Natural Language Interfaces to Databases: An Analysis of the State of the Art

Rodolfo A. Pazos R., Juan J. González B., Marco A. Aguirre L.,
José A. Martínez F., and Héctor J. Fraire H.

Instituto Tecnológico de Ciudad Madero, Mexico
{r_pazos_r,automatas2002}@yahoo.com.mx,
jjgonzalezbarbosa@hotmail.com, marco@marcoaguirre.com.mx,
jose.mtz@gmail.com

Abstract. People constantly make decisions based on information, most of which is stored in databases. Accessing this information requires the use of query languages to databases such as SQL. In order to avoid the difficulty of using these languages for users who are not computing experts, Natural Language Interfaces for Databases (NLIDB) have been developed, which permit to query databases through queries formulated in natural language. Although since the 60s many NLIDBs have been developed, their performance has not been satisfactory, there still remain very difficult problems that have not been solved by NLIDB technology, and there does not yet exist a standardized method of evaluation that permits to compare the performance of different NLIDBs. This chapter presents an analysis of NLIDBs, which includes their classification, techniques, advantages, disadvantages, and a proposal for a proper evaluation of them.

1 Introduction

Information is a fundamental factor in decision making. Currently the largest sources of information are stored in databases (DBs). Information stored in databases has been growing constantly, which has brought about the need for creating systems that permit users to obtain information easily, fast and reliably. In order for a user to obtain information from a database, he/she needs formulate a query in such a way that the computer interprets and generates the correct answer (usually in a database query language such as SQL). Unfortunately, only computing professionals can formulate queries in such way. Therefore, natural language interfaces to databases (NLIDBs) emerge as an alternative that permit users accessing information stored in databases through a query in natural language (NL).

The need for NLIDBs has grown popular nowadays as a consequence that many users request accessing information in different types of systems (computers, cell phones, PDAs) [8, 4].

Unfortunately, despite that fact that NLIDBs have been developed for more than 50 years, they have not attained a high enough performance (percentage of correctly answered queries close to 100%) to make them widely used in business applications. Most of the NLIDBs that have achieved good results (success rate around 95%) are domain dependent; i.e, those that can only be used to query a single database; however, domain-independent NLIDBs still have deficiencies in the translation process (form NL to a DB query language) and have attained a success rate in the interval 80-90%.

It is clear that the development of effective NLIDBs continues to be an open research topic, for the simple fact that no domain-independent NLIDB has achieved a success rate close to 100%.

This analysis of the state of the art in NLIDBs aims at showing the state of progress of this technology to date. To this end, the most relevant architectures, NL processing techniques and types of GUIs that have been used for implementing NLIDBs are described together with their advantages and disadvantages, as well as examples of NLIDBs that have used them. This chapter includes a description of the most relevant problems that have prevented NLIDBs achieving a level of performance good enough to make them widely used in business applications. Finally, since evaluation is an important aspect for assessing the progress of technologies, this chapter includes an overview of the databases, query corpuses and metrics that have been used for evaluating NLIDBs, as well as a proposal for standardizing their evaluation.

2 History

The first NL querying systems were developed in the 60s, and they were basically NLIs for expert systems tailored for specific domains, being some of the most famous BASEBALL [2] and LUNAR [3]. Most of those NLIDBs were developed keeping in mind a particular database, and consequently they could not be easily modified for querying different databases or they were difficult to port to different application domains. These systems had a database or a knowledge system created manually by domain experts. Some of the first NLIDBs were based on pattern matching techniques, described in Subsection 4.1.1.

Other interfaces that were developed in the late 70s were PLANES [36], REL [36] and LADDER [36]. These systems used a semantic grammar, described in Subsection 4.1.3. A boom of NLIDBs development occurred in the 80s, when numerous research prototypes were implemented, such as CHAT-80 [3], DATALOG [3], EUFID [3], and TELI [3].

The next generation of NLIDBs used an intermediate representation language, which expressed the meaning of user queries in terms of high level concepts, independently of the database structure [3]. TEAM [20] is an experimental and portable NLIDB developed in 1985, which consists of two main components: expression mapping from NL to formal representations and transformation of these expressions to a DB query language, which permits the separation of the linguistic process from the mapping process in the knowledge base.

Masque/pro translates English queries to expressions in a logic language called LQL, which are evaluated against Prolog databases. Later on, since most existing databases are relational, the authors decided to modify this system, from which

Masque/sql evolved, which is a portable NLIDB that supports SQL databases. The main adaptation consisted of translating LQL queries to SQL expressions [3].

Several NLIDBs were developed which were commercialized. Some of the most successful were INTELLECT, Q&A, English Wizard and English Query. However, when graphical user interfaces were introduced, these NLIDBs were gradually discontinued. Their performance was not very satisfactory, due to the existence of several problems: the language coverage of the system was not obvious for users; i.e., the system might be able to answer the query but it would not generate the correct result; the origin of its mistakes could not be discerned; i.e., the user could not guess if the problem was caused by a language coverage limitation or the conceptual scope of the database; and when the NL queries were ambiguous, usually the result was not what users expected.

3 Recent Work

The boom of NLIDBs development occurred in the 80s. Despite the development of many systems at that time, gradually their use decreased due to several problems that affect their performance. In this section some of the most relevant of recent works are presented; moreover, several of the systems described in this section were included because they use translation techniques different from the traditional ones.

