A Natural Language Query Processor for Database Interface

Mrs. Vidya Dhamdhere
Lecturer department of Computer Engineering Department
G.H. Raisoni college of Engg. (Pune University)
vidya.dhamdhere@gmail.com

Nijesh Hirpara (nijesh.hirpara@gmail.com)
Kalpesh Surana (kalpeshsurana@gmail.com)
Karishma Gangwani (karishmagangwani@yahoo.com)
Chirayu Bootley (chirayu.bootley24@gmail.com)
G.H.Raisoni college of Engineering and Management (Pune University)
Pune, India

Abstract—With the increasing amount of information available on the Internet one of the most challenging tasks is to provide search interfaces that are easy to use without having to learn a specific syntax. Hence, we present a query interface exploiting the intuitiveness of natural language for the largest web-based tourism platform.

Various analyses shows how users formulate queries when their imagination is not limited by conventional search interfaces with structured forms consisting of check boxes, radio buttons and special-purpose text fields. The results of this field test are thus valuable indicators into which direction the web-based tourism information system should be extended to better serve the customers.

Keywords—Tourism information system; Natural language processing; User behavior study; User interface; Web-based information system.

I. INTRODUCTION

The development and availability of efficient and appropriate search functions are still a challenge in the field of database and information systems. Consider, for example, the context of tourism information systems where intuitive search functionality plays a crucial role for the economic success. The reason, obviously, is that users, i.e. tourists, are often computer illiterate, such that formal query languages like SQL in database systems or Boolean logic in retrieval systems are an enormous barrier. This is further impaired by the development of search engines that have subtle differences in their functionality that are not apparently visible (Shneiderman et al., 1998). When faced with a natural language interface, however, these users are able to express their information needs as they are used to when interacting with a human travel agent.[5]

Natural language processing is becoming one of the most active areas in Human-computer Interaction. It is a branch of AI which includes Information Retrieval, Machine Translation and Language Analysis. The goal of NLP is to enable communication between people and computers without resorting to memorization of complex commands and procedures. In other words, NLP is a technique which can make the computer understand the languages naturally used by humans. While natural language may be the easiest symbol system for people to learn and use, it has proved to be the hardest for a computer to master. Despite the challenges, natural language processing is widely regarded as a promising and critically important endeavor in the field of computer research.

The general goal for most computational linguists is to instill the computer with the ability to understand and generate natural language so that eventually people can address their computers through text as though they were addressing another person.[3] The applications that will be possible when NLP capabilities are fully realized are impressive computers would be able to process natural language, translating languages accurately and in real time, or extracting and summarizing information from a variety of data sources, depending on the users’ requests.

The two design goals for the interface were, first, to provide multilingual access, and second, to develop a generic framework independent of a particular application domain. To achieve these goals, we strictly separated natural language query analysis from the domain-specific processing logic. The analysis process identifies relevant parts of the query using language-dependent ontologies describing the concepts of the application domain. The domain-specific processing logic defines how these relevant parts are related to each other and builds the appropriate database query. Further features of our
system are that additional languages can be added conveniently to the information system and the interaction with the system can be designed with respect to the differing capabilities of various client devices such as web browsers or PDAs.

II. OBJECTIVES

Query is the essential function provided by the program, which all the other facilities are designed to support. We use "query" to mean the process of typing in a question about the information in your database, and its translation into the "native" data access language (SQL); it also includes the automatic formatting of the answer in any of several response styles including datasheet, form, worksheet or graph.

III. RELATED WORK

The very first attempts at NLP database interfaces are just as old as any other NLP research. In fact database NLP may be one of the most important successes in NLP since it began. Asking questions to databases in natural language is a very convenient and easy method of data access, especially for casual users who do not understand complicated database query languages such as SQL. The success in this area is partly because of the real-world benefits that can come from database NLP systems, and partly because NLP works very well in a single-database domain. Databases usually provide small enough domains that ambiguity problems in natural language can be resolved successfully. Here are some examples of database NLP systems: LUNAR (Woods, 1973) involved a system that answered questions about rock samples brought back from the moon. Two databases were used, the chemical analyses and the literature references. The system was informally demonstrated at the Second Annual Lunar Science Conference in 1971. [1] LIFER/LADDER was one of the first good database NLP systems. It was designed as a natural language interface to a database of information about US Navy ships. This system, as described in a paper by Hendrix (1978), used a semantic grammar to parse questions and query a distributed database. The LIFER/LADDER system could only support simple one-table queries or multiple table queries with easy join conditions. [4]

IV. NATURAL LANGUAGE QUERY PROCESSING

This section contains a brief description of the various steps performed during natural language query processing of the demonstrator system. The software architecture is designed according to a pipeline structure as shown on the right hand side of Figure 1.

