Features and Pitfalls that Users Should Seek in Natural Language Interfaces to Databases

Rodolfo A. Pazos Rangel, Marco A. Aguirre, Juan J. González and Juan Martín Carpio

Abstract Natural Language Interfaces to Databases (NLIDBs) are tools that can be useful in making decisions, allowing different types of users to get information they need using natural language communication. Despite their important features and that for more than 50 years NLIDBs have been developed, their acceptance by end users is very low due to extremely complex problems inherent to natural language, their customization and internal operation, which has produced poor performance regarding queries correctly translated. This chapter presents a study on the main desirable features that NLIDBs should have as well as their pitfalls, describing some study cases that occur in some interfaces to illustrate the flaws of their approach.

1 Introduction

Increasingly, a vast number of people in different areas use databases (DBs) for storing important information. Obtaining unusual or complex information from a database demands a broad knowledge on computing from users, such as a language for querying databases, and to be acquainted with the schema of the database to be queried.

R. A. Pazos Rangel (✉) · M. A. Aguirre · J. J. González
Instituto Tecnológico de Ciudad Madero, Ciudad Madero, Mexico
e-mail: r_pazos_r@yahoo.com.mx

M. A. Aguirre
e-mail: marco.itcm@gmail.com

J. J. González
e-mail: jjgonzalezbarbosa@hotmail.com

J. M. Carpio
Instituto Tecnológico de León, León, Mexico
e-mail: jmcarpio61@hotmail.com

Currently, a growing need for using systems that allows common users (different from computer professionals) to obtain important information from databases is growing. This capability can be extremely important for making business decisions.

Natural language interfaces to databases (NLIDBs) emerged as an attractive solution that serves as a bridge between common users and the information they need for decision making.

Since the early 1960s, researchers have undertaken the development of NLIDBs capable of satisfying the needs of final users.

However, the complexity of the problems involved in the development of effective interfaces has prevented their widespread acceptance.

A comprehensive review of the state of the art and some of the most important problems in NLIDBs are described in [1].

This study of the state of the art on NLIDBs is intended mainly for DB administrators that are considering installing a NLIDB for querying a database. This chapter defines the desirable features that DB administrators should look for in a system of this type. Finally, since there exist different types of problems that prevent NLIDBs to be attractive to users (regarding portability, functionality and performance), this chapter describes the pitfalls that some interfaces have.

2 Desirable Features in a NLIDB

NLIDBs were created for retrieving information from DBs by means of NL communication. What is expected to obtain from this interaction is the correct answer to the query input by a user. However, besides this characteristic, there exist others that are considered important for a NLIDB for being considered attractive enough to meet the expectations of end users, mainly those from businesses. Next we describe desirable characteristics that a NLIDB should have.

2.1 Ease of Customization

One of the main characteristics that a NLIDB must have is easiness of customization: the interface should be easily and rapidly customized, requiring a minimal intervention by the DB administrator.

There exist interfaces such as C-Phrase that have an authoring tool, which facilitates the customization task.

2.2 Operability

Like all Graphical User Interfaces (GUIs), NLIDBs are expected to have a friendly user interface that enables the system to be easily operated.

Earlier NLIDBs had a command-line-based user interface, where users formulated NL queries by typing them in a simple text line; this was so because of the hardware and software limitations at that time. Simplicity and fastness were the main advantages of the usage of those NLIDBs; however, the very same simplicity limited their operation because of the lack of tools for user-interface interaction. An example of this type of interfaces is NLPQC.

As technology improved and the boom of graphical interfaces grew larger, many different types of GUIs were available, such as: menu-based, dialogue/ discourse-based and multimodal.

Preferably, users should be guided by NLIDBs along query formulation.

2.3 Authoring

Customizing a NLIDB for a new domain is a complex task that could be time consuming. For facilitating the customization process, it is necessary that interfaces have a tool that permits modifying the knowledge base or the data dictionary.

An authoring tool should permit associating elements (DB tables, columns, relations between tables, etc.) of the DB schema to the linguistic elements (words or phrases) that can occur in NL queries, as well as including other elements that the interface might need, without compromising friendliness.

One of the few important NLIDBs that have an authoring tool is C-Phrase.

2.4 Habitability

The habitability is the ability of a NLIDB of knowing user expectations, without surpassing the linguistic capabilities of the system (limited grammar and linguistic coverage).