PRECISE [30] introduces the notion of "semantically tractable sentences", which are those that can be translated to a unique semantic interpretation. Its accuracy rate for tractable sentences is impressively high (100%), however its recall rate is just 78-95%.

NLPQC [34] is a NLIDB that is used for querying the virtual library CINDI. This system is constituted by a run-time preprocessor. The preprocessor builds a conceptual knowledge base of the DB schema using WordNet, which is used at run time for analyzing the user query according to several predefined frameworks and for generating an SQL expression.

WYSIWYM [29] uses a semantic graph inspired by the one described in [37]. The interface helps users formulate queries using predefined frameworks, through a query interface, where the user completes the framework choosing from a list of search values for the columns involved in the query.

DaNaLIX [18] is a NLIDB to query XML databases. In this system the domain knowledge can be obtained from the interaction with the user. Query analysis focuses on keywords and its relationships with the XML elements, which are used for translating the query to XQuery.

C-PHRASE [22] has an authoring tool for editing the data dictionary of the NLIDB for a specific domain. Query analysis focuses mainly on recognizing noun phrases, and it uses a semantic grammar based on rules and patterns, which are used for translating the query to SQL.

In [11] a NLIDB that uses Sequence and Tree Kernels (STKs) and some variants is presented. In this system sets of data are constructed that consist of pairs, where each pair is constituted by a NL query and its translation into SQL. The NLIDB represents these pairs through syntactic trees, and uses kernel functions and support vector machines for carrying out the translation from natural language to SQL.

4 Classification of NLIDBs

In this section three types of classification of NLIDBs are presented; according to: their type of architecture, language processing technique, and type of graphical interface.

4.1 Classification by Architecture

4.1.1 Pattern Matching Systems

The first NLIDBs used this type of method, which employs a simple technique based on patterns or rules that are applied to the NL query [3]. In pattern-matching systems, their inner workings are closely related to the peculiarities of the database to be queried; consequently, these systems are limited to specific databases and the complexity of the patterns considered. In order to exemplify the simplicity of the method, let us consider a rule that could be used for the ATIS database [33]:

Pattern of NL query:
"Flight Code" of <A Flight Number>
SQL expression that corresponds to the preceding pattern:

SELECT flight_code.flight FROM flight WHERE flight_number = <A Flight Number>

The advantage of this technique is that it is neither necessary to implement a syntactic parsing nor interpretation modules for translating the query, which facilitates their implementation. The main disadvantage is that the simplicity and superficiality of the implementation (because of the limitation of the rules defined) frequently causes an incorrect translation of the NL query to a DB query language. Additionally, the coverage of the system is limited by the number of patterns it uses.

Some NLIDBs like the system described in [1] and gNarLI use this type of technique.

4.1.2 Syntax-Based Systems

Syntax-based systems use a syntactic parser for generating the parse tree corresponding to NL queries [3]. The parse tree is mapped to the DB query language based on predefined syntactic grammars.

The syntactic parser is a program that parses NL sentences. Parsers are implemented from a set of grammar rules and lexicons. Parsers are generally designed for a specific natural language due to the idiosyncrasies of different languages regarding grammar rules and lexicons.

The result of parsing a sentence is the syntactic representation of the sentence. The representation obtained consists of three types of information: part of speech of words, phrases, and relationships between words or phrases. The part of speech of a word in a sentence is its syntactic category (verb, noun, adjective, preposition, etc.). A phrase is a sequence of two or more words that are grammatically linked without subject and predicate.

Parsers also generate a representation for a sentence, called constituent tree. Relationships between words are represented in the constituent tree. Parsers use a set of rules based on the constituent tree for generating the syntactic relationships between words and phrases.

Once a user query is parsed into a constituent tree, syntax-based systems use a syntactic grammar for mapping user queries. An example of a possible grammar for a query to the GeoBase database looks as shown below.

- S → WQ VP
- WH → "what" | "which"
- WQ → WH "river" | WH "state"
- VP → V DP
- V → "passes through" | "borders"
- DP → "Illinois" | "Missouri" | "Indiana"

Considering the following query:

Which river passes through Illinois?

According to the syntactic grammar, the system could map this query to the following logical query to the database (where X denotes a variable):

? (river(x)∩flow_through(X,Ilinois)).

The advantage of using this technique is that it facilitates mapping from the syntactic tree to the DB query language, because the tree contains detailed information of the sentence of the user query. The disadvantages are related to problems concerning domain independence and DB query language. Problems such as semantic ellipsis may cause ambiguity when the query refers to a column that belongs to several DB tables. Other problems are related to the possibility of generating several syntactic trees, which could lead to several interpretations of the user query. There also exists the difficulty of mapping the sentence directly to a DB query language (such as SQL) due to the rules defined. The syntactic structures are deeply rooted to the database information; therefore, the systems that use this technique are difficult to port to another database.

An example of NLIDB that uses this translation technique is LUNAR.

4.1.3 Semantic Grammar Systems

The semantic grammar systems are similar to syntax-based systems: syntactic parsing is used for generating the constituent tree of the user query. This technique uses predefined grammars for mapping from the constituent tree to an SQL expression. An example of a possible semantic grammar is shown below [3].