Language Identification

To identify the language of a query, we use an n-gram-based text classification approach (Cavnar and Trenkle, 1994) where each language is represented by a class. An n-gram is an n-character slice of a longer character string. As an example, for n = 3, the tri-grams of the string "language" are: {_la, lan, ang, ngu, gua, uag, age, ge_}. Dealing with multiple words in a string, the blank character is usually replaced by an underscore “_” and is also taken into account for the construction of an n-gram document representation.

This language classification approach using n-grams requires sample texts for each language to build statistical models, i.e. n-gram frequency profiles, of the languages. [2] We used various tourism-related texts, e.g. hotel descriptions and holiday package descriptions, as well as news articles both in English and German. The n-grams, with n ranging from 1 to 5, of these sample texts were analyzed and sorted in descending order according to their frequency, separately for each language. These sorted histograms are the n-gram frequency profiles for a given language.

In the English text, {the, and, of, in} and the suffix {ion} are the most frequent tri-grams. Contrarily, in the German texts, the most frequent tri-grams are endings like {en, er, ie, ch} and words like {der, ich, ein}.

To determine the language of a query, the n-gram profile, n = 1 … 5, of the query string is built as described above. The distance between two n-gram profiles is computed by a rank-order statistic. For each n-gram occurring in the query, the difference between the rank of the n-gram in the query profile and its rank in a language profile is calculated. For example, the tri-gram {the} might be at rank five in a hypothetical query but is at rank two in the English language profile. Hence, the difference in this example is three. These differences are computed analogously for every available language.
The sum of these differences is the distance between the query and the language in question. Such a distance is computed for all languages, and the language with the profile having the smallest distance to the query is selected as the identified language, in other words, the most probable language of the query. If the smallest distance is still above a certain threshold, it can be assumed that the language of the query is not identifiable with a sufficient accuracy. In such a case the user will be asked to rephrase her or his query.

Error Correction

To improve the retrieval performance, potential orthographic errors and misspellings have to be considered in our web-based interface. After identifying the language we use a spell-checking module to determine the correctness of the query terms. The efficiency of the spell checking process improves during the runtime of the system by learning from previous queries. The spell checker uses the metaphone algorithm (Philips, 1990) to transform the words into their soundalikes. Because this algorithm has originally been developed for the English language, the rule set defining the mapping of words to the phonetic code has to be adapted for other languages. In addition to the base dictionary of the spell checker, domain-dependent words and proper names like names of cities, regions or states, have to be added to the dictionary.

For every misspelled term of the query, a list of potentially correct words is returned. First, the misspelled word is mapped to its metaphone equivalent, then the words in the dictionary, whose metaphone translations have at most an edit distance (Levenshtein, 1966) of two, are added to the list of suggested words. The suggestions are ranked according to the mean of:

• The edit distance between the misspelled word and the suggested word, and
• The edit distance between the misspelled word's metaphone and the suggested word's metaphone.

The smaller this value is for a suggestion, the more likely it is to be the correct substitution from the orthographic or phonetic point of view. However, this ranking does not take domain-specific knowledge into account.

Because of this deficiency, correctly spelled words in queries are stored and their respective number of occurrences is counted. The words in the suggestion list for a misspelled term are looked up in this repository and the suggested word having the highest number of occurrences is chosen as the replacement of the erroneous original query term. In case of two or more words having the same number of occurrences the word that is ranked first is selected. If the query term is not present in the repository up to this moment, it is replaced by the first suggestion, i.e. the word being phonetically or orthographically closest. Therefore, suggested words that are very similar to the misspelled word, yet make no sense in the context of the application domain, might be rejected as replacements. Consequently, the word correction process is improved by dynamic adaptation to past knowledge.

Another important issue in interpreting the natural language query is to detect terms consisting of multiple words. Proper names like “St. Anton am Arlberg” or substantives like “swimming pool” have to be treated as one element of the query. Regular expressions are used to identify such cases.