Obviously users expect the NLIDB to answer correctly all their queries; however, it can not be expected from the interface to be able of interpreting the queries as a human formulates them (including some jargon, words whose meaning is not in the data dictionary, etc.). For example, EasyAsk keeps terminology for business applications, but this would not be adequate if the interface were ported to an application of a different domain.

Some interfaces intend to increase their linguistic capacity by including domain-independent dictionaries, which are complemented with all the semantic information that is expected to be needed by users when querying a database for some specific domain.

2.5 Transparency

This characteristic establishes that the capabilities and limitations of a NLIDB must be evident to users. Though this information should be clear, usually the interfaces only describe their operation without making clear their capabilities and limitations. When something unexpected occurs (such as an erroneous translation or misinterpreted query, something that can not be customized, etc.), users might be disappointed because they do not know what has happened.

2.6 Robustness

A NLIDB must be capable of answering all the queries formulated by users. The system must cope with the problems that may arise when processing a query, as well as dealing with queries that involve aggregate functions, temporal queries, deductive queries, nested queries, and queries that involve negation.

2.7 Efficiency

A reason why a user would like to use a NLIDB is because he/she expects the interface to answer quickly. In the experiments reported on NLIDBs, the response time is not evaluated. Usually, the databases used for testing are small, and therefore response times are small. However, some NLIDBs (for example, ELF) have data dictionaries whose volume of information is proportional to the size of the database to be queried; therefore, the time for translating a query (from NL to SQL) increases with the size of the database, which would prevent using this type of NLIDBs for querying very large databases.

2.8 Accuracy

NLIDB users expect all the queries they formulate are correctly answered. This is considered the most important characteristic; however, many factors influence the results that can be obtained from an interface. These factors are related to natural language issues, interface customization, and the inner workings of the interface.

The average recalls achieved by state-of-the-art NLIDBs vary from 60 to 90 %, which are unsatisfactory for considering using them in businesses for decision making.

2.9 Intelligence

NLIDBs must be capable of answering temporal and deductive queries. Temporal queries are issued to a database that permits storing the history of values that a piece of data may adopt over time; while deductive queries are those that are based on deductions by using inference. Additionally, NLIDBs should be able to improve their performance (percentage of correctly answered queries) by feedback received from user-interface interaction.

Though intelligence is a desirable feature in an interface, the customization of this type of systems requires deep knowledge of their inner workings, which might make difficult their customization.

Interfaces such as DaNaLIX permit feedbacking the system by user-interface interaction.

2.10 Multimodality

Normally, most NLIDBs permit formulating queries through a keyboard; however, this might not be the most adequate in some situations. Current technology permits users to interact using mobile devices by means of speech, menus, graphical objects, touch screens, etc., which is expected to be adopted by NLIDBs for increasing their functionality.

There exist applications that use these types of interactions that operate as NLIDBs; however, they are designed for operating with just one domain, and consequently they have very good performance. Unfortunately, porting them to another domain would imply modifying their inner workings so they could answer correctly for the new domain.

Interfaces such as EasyAsk include voice recognition, and others like EDITE [2] use graphical objects for facilitating user-interface interaction.

2.11 Independence

There exist four types of independence in NLIDBs: domain, database management system, natural language, and hardware and software independence.

2.11.1 Domain Independence

Domain independence is one of the most important characteristics, since a NLIDB is expected to be portable to any database, so that it may answer queries related to the domain of interest.

An interface must have an architecture, and together with its data dictionary should permit customizing the interface for a new or different domain without requiring a lengthy intervention by the DB administrator.

An example of the information that must contain a data dictionary model can be seen in [3].

As previously mentioned, commercial interfaces such as ELF and EasyAsk have tools for semiautomatically customizing them for a specific domain, which usually is not good enough for correctly answering all the queries formulated by users. Additionally, EasyAsk provides utilities for customizing the interface to any domain by carrying out software design tasks, such as defining user requirements, design and implementation of the application and support, which can make more expensive using the interface.

2.11.2 DBMS Independence

A NLIDB must be able to retrieve information from a database independently of the database management system (DBMS). Some examples of DBMSs used are Oracle, Sybase SQL Server, PostgrestSQL, Microsoft SQL Server, Access, DB2, Informix, and MySQL.