S → River_question Flow_through
River_question → "which river"
Flow_through → "passes through" State
State → "Illinois"

Semantic grammars offer more-semantically-enriched information than syntactic grammars, which extends the scope of application of such systems. Semantic information about the domain knowledge is related to the semantic grammar. Semantic grammar categories are usually selected to enforce semantic constraints.

The advantage of this type of architecture is that the semantic representation does not have a complex syntactic tree. The structures permit assigning semantic information to the tree nodes, which reduces the elliptical problems of user queries.

The disadvantage of this method is that the semantic information about the knowledge domain is hard-wired into the semantic grammar (a new semantic grammar has to be written whenever the system is configured for a new domain), which makes difficult to port NLIDBs from one database to another. Moreover, the structures of the syntactic tree can hardly be useful for other databases. Some systems intend to obtain information from the domain knowledge by interacting with the user or extracting such information from a corpus of queries formulated by users.

Examples of NLIDBs that employ this translation technique are LADDER, PLANES, REL, PHILIQA1, EUFID, ASK, DATALOG, Nchiql, QBI, CoBASE, PRECISE, NLPQC, WYSIWYM, and the system presented in [11].

4.1.4 Intermediate-Representation Language Systems

Currently most NLIDBs use this technique for translating a NL user query to an intermediate logical query (in some intermediate meaning representation language). Intermediate logical queries express the meaning of user queries in terms of high-level-world concepts, which are independent of the database structure [3].

In this type of systems a sentence is mapped first to a logical query language, and subsequently this query is translated to a DB query language. This type of systems may include more than one meaning representation language.

In intermediate-representation language systems, a system can be divided into two parts: the first generates the logical query and the second generates the DB query from the logical query. The use of logical query languages permits adding reasoning capabilities to the system through the incorporation of reasoning modules between the semantic analyzer and the DB query generator. A recent variation of this architecture consists of using an internal structure (such as a semantic network) instead of an intermediate-representation language.

The advantage of using this technique is that the part that generates the logical queries is independent of the DB management system. This technique also permits the domain to be independent of the system modules and the addition of reasoning modules that enables the system to carry out reasoning based on information stored in the database. A disadvantage is that most of these kind of NLIDBs use deductive databases, which are much less used than relational databases; thus, the scope of application is limited.

Examples of systems that use this technique are RENDEZVOUS, CHAT-80, DIALOGIC, TEAM, DIAGRAM, JANUS, TELI, TAMIC, EDITE, and C-PHRASE.

4.2 Classification by Language Processing Techniques

4.2.1 Symbolic Approach

The symbolic method was prevalent in the first decades of NLIDBs development. Words are symbols that represent objects in the real world, and they are subject to well-defined grammar rules.

In this type of approach, language is coded in rules or other forms of representation. Language is analyzed at several levels for obtaining information, from which rules are obtained that are applied for achieving the intended linguistic functionality for the system. Similarly to the human language capabilities, a rule-based reasoning is carried out which is supported by symbolic processing [2].

The advantage of this approach resides on its simplicity, since it does not require a complex implementation. Its disadvantage is that its functionality and scope are limited to the rules defined for interpretation.

Many NLIDBs use this technique since they use rules for answering user queries. Some systems that use this technique are LUNAR, LADDER, PLANES, REL, PHILIQA1, EUFID, ASK, DATALOG, TAMIC, EDITE, NLPQC, and C-PHRASE.

4.2.2 Empirical Approach

Statistical analysis is the main tool for this type of approach, since it is based on text corpora analysis. The corpora are used as source of information for the natural language [2]. In this approach the syntactic parsing is carried out on a training corpus, in which the ellipsis and ambiguity problems can be resolved considering the context of the information obtained.

The advantage of using this approach is that, unlike the symbolic approach, this method automatically carries out the acquisition and learning of more complex knowledge. Its disadvantage is that its effectiveness depends on the training corpus, and it also needs some processing time that depends on the corpus size.

Examples of NLIDBs that use this approach are PRECISE, WYSIWYM, DaNaLIX and the system presented in [11].

4.2.3 Connectionist Approach

Artificial neural networks permit modeling language processing. Instead of symbols, this method is based on distributed representations that correspond to statistical regularities in natural language [2].

The advantage of this approach consists of permitting high-level representations of knowledge, which allows more information and possibly larger domain independence. The disadvantage is that it probably needs more processing capacity that the other techniques.

An example of the use of this method is presented in [11]. Additionally, systems such as Nchiql, QBI, CoBase, and WYSIWYM use semantic graphs for interpreting user queries. In Owda's NLIDB [25] knowledge representation trees are used.

4.3 Classification by Type of GUI

4.3.1 Command Line Based

This is a method that permits users to input NL queries through a simple text line. The first NLIDBs used this type of approach because of the existing software and hardware limitations.

Though inexperienced users easily learn to interact with graphic interfaces, a well-designed command line interface has several advantages over other interface types. Since the interface does not have complex components, its simplicity and uniformity make it easy to use. In general, the execution of a command can be faster than using a graphic interface.

A disadvantage of this type of approach is that it might be complicated to minimize user mistakes. Though this type of interfaces have uniformity in the interaction with the user, since they do not have other tools such as clarifying dialogues, the results obtained are not always satisfactory because there is no feedback from the NLIDB.

