damljanovic, D., Agatonovic, M., Cunningham, H.: Natural Language Interfaces to Ontologies: Combining Syntactic Analysis and Ontology-based Lookup through the User Interaction. In: The Semantic Web: Research and Applications, pp. 106–120 (2010).