Using semantic templates for a natural language interface to the CINDI virtual library

Article *in* Data & Knowledge Engineering · October 2005

DOI: 10.1016/j.datak.2004.12.002 · Source: DBLP

3 authors, including:

Leila Kosseim
Concordia University Montreal
114 PUBLICATIONS **636** CITATIONS

Available online at www.sciencedirect.com
SCIENCE DIRECT®
Data & Knowledge Engineering 55 (2005) 4–19
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

Using semantic templates for a natural language interface to the CINDI virtual library

Niculae Stratica *, Leila Kosseim, Bipin C. Desai

Department of Computer Science, Concordia University, 1455 de Maisonneuve Blvd. West, Montreal H3G 1M8, Canada

Received 2 December 2004; accepted 2 December 2004
Available online 23 December 2004

Abstract

In this paper, we present our work in building a template-based system for translating English sentences into SQL queries for a relational database system. The input sentences are syntactically parsed using the Link Parser, and semantically parsed through the use of domain-specific templates. The system is composed of a pre-processor and a run-time module. The pre-processor builds a conceptual knowledge base from the database schema using WordNet. This knowledge base is then used at run time to semantically parse the input and create the corresponding SQL query. The system is meant to be domain independent and has been tested with the CINDI database that contains information on a virtual library.

Keywords: Natural language processing; Syntactic analysis; Semantic analysis; Relational data base; CINDI; Natural language interface

* Corresponding author.
E-mail addresses: nstratica@primus.ca (N. Stratica), kosseim@cs.concordia.ca (L. Kosseim), bcdesai@cs.concordia.ca (B.C. Desai).

0169-023X/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2004.12.002

1. Introduction

1.1. Natural language interfaces to databases (NLIDB)

One of the earliest and most widely studied areas of Natural Language Processing (NLP) is the development of a natural language interface to database systems (NLIDB) (e.g. [11,23]). Such front ends relieve the users of the need to know the structure of the database and offer a much more convenient and flexible interaction as the user is not required to learn a new query language. These features make NLIDB even more attractive today due to the increased computing throughput of today's systems. This is why much work is still performed in this area (e.g. [1,7,10]).

Compared to NLIDB, open-domain question–answering (QA) is a fairly recent field in NLP, but the approaches typically used in this domain can be very useful in NLIDB. Recent developments in QA (e.g. [21]) have made it possible for users to ask a fact-based question in natural language (e.g. Who was the Prime Minister of Canada in 1873?) and receive a specific answer (Alexander Mackenzie) rather than an entire document where the users must further search for the specific answer themselves. In this respect, QA can be seen as the next generation of handy tools to search huge text collections such as the Internet [13].

1.2. NLIDB versus question–answering

Natural language interfaces to database systems and question answering systems differ fundamentally in their scientific goals and their technical constraints. QA systems try to answer a natural language question by analyzing a collection of unrestricted and unstructured texts; while NL interfaces to DB have the advantage of dealing with structured texts; that is, texts that have already been decomposed semantically and entities and relationships have already been identified. However, both NLIDB and QA systems share an interesting similarity: they both take as input a question formulated in natural language and must interpret it in order to answer it properly. The advantage of the QA domain is that standard metrics do exist to evaluate and compare different approaches. Since 1999, the TREC conference, organized annually by the National Institute of Standard and Technology, evaluate competing QA systems on a standard question set and document collection [17–20]. In 2002, the best scoring system [14] at the TREC conference had a *confidence-weighted* score of 0.856 (out of a maximum of 1). With a corpus of 500 questions, the system was able to find 415 correct answers and exact answers in a collection of 1 million documents (≈3 Gigabytes of text) [20].

Analyzing an input question in QA and in NLIDB systems is often based on a part-of-speech (POS) tagging, followed by a syntactic analysis (partial or full) and finally, a more or less precise semantic interpretation. Although there exist standard techniques for POS tagging (e.g. [2,8,9]) and syntactic analysis (e.g. [16,8]), techniques for semantic parsing are still very varied and ad hoc. In an open-domain situation, where the user can ask questions on any topic, this task is often very tentative and relies mainly on lexical semantics only. However, when the discourse domain is limited (as is the case of NLIDB), the interpretation of a question becomes easier as the space of possible meanings is smaller, and specific templates can be used [22]. We believe that meta-knowledge of the database, namely the schema of the database, can be used as an additional resource to better interpret the question in a limited domain. The two strategies described above are presented in Fig. 1.