An example of NLIDB that uses this type of interface is NLPQC.

4.3.2 Menu-Based

In this type of systems users formulate queries choosing from menus of phrases and search values. The menus are displayed on the screen and are constantly updated. The user can browse the menus to find out the queries that the system can answer.

The advantage of this kind of systems is its ease of use and the guarantee that queries are correctly worded. A major disadvantage is that the coverage of the system is limited by the number of phrases included in its menus. Another disadvantage is the heavy use of computer resources due to the constant updating of the system menus. Additionally, query formulation may occasionally require searching through a lot of information, which may cause a lengthy search in the menus.

Some examples of menu-based NLIDBs are NLMenu System, TEAM, and WYSIWYM, as well as the systems presented in [15] and [19].

4.3.3 Conversation/Dialogue-Based

Conversation-based systems use conversation agents that permit users to interact with the NLIDB, in such a way that the users may refine their queries through dialogues until they get the desired result. There exists another type of dialogues used by NLIDBs, which are called clarifying dialogues. These dialogues, like conversation-based dialogues, are used after the query has been preprocessed in order to clarify the meaning of a query that has not been fully understood (usually asking for omitted words in the query wording).

The advantage of this type of interface is its reliability and friendliness, since it permits the user to refine or clarify the query for attaining the desired result. Interaction with the user makes possible to integrate learning modules that permit feedback to the system, which in turn leads to improved performance. With a good dialogue design, this type of interfaces guarantees obtaining the desired results with better accuracy.

However, it might be necessary to configure the system so it adequately responds and interprets the responses obtained from the user as a result of the interaction with the system. There might also occur superfluous interactions with the user. Another disadvantage is that the interaction with the system could be lengthy when dealing with complex queries or elliptical queries (those in which words crucial for their interpretation are omitted). Due to the processing and coverage of dialogue cases, the system might require a larger amount of computational resources.

Some examples of conversation/dialogue-based NLIDBs are SHRDLU, WISBER, and TAMIC, as well as the systems presented in [15] and [25]. Other systems such as IR-NLI support spoken dialog.

4.3.4 Multimode-Based

This kind of interfaces has sophisticated selection and navigation techniques for selecting objects and context-sensitive menus for manipulating them. This technique could be considered as a combination of menu-based interfaces and dialogue-based interfaces, which in some cases includes the use of checkboxes, lists, objects, etc., and even speech for interacting with the system.

Because of the combination of techniques, multi-mode interfaces share their advantages and disadvantages. Among their advantages, it can be mentioned that users can formulate queries at a higher level, since queries can be formulated at different levels of construction, additionally they permit clarifying details of the user query.

The main disadvantage if this type of interface is that it requires a larger use of computational resources because of the larger amount of constituent elements. Occasionally, user-system interaction could be tedious and its use might possibly be difficult when the user ignores the operation of the system.

Some examples of multimode-based NLIDBs are JANUS, QBI, CoBase, info-Vis, DaNaLiX, and C-PHRASE.

5 Problems in NLIDBs

For over 50 years a large number of NLIDBs prototypes and commercial systems have been developed; however, to date they have not attained satisfactory results; i.e, recall rate close to 100%.

Interfaces such as PRECISE [30] (considered one of the most successful), only answers queries that are considered "tractable", where tractable is defined as "easy to understand, questions where the words or phrases correspond to database elements or constraints on join paths." Unfortunately, in complex databases (like ATIS), many queries involve semantic ellipsis which makes queries difficult to understand; for example in the query corpus for ATIS, the percentage of elliptical queries is approximately 85%.

The pitiful situation of commercial NLIDBs gives an indication of the backward state of NLIDB technology; most of the commercial products have been discontinued due to their poor performance and the difficulty for porting and configuring them. For example, LanguageAccess was discontinued by IBM, English Query (developed by Microsoft) was included for the last time in SQL Server ver. 8.0 in 2000, and English Wizard (by Linguistic Technology Corporation) was discontinued several years ago. ELF [5] (reputed as being one of the best NLIDBs and still available for sell) does not provide anymore technical support and it has not been updated nor improved its operation for many years.

Though many ILNBDs problems still remain unsolved, most researchers that have worked in this area, have moved to other research topics and have not continued to improve their prototypes performance.

Some of the main problems in which most of the researchers agree, are those related to portability (domain independence) and translation from NL to a DB query language [22, 30].

5.1 Domain Independence

A major issue in domain-independent NLIDBs is that it is very difficult to attain a recall rate above 90%, and usually this requires a tedious configuration process for adapting an interface to a specific database. The NLIDBs that offer domain independence need an initial configuration process before the final users can introduce their queries.

Configuring a NLIDB consists of supplying the system with the words and concepts needed for the new domain, which are related to the information stored in the database. Usually DB administrators need high levels of expertise and a long time for configuring the system [22]. Unfortunately, NLIDBs will not be widely used in businesses until their configuration is so easy that any computer professional could be able to perform it.

As a result of the difficulty of developing domain-independent NLIDBs that achieve recall rates closer to 100%, in recent years many researchers have focused their efforts in developing NLIDBs for specific domains and for languages different from English, some of which are described in [1, 7, 9, 10, 12, 14, 16, 17, 19, 24, 26, 31, 32].