To get an idea on the DBMSs that can be queried by NLIDBs, let us consider ELF which supports only Microsoft Access versions 2007 and 2010. Interfaces like DaNaLIX interact with XML databases; however, this type of databases does not require a typical DBMS, they simply use a framework that offers functionalities similar to those of a DBMS: query processing, rule insertion, etc.

DBMS independence is related to domain independence since it is useful for adapting the interface to any database regardless of its format (relational, ontology oriented, object oriented, etc.). Unfortunately, developing a NLIDB that includes this feature requires many implementation details.

2.11.3 Natural Language Independence

Most of the NLIDBs implemented support only English for formulating queries. Some, such as the ones described in [4–6], support a few other languages.

Some efforts have been devoted to developing NLIDBs capable of supporting multiple languages. To this end, it has been attempted to separate the syntactic parsing from the semantic analysis; however, the translation effectiveness varies widely depending on the syntactic and semantic complexity of every language, and only languages with similar syntax have been supported by multilingual interfaces. Some of the interfaces that have attempted to provide NL independence are those described in [7–11].

2.11.4 Hardware and Software Independence

A NLIDB must be able to be ported to any type of computer, and nowadays, to any wireless device (smart phone, tablet, notebook, PDA, etc.) regardless of the hardware and software that it uses.

The architecture of modern computers permits using many applications; however, more than hardware options, what might affect the use of a NLIDB in different devices is the software that supports it. This difficulty arises from the diversity of operating systems in the market together with the framework needed for supporting the interface.

2.12 Handling of Linguistic Phenomena

A NLIDB must consider the linguistic problems that may affect the meaning of a query.

Some linguistic problems are extremely complex, which can be observed when people communicate with each other, they might not follow syntactic or semantic rules, and their expressions usually involve syntactic and semantic ellipsis. Some of the most important linguistic problems are: anaphora, ellipsis and ambiguity. There exist several works that deal with these problems from the computational linguistic area; however, these problems have been overlooked in most NLIDBs.

3 Pitfalls in NLIDBs

Users expect NLIDBs to be intelligent and robust enough to answer correctly all their queries, as it has been observed in investigations based on the Wizard of Oz experiment. However, results are usually disappointing because of the extremely difficult problems that NLIDBs have to cope with.

Despite NL processing is one of the main fields of artificial intelligence, progress has been frustratingly slow, because researchers have underestimated the problems involved in NL processing.

Many interfaces have used different types of approaches and architectures for addressing existing problems, but they have not been satisfactory enough to date so as to achieve a performance (i.e., recall) close to 100 %. Furthermore, some NLIDB have design flaws that render them inefficient for large and complex databases (as illustrated in Subsects. 3.4 and 3.5).

In this section we describe the most important pitfalls that prospective users of NLIDBs should be aware of, grouped in six main categories: type of graphical user interface, domain indpendence, customization, escalability, translation process, and performance evaluation.

3.1 Type of Graphical User Interface

The purpose of using different types of GUI design techniques is facilitating user-NLIDB communication. As it has been mentioned, the easiest interface to use is the one that permits introducing a query in the command line; for this type of interfaces, their main limitation is that they do not permit a sophisticated user-interface interaction.

There exist many menu-based interfaces like the one described in [12]. Though it is desirable that an interface guides the user along query formulation, this type of interfaces does not permit formulating queries in free-text NL. This technique may require users to go through several steps for formulating a query, which might be lengthy.

On the other hand, the introduction of queries by voice (like the one used in EasyAsk) requires voice recognition that obtains a performance close to 100 %, which remains being a challenge.

The use of different techniques for developing GUIs provides more functionality to NLIDBs; however, adding more functionality to these systems involves the use of more computer resources and usually requires more training for operating them, which may result in making their use more difficult.

3.2 Domain Independence

Achieving domain independence in NLIDBs requires customizing them for a new or different domain. Additionally, it is very difficult for domain-independent interfaces to obtain recalls above 90 %, and achieving this performance usually involves a lengthy and complex customization process.

For making shure that a NLIDB is domain-indpendent, it must be tested with different databases, and these should have a complexity similar to databases found in large businesses. The complexity of a database can be measured by the number of tables, the overall number of columns, the number of columns with the same or similar information (for example, the number of columns for storing people names), and the structure of relations among tables (number of relations and cycles in the graph that represents the DB schema).