SQL Mapping

With the underlying relational database management system PostgreSQL, the natural language query has to be transformed into a SQL statement to retrieve the requested information. The knowledge base of the domain is split into three parts. First, we have an ontology specifying the concepts that are relevant in the application domain and describing linguistic relationships like synonymy. Second, a lightweight grammar describes how certain concepts may be modified by prepositions, adverbial or adjectival structures that are also specified in the ontology. Finally, the third part of the knowledge base describes parameterized SQL fragments that are used to build a single SQL statement representing the natural language query.

The query terms are tagged with class information, i.e. the relevant concepts of the domain (e.g. “hotel” as a type of accommodation or “sauna” as a facility provided by a hotel), numerals or modifying terms like “not”, “at least”, “close to” or “in”. If none of the classes specified in the ontology can be applied, the database tables containing proper names have to be searched. If a substantive is found in one of these tables, it is tagged with the respective table's name, such that “Tyrol” will be marked as a federal state.

In the next step, this class information is used by the grammar to select the appropriate SQL fragments. To illustrate this processing step, consider the following SQL fragment as the condition for an accommodation being located in or not in a particular city, where @OP is a placeholder for an operator and @PARAM for the city name.

```
SELECT entity."EID" FROM entity WHERE entity."CID" = city."CID" AND city."Name" @OP @PARAM
```

Depending on modifying terms found in the query as specified in the grammar, the SQL fragment is selected and the parameters are substituted with the appropriate values. A query for accommodation in Innsbruck produces the following fragment.

```
SELECT entity."EID" FROM entity WHERE entity."CID" = city."CID" AND city."Name" = 'Innsbruck'
```

Finally, the SQL fragments have to be combined to a single SQL statement reflecting the natural language query of the user. The operators combining the SQL fragments are again chosen according to the definitions in the grammar.
V. SYSTEM IMPLEMENTATION PLAN

VI. SYSTEM DESIGN

VII. DESIGN CONSIDERATION FOR THE SYSTEM INTERFACE.

The graphical User Interface is as shown below. The user has to first login and then connect to the database. A database setting is required to access the database after getting the information about host name, database name, user name and the password. Once the database is setup, English Query can translate very complex English queries to SQL with the capability of searching multiple tables and multiple fields. Steps followed to get the result:-

- Type the natural language query in the dialogue box given.
- Click on “Proceed” button.
- The system will ask the user for the expected meaning. In case of ambiguities the user has to select the desired query.
- Click on “Generate SQL” button.
- The system will generate SQL query.
- Click on “Run Query”.
- The result of the query will be displayed in the Output box.

Our major design goal at the outset of the project was to provide a simple and easy to use interface. Hence, the interface is dominated by a text-box where the user can enter her or his query and a submit button, the latter one labeled with “ask”. During the field trial we additionally provided short textual descriptions in both German and English in form of sample queries. The sample query “I am looking for a double room in the center of Salzburg with indoor pool.” is the only hint on the capabilities of the interface. The intention was to collect a broad range of accommodation requests and, thus, to find out what the users really want. Our aim was not to bias the user’s imagination when formulating a query. This, admittedly, with the risk of disappointing the user when no or just inappropriate results were found.

We also implemented the look and feel of the design in order to avoid distraction from the user’s task. On the result screen (see Figure 4), we present the original query as well as the concepts identified by the natural language processing to provide the user with feedback regarding the quality of natural language analysis. Below the list of accommodations matching the criteria, we have provided a feedback form where users can enter a comment and rate the quality of the result. After the field test, it turned out that only 3.37% of the queries have either been annotated or rated where the numbers of positive and negative comments were nearly equal. Due to the unsupervised nature of the test without any reward for the users, this figure is not surprising because of the additional time it takes to assess the quality of the result and then comment on it. At the bottom of the page, the input field prefilled with the posed query is presented to allow for convenient query reformulation or refinement. About 10% of the queries were modified by adding or deleting parts of the original query.
CONCLUSION

Natural Language Processing can bring powerful enhancements to virtually any computer program interface. This system is currently capable of handling simple queries with standard join conditions. Because not all forms of SQL queries are supported, further development would be required before the system can be used within NLQP.

REFERENCES