Currently some NLIDBs that are considered domain-independent, have not shown supporting experimental results (CoBase [37] and InBASE [6]), have been tested with a single domain (NLPQC, STEP [22], C-PHRASE, and [23]), and other NLIDBs have been tested with databases that are not complex (ENLIGHT [13], NaLIX [18], DaNaLIX, and the system presented in [11]).

Table 1. Types of problems that occurs in queries

Cases	Problems
1	**Use of words or phrases of different syntactic categories**
1.1	Use of nouns or nominal phrases for referring to tables or columns. Example: List ***number of seats*** on D9S.
1.2	Use of verbs or verbal phrases for referring to tables or columns. Example: What time does flight 102136 ***leave*** ATL to DFW?
1.3	Use of prepositions or prepositional phrases for referring to tables or columns. Example: Give me an economy class flight ***from*** DFW ***to*** BWI one-way.
1.4	Use of adjectives or adjectival phrases for referring to tables or columns. Example: How ***fast*** can the Concorde fly?
1.5	Use of temporal adverbs. Example: List fares for all flights leaving ***after*** twelve o'clock noon from BOS to BWI.
1.6	Use of conjunctions. Example: Flights exiting Fort Worth ***and*** entering Dallas.
2	**Semantic ellipsis**
2.1	Lacking information of tables or columns. Example: List ***fares*** for all flights leaving after twelve o'clock noon from BOS to BWI. Note: there exist two columns related to the word "fare": "one_way_cost" and "rnd_trip_cost", thus it is not clear which of these columns is being referred to.
2.2	Lacking information of tables or columns referred to by some value. Example: How much is ***Delta*** flight ***539***?
2.3	Lacking information of tables or columns about the information requested. Example: ***All flights*** from ATL to SFO on Delta first class.
3	**Covering of the capability of SQL**
3.1	Queries that involve several tables. Example: Give me an ***economy*** class ***flight*** from DFW to BWI ***one-way.*** Note: the columns referred to by “economy”, “flight” and “one-way” belong to different tables.
3.2	Queries that involve aggregate functions and the Group By clause. Example: Which flight from Philadelphia to Dallas has the ***cheapest fare***?
4	**Other type of problems**
4.1	Search values that involve two or more columns. Example: Give me the hire date of the employee ***Margaret Peacock***. Note: the value "Margaret Peacock" involves two columns: “FirstName” and “LastName.”

Table 1. *(continued)*

4.2	Search values constituted by two or more words. Example: Give me the postal code and city of the supplier "***Exotic Liquids***".
4.3	Incomplete search values. Example: What is the name of the store where is "***the busy".***
4.4	Inexistent tables or columns. Example: Get me a ***date*** on flight 294 leaving ATL to Washington. Note: the table or column "date" does not exist in the database.
4.5	Spacing, punctuation and formatting mistakes. Example: Flights between SFO and Dallas between noon and ***5:00 P.M.*** Note: the ATIS database uses the military format for time.
4.6	Imprecise search values. Example: Show me the Atlanta to Dallas flights in the ***morning***.

5.2 Translation Process

The translation process is one of the most important aspects of a NLIDB, since it deals with understanding the query and translating it to a DB query language. In Sections 4.1 and 4.2 several translation approaches were described.

Translation issues are caused by problems that occur in queries. From an analysis carried out on query corpora involving three databases (ATIS, Northwind and Pubs), four general types of problems (i.e., those that usually are found in queries for most databases) were identified and classified [29]:

1. The use of words or phrases of different syntactic categories (such as nouns, verbs, adjectives, and prepositions) for referring to tables or columns of the database.
2. Semantic ellipsis that occurs when words that are necessary for clearly understanding the query are omitted.
3. Coverage of the capabilities of SQL, such as involving several tables of the database and the use of aggregate functions.
4. Other type of problems related to human errors, such as nonexistent information in the database, words that indicate imprecise values, etc.

The classification obtained and some examples are shown in Table 1. It is important to point out that the classification carried out according to the problems found in the corpora mentioned can be applied to most databases.

5.3 Desirable Features of a NLIDB

Obviously the main characteristic that any NLIDB must have is that it should answer correctly all the user queries. Additionally, a survey of the literature on ILNBDs revealed that, regarding the issues faced by ILNBDs, other characteristics that are considered the most important are the ones mentioned next.

Ease of configuration. The interface should be easily and swiftly configured (requiring a minimal user intervention). It should preferably be auto-configurable.

Operability. The system should be easy to use (friendly interface). Preferably, users should be guided through the querying process.

Authoring. The interface should have a tool that permits carrying out modifications of the knowledge base or data dictionary.

Habitability. The capacity of an interface for meeting user expectations, without surpassing the system linguistic capacity (limited grammar and linguistic coverage).

Transparency. The capabilities and limitations of the system should be evident to its users.

Robustness. Capacity of answering all the user queries.

Efficiency. The system should answer quickly.

Accuracy. The system should answer correctly all the queries.

Intelligence. The system should be able to answer deductive and temporal queries. It should also be able to improve its performance through information feedback obtained from user-system interaction.

Multimodality. Since most NLIDBs only support user-system interaction through keyboard, which is little practical, it is expected that the interface uses different means for inputting queries (such as speech, menus, forms, graphic objects, etc.) that permit users to obtain the advantages from the use of several input media.