Despite the large number of NLIDBs developed, there only exist a small number of databases that have been used for testing them. Most have been evaluated with the Tang and Mooney databases [13], which involve three different domains (geography, restaurants and jobs). These databases were implemented in Prolog, do not have a relational structure, and their structure is simple since they have from 1 to 8 tables.

One of the most complex databases used for evaluation is ATIS [14] which is a relational DB that stores information on airline flights. Its complexity resides in its structure, since it involves 27 tables and 123 columns. Additionally, the graph that represents the DB schema has many relations among tables and cycles, and 85 % of the query corpus involves semantic ellipsis.

In conclusion, potential users of a NLIDB should find out if it has been tested with several databases of different domains and how good is its performance with complex databases.

3.3 Customization

The customization process consists of populating the data dictionary of a NLIDB with the information necessary for correctly interpreting queries formulated by users. This information consists of the association of elements of the DB schema (tables, columns and relations) with words/phrases that occur in NL queries.

Some interfaces have tools for carrying out a semiautomatic customization (such as EasyAsk, ELF, EnglishQuery, etc.), some others use learning techniques (like those described in [15–19]). However, these types of customization are not good enough for answering all the queries formulated by users; therefore, it is usually necessary to fine-tune the customization for improving interface performance.

Other interfaces (like C-Phrase) have an authoring tool that facilitates the customization process; however, a factor that greatly affects the quality of the customization is the ability of the customizer. Therefore, a desirable feature in a NLIDB is that it should have a tool that permits an easy and quick customization, and that it renders a customization quality that is independent of the customizer ability.

NLIDBs that use supervised learning techniques for query translation need to be trained with a set of queries, which can be considered as a customization process. Unfortunately, the performance of this type of NLIDBs depends on the corpus of training queries, which leads to two problems: selecting an adequate training corpus and fine-tuning the learned knowledge when the interface fails in answering some queries.

It is expected that users with a bachelor's degree in engineering in computer science be able to customize a NLIDB. However, a desirable feature of a NLIDB, concerning customization, is that it does not require from the customizer to learn neither specialized concepts (for example, linguistic terminology, logic programming terms, etc.) nor the inner workings of the interface.

3.4 Scalability

Some NLIDBs use approaches that are not adequate for very large databases; i.e., those with tables that have over one million rows. For example, some interfaces look into the database for search values present in queries in order to identify the column name needed for generating the SLQ expression when translating from NL to SQL. This approach, though effective for small databases, is not efficient for large ones.

Most of the NLIDBs have been tested with small databases that contain tables with less than 100,000 rows. In a test carried out with ELF, a database was used that included verbs, conjugations and lexical tags from Freeling [10], which contained approximately 497,000 rows in a table with three columns. In this test the compilation of the dictionary took around 5 min, and the result was that ELF could not answer any query, because the translation approach is not adequate for dealing with large databases.

3.5 Translation Process

The translation process of a NLIDB must be able to deal with all the possible problems that may occur when translating a NL query. Unfortunately, the complexity of NL query processing has been underestimated; therefore, most NLIDBs do not effectively deal with all the problems, rendering an unsatisfactory performance, especially for decision making in businesses.

In some NLIDBs statistical and machine learning approaches are used for translating queries; however, these techniques involve some degree of uncertainty, which has caused that their recalls reach only 60–85 %.

Other NLIDBs use templates or pattern matching. These types of approaches limit their performance, since they can only permit translating those queries that are predefined in the templates or those that match the pre-established patterns.

There exist some processing flaws that may affect query translation; for example, NLIDBs whose translation approach requires looking into the database for search values present in queries. In this translation approach, each value stored in the database is saved in the data dictionary together with the id of the column where the value is stored. This approach has a major drawback: when a new value is stored in the database, then the NLIDB will not be able to translate a query that involves this value because it will not be found in the data dictionary. Therefore, to mitigate this situation it is necessary to frequently update the data dictionary for databases that have many insertions and updates.

For exemplifying this situation, ELF was tested using the Northwind database. In this experiment a new row was inserted in table *Products* that contained the value "Cheddar" in column *ProductName*, and next the following query was formulated:

How much does Cheddar cost?

In this case the interface could not identify the column *ProductName* needed for generating the SQL expression because "Cheddar" was not in the data dictionary.