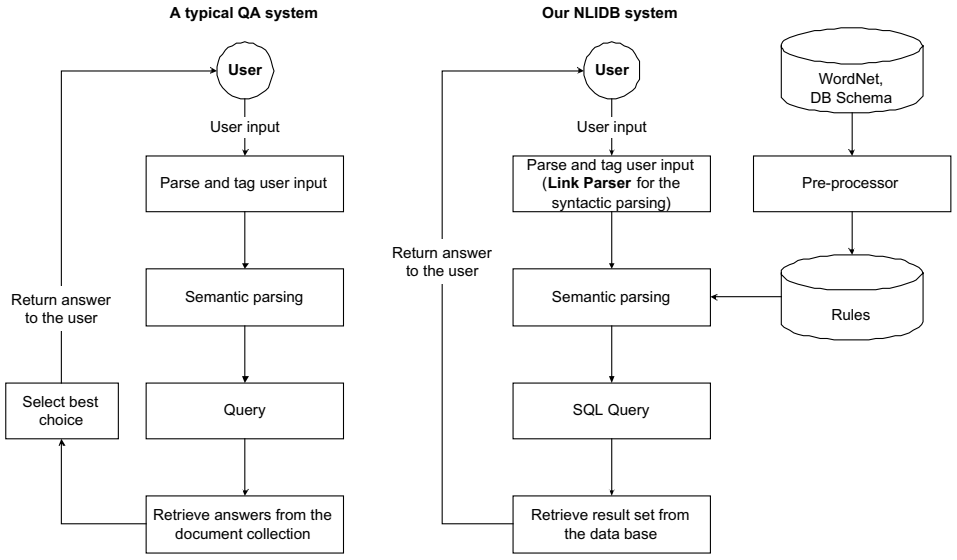

Fig. 1. Left: The process flow of a typical QA system. Right: The process flow of the NL interface to a database. The semantic parsing makes use of the schema.

The left side of Fig. 1 illustrates the general architecture of a QA system. The processing is linear, going through the stages detailed above (POS tagging, syntactic parsing and semantic interpretation). The right side of Fig. 1 shows how semantic interpretation can be improved by making use of the meta-information. Once the question has been analyzed, the system tries to match the template representation to the knowledge base created by the pre-processor and to generate the SQL query.

2. Our proposed model for CINDI

CINDI, the digital library system being developed at Concordia University [3], is a virtual library built to facilitate the registration of digital resources and their subsequent bibliographic search [5,6]. CINDI is based on the use of Semantic Headers [4] that store relevant information for search and discovery: these are the usual search terms such as author's name, title of the contents of the digital resources, its subject classification, abstract etc. stored in CINDI's Semantic Header database. CINDI's interface sub-system uses an expert system to guide the user in the search tasks at hand. Once the user input is processed, the CINDI system uses the Semantic Headers database to retrieve the information from the resource catalog. CINDI thus provides a mechanism to register, manage and search a bibliography and provide access to the actual resources once the search is successful. The focus of the current paper is to report on the design and implementation of a Natural Language interface for the CINDI system.

The detailed design of the natural language processor is presented in Fig. 2.

The questions are tagged and syntactically parsed using the Link Parser [16] and semantically parsed through the use of semantic templates that we developed (see Section 2.2). The template

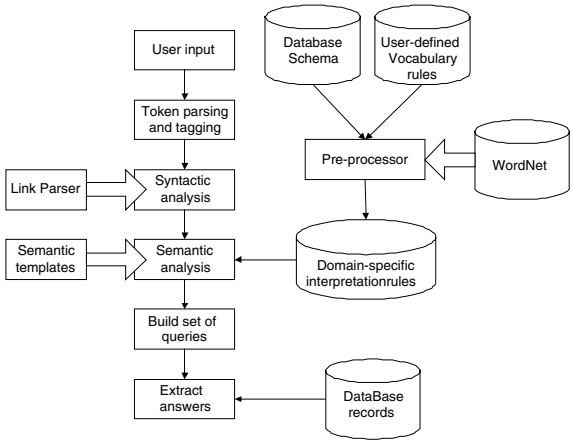

Fig. 2. The design of the natural language processor.

representation of the question is similar to the work of [15] in question answering. Once the question is parsed, its template representation is mapped to the meta-representation of the database constructed by the pre-processor and subsequently translated into an SQL query.

2.1. The pre-processor

As mentioned previously, the role of the pre-processor is to limit the scope of the interpretation of the input by building a knowledge base that is specific to the schema of the database. As shown in Fig. 2, the pre-processor uses the schema of the database and user-defined vocabulary rules coupled with WordNet [12] to create domain-specific interpretation rules (the knowledge base). The Link Parser and the WordNet "C" code has been integrated with a combination of "C" and "C++" code in the NLIDB system.

2.1.1. Reference to CINDI relations

In order for the interface to take into account lexical variations in the input questions, the pre-processor builds a semantic knowledge base composed of interpretation rules and semantic sets for all possible relation and attribute names in the database. This allows us to build the same semantic representation, and hence, the same SQL query for questions such as:

Who is (*are*) *the author*(*s*) *of the book*(*s*) "*Algorithms*"?
Who is (*are*) *the writer*(*s*) *of the book*(*s*) "*Algorithms*"?
Who is (*are*) *the author*(*s*) *of the resource*(*s*) "*Algorithms*"?
Who is (*are*) *the writer*(*s*) *of the resource*(*s*) "*Algorithms*"?