Independence. The interface should be independent in four aspects:

1. Domain independence. The interface should be able to be ported to any database, so that it can answer queries concerning the domain at hand.
2. DBMS independence. The interface should extract information from the database regardless of the database management system (Oracle, Sybase SQL Server, PostgrestSQL, Microsoft SQL Server, Access, DB2, and Informix, and MySQL).
3. Natural language independence. The interface should be multi-lingual.
4. Hardware and software independence. The interface should permit porting it to any computing platform.

Handling of linguistic phenomena. The interface should consider the linguistic problems that may affect the meaning of the syntactic elements involved in user queries. Some of the most important are: anaphora, ellipsis and ambiguity.

6 Evaluation of NLIDBs

Currently, evaluating a NLIDB is a disturbing task, because there are no generally accepted evaluation benchmarks that can be used for comparing the performance of different NLIDBs, unlike other more mature research fields. Some characteristics of the evaluations of most NLIDBs are presented next, regarding databases used, test corpora and evaluation metrics.

6.1 Databases Used for Evaluation

Despite the large number of NLIDBs developed to date, there are only a few databases that have been used for evaluation.

Many of the NLIDBs use three databases developed by L. Tang and R. Mooney (used in [35]): Geobase, Restbase and Jobdata, relative to geography, restaurants and jobs domains. It is important to point out that these databases are not relational (they are programmed in Prolog) and their structure is simple (they contain from 1 to 8 entity classes). Researchers that have used these databases to test their NLIDBs, had to convert manually these databases to relational databases.

The first NLIDBs were designed for this type of databases, called deductive databases, because they contain rules for deducting facts from logic predicates. NLIDBs for deductive database never became popular because the vast majority of the databases used in businesses are relational.

Some of the databases used for testing are Northwind (traders database) [21], Pubs (books database) [21], CINDI (database of the virtual library of the Concordia University) [34], and in some cases student databases, which do not have a complex structure, and for which there does not exist a real-life or a corpus of complex queries for testing.

Finally, ATIS, a database that handles information on airline flights, is one of the most complex relational databases used for testing. Due to the complexity of its structure (27 tables, 123 columns and a corpus with 85% of elliptical queries), only few NLIDBs have dared to test their performance using this database.

6.2 Query Corpora Used for Evaluation

Similarly to the fact that there is no generally accepted benchmark database, neither does exist a benchmark corpus of queries for testing NLIDBs. From the query corpora mentioned in the literature, only the ATIS, Geobase, Restbase and Jobdata have corpora of real-life queries. Unfortunately, despite the large number of queries in ATIS (the largest corpus) many queries are repeated (i.e, they have the same structure but different search values), and the queries do not encompass all the types of problems that occur in queries.

The corpora for Geobase, Restbase and Jobdata are constituted by deductive queries because the databases are deductive. The queries of these corpora may have a high degree of logical complexity, but their databases do not have a complex structure (as compared with ATIS, for example).

The ATIS corpus, though it has many repeated queries, it has a large variety of queries that involve the most common interpretation problems found in real life. Moreover, the complexity of the database adds further complexity to some queries.

6.3 Evaluation Metrics

There exist many metrics that could be used for evaluating ILNBDs; however, the most used ones by researchers are the following:

$$\text{accuracy} = \frac{\text{total number of correct queries}}{\text{total number of parsed queries}} \times 100 \qquad (1)$$

$$\text{recall} = \frac{\text{total number of correct queries}}{\text{total number of queries}} \times 100 \qquad (2)$$

As shown by expression (1), accuracy is the percentage of correctly answered queries with respect to the number of translated queries. As shown by (2), recall is the percentage of correctly answered queries with respect to the number of queries input to the interface.

Unfortunately, there is no uniformity on what "correct query" means. In some NLIDBs evaluations, queries whose results include additional information to that requested in the query, are counted as "correct queries"; while in other NLIDBs evaluations "correct queries" include only those queries whose results include just the requested information. This lack of uniformity makes impossible to compare the performance of different NLIDBs.

6.4 Proposed Evaluation

As it has been mentioned, evaluating a NLIDB (by researchers and potential users of commercial interfaces) might be a disturbing task, since there is no established benchmark that permits standardizing the tests to carry out for a NLIDB. This situation makes impossible to compare the performance of different NLIDBs and to assess the progress of NLIDB technology.

As a result of the survey carried out on different state-of-the-art NLIDBs, we propose some testing means for guaranteeing a meaningful and useful evaluation of NLIDBs, which includes databases, query corpora, and evaluation metrics.

We propose to evaluate interfaces using corpora that include queries that involve the problems listed in Table 1, in order to find out how many of such problems are dealt with correctly by an interface.

Concerning databases, the use of at least three relational databases is proposed, whose structure has a complexity degree from medium to high (regarding number of tables and relations among tables). The ATIS database is a good candidate for the benchmark, since it has 27 tables and a complex relations structure. However, additional databases are needed, since the ATIS database is not adequate for formulating queries that involve certain problems; for example, queries that involve the Group By clause. By the way, the group of benchmark databases should permit formulating queries that involve all the problems listed in Table 1.

Additionally, we propose that only the recall rate should be used to assess performance, and to this end, each query in the benchmark corpora should be accompanied with the equivalent expression in SQL, in order to verify the correct translation of the query.