Another type of error occurs when a value is present in two or more DB columns. For example, in this experiment table *Products* had a row that contained the value “Chang” in column *ProductName*, next a row was inserted into table *Employees* that had the same value “Chang” in column *LastName*, then upon formulating the following query:

Show the data of the employee Chang

The interface got confused because it found the value “Chang” in tables *Products* and *Employees* and generated the following SQL expression (which involves information from tables *Employees*, *Products*, *OrderDetails* and *Orders*): *SELECT DISTINCT Employees.EmployeeID, OrderDetails.OrderID, Orders.ShipName, Employees.LastName FROM Orders, Employees, OrderDetails, Products, Orders INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID, Orders INNER JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID, OrderDetails INNER JOIN Products ON OrderDetails.ProductID = Products.ProductID WHERE* (*Products.ProductName* = “*Chang*”).

Finally, prospective users of NLIDBs should be aware that, in general, the more complex a database is the more likely it is that an interface will fail when translating a query from NL to SQL.

3.6 Evaluation

One of the problems that has prevented the progress of NLIDB technology is the lack of evaluation standards and well established benchmarks.

Another situation that usually occurs in NLIDB evaluation is that most interfaces are customized and tested by the authors, which does not guarantees the impartiality of the results.

Potential users of NLIDBs should be aware that there exist several metrics for measuring the performance of interfaces: accuracy, recall and F1. The most widely used for evaluating performance is accuracy, which is defined as the percentage of correctly answered queries with respect to the number of translated queries. However, this metric is not the one that end users should be more interested in but recall, which is defined as the number of correctly, answered queries with respect to all the queries input to the interface. It is important to point out that the value of recall is less than or equal to accuracy, and it is usually the case that some NLIDBs report very good accuracies but their recalls are not so good.

Concerning performance measurement, it is difficult to compare the performance reported for different NLIDBs, since there is no uniformity on what a *correct answer* means. The evaluations of some interfaces consider an answer *correct* even if it includes information additional to the one requested, while other evaluations consider an answer correct only if it includes just the expected information. This distinction might be extremely important for a business that intends using a NLIDB for critical decision making.

An example of this situation is shown in a test carried out using ELF and the Northwind database, where the following query was formulated:

Show the units in stock of product Chang

Though the phrase "units in stock" explicitly refers to column *Products.UnitsInStock*, the interface considers the word "units" as referring also to column *OrderDetails.Quantity*, and it also adds to the output the name of the product and the order number; which were not requested in the query, as the resulting SQL expression shows: *SELECT DISTINCT OrderDetails.Quantity, Products.UnitsInStock, OrderDetails.OrderID, Products.ProductName FROM OrderDetails, Products, OrderDetails INNER JOIN Products ON OrderDetails.ProductID = Products.ProductID WHERE (Products.ProductName =* "*Chang*").

Similarly to the previous example, when the following query was formulated:

How much does Chai cost?

One would expect that the result of the query only included the price of product "Chai"; however, the interface returned all the information of the row that contains "Chai", as the following resulting SQL expression shows: *SELECT DISTINCTROW Products.* FROM Products WHERE* (*Products.ProductName* = "*Chai*").

Strictly, there should only exist two types of evaluation for a query translation: correct and incorrect, which is what it really matters for end users. However, there exist performance evaluations, such as those reported in [20], where the interface generates more than one semantic interpretation of a query, in which the answer is evaluated with some percentage ρ of recall (where $0 < \rho < 100$ %). Considering partially-correct answers is not adequate, since end users are usually interested in fully correct answers, not partially correct answers. Another detail that potential users of NLIDBs should be aware of is that the evaluation of some interfaces report recall values, which are actually accuracy figures (see definition at the beginning of this subsection).

4 Conclusion

A little over 30 years ago, considerable progress in NLIDB technology was made. However, NLIDBs have not been widely accepted as expected because of the complexity of NL processing, customization and internal operation, which has resulted in unsatisfactory performance (low percentage of correctly answered queries).

Currently, the information contained in databases has achieved great importance in many areas (especially businesses), which has fostered the development of NLIDBs for facilitating inexperienced users accessing information.

In this chapter, the features that a DB administrator should seek when considering the use of a NLIDB are presented, which are: operability, authoring, habitability, transparency, robustness, efficiency, accuracy, intelligence, multimodality, and independence. Though these characteristics are very important, it is difficult that an interface includes all of them. Even with existing technology, developing a robust enough interface requires solving most of the problems, which is a complex task.