We claim that **the performance of a NLIDB can not be fully described by a single recall number, but by a group of numbers**, one for each aspect to evaluate. For example, a test of ELF carried out by us on a set of five complex queries that involve the Group By clause yielded 0% of correctly answered queries; however, when tested with another query corpus ELF's recall rate is around 75% [28]. Therefore, to fully assess the performance of a NLIDB, a recall percentage should be measured for queries that involve words or phrases of different syntactic categories (nouns, verbs, adjectives, prepositions), a recall percentage should be measured for elliptical queries, another recall percentage for queries that involve Group By, and so on.

7 Concluding Remarks

This analysis of the state of the art in NLIDBs aims at showing the state of progress in this field to date. Our conclusion is that, despite the large number of ILNBDs that have been implemented for more than 50 years, these systems have not attained a high enough success rate (recall close to 100%) to make them widely used in business applications. We claim that for NLIDBs to be useful for such applications, they have to reach a success rate greater than or equal to the success rate of a good computing professional. Therefore, the good news for NLIDB researchers is that there is plenty of work ahead.

The main issues that hinder the advancement of NLIDB technology are: achieving high recall rates with domain-independent systems, developing easily configurable NLIDBs, and lack of benchmarks for evaluating NLIDB performance.

From the experience gained during the development of two NLIDB prototypes during the past 11 years [27, 28], we have arrived at the following conclusion: the process of correctly translating from a NL query to SQL is an extremely difficult problem, since it involves many issues (most of which are summarized in Table 1). We think that new efforts should focus on thoroughly studying and fully solving each of those problems.

Finally, like other more mature research fields, there is the need for an established benchmark that permits obtaining a meaningful evaluation of the performance of NLIDBs. To this end, we propose: the development of a benchmark consisting of two or three databases (ATIS could be a good candidate), the development of query corpora for the benchmark databases consisting of queries that involve the problems presented in Table 1, and the use an overall recall rate together with a recall rate for each of the different problems that occur in queries (those presented in Table 1).

References

1. Ahmad, R., Abid, M., Ali, R.: Efficient Transformation of Natural Language Query to SQL. In: Proc. 2nd Conference on Language and Technology, pp. 53–60 (2009)
2. Akerkar, R., Joshi, M.: Natural Language Interface Using Shallow Parsing. International Journal of Computer Science & Applications 5, 70–90 (2008)
3. Androutsopoulos, I., Ritchie, G., Thanisch, P.: Natural Language Interfaces to Databases - An Introduction. Natural Language Engineering 1, 29–81 (1995)
4. BBC, Bill Gates: Mouse is out, Touch Screen and Natural Language Interface are in (2008),

http://news.bbc.co.uk/player/nol/newsid_7170000/newsid_7174300/7174330.stm?bw=bb&mp=wm&asb=1&news=1&bbcws=1 (accessed June 10, 2012)

5. Bhootra, R.: Natural Language Interfaces: Comparing English Language Front End and English Query. Dissertation, Virginia Commonwealth University (2004)
6. Boldasov, M., Sokolova, E., Malkovsky, M.: User Query Understanding by the InBASE System as a Source for a Multilingual NL Generation Module. In: Proc. 5th International Conference on Text, Speech and Dialogue, pp. 33–40 (2002)
7. Boonjing, V., Hsu, C.: A New Feasible Natural Language Query Method. International Journal on AI Tools 15, 323–330 (2006)
8. Cimiano, P., Haase, P., Heizmann, J.: Porting Natural Language Interfaces between Domains: an Experimental User Study with the ORAKEL System. In: Proc. 12th International Conference on Intelligent User Interfaces, pp. 180–190 (2007), doi:10.1145/1216295.1216330
9. Djahantigui, F., Norouzifard, M., Davarpanah, S., Shenassa, M.: Using Natural Language Processing in Order to Create SQL Queries. In: Proc. International Conference on Computer and Communication Engineering 2008, vol. 13, pp. 600–604 (2008)
10. El-Mouadib, F., Zubi, Z., Almagrous, A., El-Feghi, I.: Interactive natural language interface. Journal WSEAS Transactions on Computers 8, 661–680 (2009)
11. Giordani, A., Moschitti, A.: Semantic Mapping Between Natural Language Questions and SQL Queries via Syntactic Pairing. International Conference on Applications of Natural Language to Information Systems, pp. 207–222 (2010), doi:10.1007/978-3-642-12550-8_17
12. Jain, H., Bathia, P.: Hindi Language Interface to Databases. Journal of Global Research in Computer Science 2(4) (2011)
13. Joshi, M.: Intelligent Natural Language Interface. Dissertation, Maharshtra University (2006)
14. Karande, N., Patil, G.: Natural Language Database Interface for Selection of Data Using Grammar and Parsing. In: Proc. World Academy of Science Engineering Technology, vol. 59, pp. 484–487 (2009)
15. Kim, H.: A Dialogue-Based NLIDB System in a Schedule Management Domain. In: Proc. 33rd conference on Current Trends in Theory and Practice of Computer Science, pp. 869–877 (2007), doi:10.1007/978-3-540-69507-3_75
16. Kovacs, L.: SQL Generation for Natural Language Interface. Journal of Computer Science and Control Systems 2(18), 19–22 (2009)
17. Le, T.: A Frame-based Approach to Text Generation. In: The 21st Pacific Asia Conference on Language, Information and Computation, vol. 21, pp. 192–201 (2007)
18. Li, Y., Chaudhuri, I., Yang, H., et al.: DaNaLIX: a Domain-adaptive Natural Language Interface for Querying XML. In: Proc. 2007 ACM SIGMOD International Conference on Management of Data, pp. 1165–1168 (2007), doi:10.1145/1247480.1247643
19. Majeed, F., Shoaib, M., Ashraf, F.: An Approach to the Optimization of Menu-based Natural Language Interfaces to Databases. IJCSI International Journal of Computer Science Issues 8(4) (2011)
20. Martin, P., Appelt, D., Grozs, B., Pereira, F.: TEAM: An Experimental Transportable Natural-Language Interface. IEEE Database Eng. Bull 8(3), 10–22 (1985)
21. Microsoft Northwind and Pubs Sample Databases for SQL Server 2000 (2004), http://www.microsoft.com/en-us/download/details.aspx?id=23654 (accessed June 10, 2012)

22. Minock, M., Olofsson, P., Näslund, A.: Towards Building Robust Natural Language Interfaces to Databases. In: Kapetanios, E., Sugumaran, V., Spiliopoulou, M. (eds.) NLDB 2008. LNCS, vol. 5039, pp. 187–198. Springer, Heidelberg (2008)
23. Nguyen, A.K., Le, H.T.: Natural Language Interface Construction Using Semantic Grammars. In: Ho, T.-B., Zhou, Z.-H. (eds.) PRICAI 2008. LNCS (LNAI), vol. 5351, pp. 728–739. Springer, Heidelberg (2008)
24. Nihalani, N., Motwani, M., Silaka, S.: Natural Language Interface to Database using semantic matching. International Journal of Computer Applications 31, 29–34 (2011), doi:10.5120/3942-5552
25. Owda, M., Bandar, Z., Crockett, K.: Conversation-Based Natural Language Interface to Relational Databases. In: Proc. 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, pp. 363–367 (2007)
26. Pakray, P.: Keyword Based Multilingual Restricted Domain Question Answering. Dissertation. Jadavpur University (2007)
27. Rangel, R.A.P., Joaquín Pérez, O., Juan Javier González, B., Gelbukh, A., Sidorov, G., Rodríguez M., M.J.: A Domain Independent Natural Language Interface to Databases Capable of Processing Complex Queries. In: Gelbukh, A., de Albornoz, Á., Terashima-Marín, H. (eds.) MICAI 2005. LNCS (LNAI), vol. 3789, pp. 833–842. Springer, Heidelberg (2005)
28. Pazos R., R.A., Rojas P., J.C., Santaolaya S., R., Martínez F., J.A., Gonzalez B., J.J.: Dialogue Manager for a NLIDB for Solving the Semantic Ellipsis Problem in Query Formulation. In: Setchi, R., Jordanov, I., Howlett, R.J., Jain, L.C. (eds.) KES 2010, Part II. LNCS, vol. 6277, pp. 203–213. Springer, Heidelberg (2010)
29. Pazos R., R.A., González B., J.J., Aguirre L., M.A.: Semantic Model for Improving the Performance of Natural Language Interfaces to Databases. In: Batyrshin, I., Sidorov, G. (eds.) MICAI 2011, Part I. LNCS, vol. 7094, pp. 277–290. Springer, Heidelberg (2011)
30. Popescu, A., Armanasu, A., Etzioni, O., et al.: Modern Natural Language Interfaces to Databases: Composing Statistical Parsing with Semantic Tractability. In: Proc. 20th International Conference on Computational Linguistics, vol. 141 (2004), doi:10.3115/1220355.1220376
31. Rao, G.: Natural Language Query Processing Using Semantic Grammar. International Journal on Computer Science and Engineering 2(2), 219–223 (2010)
32. Sarhan, A.: A Proposed Architecture for Dynamically built NLIDB Systems. International Journal of Knowledge Based Intelligent Engineering Systems 13(2), 59–70 (2009)
33. Sri International, Air Travel Information Service (1990), `http://www.ai.sri.com/natural-language/projects/arpa-sls/atis.html` (accessed June 10, 2012)
34. Stratica, N.: Using Semantic Templates for a Natural Language Interface to the CINDI Virtual Library. In: Data & Knowledge Engineering, vol. 55, pp. 4–19. Elsevier Science Publishers, The Netherlands (2005), doi:10.1016/j.datak.2004.12.002
35. Tang, L.R., Mooney, R.J.: Using Multiple Clause Constructors in Inductive Logic Programming for Semantic Parsing. In: Flach, P.A., De Raedt, L. (eds.) ECML 2001. LNCS (LNAI), vol. 2167, pp. 466–477. Springer, Heidelberg (2001)
36. Templeton, M., Burget, J.: Problems in Natural-Language Interface to DBMS with Examples from EUFID. In: ANLC 1983 Proc. First Conference on Applied Natural Language Processing, pp. 3–16 (1983), doi:10.3115/974194.974197
37. Zhang, G., Chu, W., Meng, F., et al.: Query Formulation from High-Level Concepts for Relational Databases. In: Proc. 1999 User Interfaces to Data Intensive Systems, pp. 64–75 (1999)