One of the main flaws in the design of NLIDBs is the approach used for dealing with the translation process, which has caused the stagnation of this research area: recall of most NLIDBs lies in the range 60–85 %, and it has been extremely difficult to increase it. Unfortunately, many approaches for NLIDB design have overlooked the complexity of the problems involved. Therefore, we think that an approach for dealing with very complex problems has to be used; for example, an approach that analyzes all the problems that may occur in NL to SQL translation and affective techniques for dealing with each problem.

References

1. Pazos, R., González, J., Aguirre, M., Martínez, J., Fraire, H.: Natural language interfaces to databases: an analysis of the state of the art. Recent Adv. Hybrid Intell. Syst. Stud. Comput. Intell. **451**, 463–480 (2013)
2. Reis, P., Mamede, N., Matias, J.: Edite: a natural language interface to databases: a new dimension for an old approach. In: Proceedings of the 4th International Conference on Information and Communication Technology in Tourism (1997)
3. Pazos, R., González, J., Aguirre, M.: Semantic model for improving the performance of natural language interfaces. In: Proceedings of the MICAI 2011 Mexican International Conference on Advances in Artificial Intelligence, pp. 277–290 (2011)
4. Jain, H.: Hindi language interface to databases. Master's thesis, Thapar University (2011)
5. Kovacs, L.: SQL generation for natural language interface. J. Comput. Sci. Control Syst. **2**(18), 19–22 (2009)
6. Meng, X., Wang, S.: NChiql: the Chinese natural language interface to databases. Lecture Notes in Computer Science 2113, pp. 145–154 (2001)
7. Boldasov, M., Sokolova, E., Malkovsky, M.: User query understanding by the InBASE system as a source for a multilingual NL generation module. Lect. Notes Comput. Sci. **2448**, 33–40 (2002)
8. Jung, H., Geunbae, G.: Multilingual question answering with high portability on relational databases. In: Proceedings Conference on Multilingual Summarization and Question Answering 19, pp. 1–8 (2002)
9. Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., Steedman, M.: Inducing probabilistic CCG grammars from logical form with higher-order unification. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, pp. 1223–1233 (2010)
10. Padró Ll., Stanilovsky, E.: FreeLing 3.0: towards wider multilinguality. In: Proceedings of the Language Resources and Evaluation Conference (2012)
11. Pakray, P.: Keyword based multilingual restricted domain question answering. Master's thesis, Jadavpur University (2007)
12. Hallet, C., Scott, D., Power, R.: Composing questions through conceptual authoring. Comput. Linguist. **33**, 105–133 (2007)
13. Tang, L., Mooney, R.: Using multiple clause constructors in inductive logic programming for semantic parsing. In: Proceedings of 12th European Conference on Machine Learning, pp. 466–477 (2001)
14. Price, P.: Evaluation of spoken language systems: the ATIS domain. In: Proceedings of the DARPA Speech and Natural Language Workshop, pp. 91–95

15. Clarke, J., Goldwasser, D., Chang, M.W., Roth, D.: Driving semantic parsing from the world's response. In: Proceedings 14th Conference on Computational Natural Language Learning, pp. 18–27 (2010)
16. Giordani, A., Moschitti, A.: Translating questions to SQL queries with generative parsers discriminatively reranked. In: Proceedings of the Conference on Computational Linguistics (Posters), 401–410 (2012)
17. Kate, R., Mooney, R.: Using string-kernels for learning semantic parsers. In Proceedings of 21st ICCL and 44th Annual Meeting of the Association for Computational Linguistics, pp. 913–920 (2006)
18. Liang, P., Jordan, M., Klein, D.: Learning dependency-based compositional semantics. In: Proceedings of 49th Annual Meeting of the Association for Computational Linguistics, pp. 590–599 (2011)
19. Lu, W., Tou, H.N., Lee, W.S., Zettlemoyer, L.: A generative model for parsing natural language to meaning representations. In Proceedings of Conference on Empirical Methods in Natural Language Processing, pp. 783–792 (2008)
20. Kaufman, E., Bernstein, A., Fischer, L.: NLP-Reduce: A "naïve" but domain-independent natural language interface for querying ontologies. In: Proceedings of the 4th European Semantic Web Conference, pp. 1–2 (2007)