To build the semantic knowledge base, the pre-processor first reads the schema of the database, identifies all relation names and attribute names and finally, uses WordNet [12] to create a list of hyponyms, hypernyms and synonyms for each relation and attribute name. This initial semantic set is then proposed to the system administrator, who can modify it to reflect the specific senses of

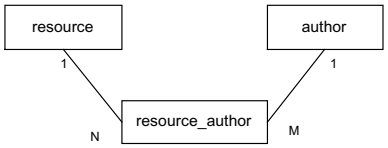

Fig. 3. Example of a relationship and the corresponding relations involving author and resource in the CINDI system.

the database names in the context of the database discourse domain. The adjusted semantic set is then used at run time to find which relation names are most likely to be referred to by the terms in the input sentence.

For example, in the CINDI database, a relation name such as author will trigger the pre-processor to generate the initial semantic set containing, but not being limited to, the following terms {*writer*, *author*, *generator*, *source*, *communicator*, *person*, *individual*, *maker*, *shaper*, *coauthor*, *novelist*, ...}, which the NLIDB system administrator at the time of installation could reduce to: {*author*, *creator*, *generator*, *writer*}. Using semantic sets, the four questions above will produce the same SQL query at run time, because they involve words from the same semantic set.

Let us consider another example. Still in the library domain, let us consider three relations named: `resource`, `author` and `resource_author`. The three relations are shown in Fig. 3.

Assume that the relation resource has the attributes resource id, title and publish date, the relation author has two attributes: author id and name, and the relation resource author has two foreign keys: resource id and author id. With this small schema, the pre-processor will compute the semantic sets of the relation resource, with help from WordNet. The user will instruct the pre-processor to accept the terms {*book*, *document*, *article*, *paper*, *report*, *resource*, *thesis*, *tutorial*} as possible references to resource. If a database relation name cannot be found in WordNet, the system administrator is asked to propose a synonym in the domain, and the pre-processor will be able to find an initial semantic set. For example, with a relation name such as `resource_author` which could not be found in the word list of WordNet, the administrator can identify `requirements` as a likely synonym and the pre-processor can create the appropriate semantic set. Similarly, when two relations are in a many-to-many relationship, then the corresponding relation for this relationship might not have a meaning of its own and it is not always associated with a dictionary word.

2.1.2. References to relations

While the entities in the database are referred to by noun phrases that are semantically mapped to database relations, the relationship between these entities are often referred to by verb phrases. At the pre-processing time, the system administrator identifies the possible verbs that can relate the entities in a specific database schema and uses WordNet again to extract and edit the semantic set of each verb. That is, WordNet is used to search for synonyms, hypernyms and hyponyms of the verb.

2.1.3. Default attributes

Default attributes are used to indicate which attribute is being referred to, when the user input question does not contain an explicit reference to an attribute name. For example, as we saw previously, in CINDI, the verb wrote in *Who wrote The Old Man and The Sea*? relates the relations

author and resource. However, the attributes of interest are not explicitly stated in the question as in *What is the name of the author who wrote the book with the title*: *The Old Man and The Sea*? In order to determine which attributes are referenced when none are explicated, the database administrator designates a default attribute for each relation in the database. In CINDI, the default attribute of the relation author is name, and the default attribute for resource is title.

2.1.4. Relations rules

Relations rules are meant to take into account dependencies between relations; and indicate the general pattern for mapping a user input to an SQL query. For example, given the 3-relation relationship shown in Fig. 3, if the relations resource and author are involved in a SQL query, then the relation resource author must appear as well. The relation rules determine the list of relation and the list of conditions in the final SQL query. The rule associated with the relation configuration of Fig. 3 is:

```
If (relation_list contains (author, resource))
then relation_list = relation_list + resource_author
```

Each relation rule is associated with a SQL template that is created by the pre-processor and used at run time to build the final SQL query with the specific values found in the user input. For example, with the relation of Fig. 3, the associated SQL template will be:

```
SELECT (⟨attribute_list⟩)
FROM author, resource, resource_author
WHERE author.author_id = resource_author.author_id
AND resource.resource_id = resource_author.resource_id
AND ⟨conditions_list⟩
```

2.2. Run-time interpretation

Once the pre-processor has built the interpretation rules, these are used at run time to interpret the input questions. The analysis of the input questions is performed by matching the syntactic parse of the question to a set of fixed templates. This approach is similar to the one used in QA (e.g. [15]).

2.2.1. Syntactic analysis

We use the Link Parser [16] to parse the input question. This particular parser was chosen for three main reasons. First, its grammar coverage and parsing accuracy are high and it is very fast; making it ideal for our application that must analyze questions dynamically. Second, the Link Parser returns a set of all possible parse trees and ranks them in order of likelihood, thus allowing us to build several possible queries or take into account the confidence level of the parser to

evaluate our result. And finally, both the executable and the source code are free,[1] making it easy to incorporate into the CINDI interface.

The input question is first sent to the Link Parser which returns a ranked list of parse trees (from the most likely to the least likely). Fig. 4 shows an example of the Link Parser output for the sentence *List the address of the writer Mark Twain*. Only one parse tree was found.

The same sentence has been passed through the NLIDB system which embeds the 'C' code of the LinkParser engine. The screen output is shown in Fig. 5.

2.2.2. Semantic analysis

Once the input question has been syntactically parsed, we use fixed templates to interpret its meaning. Three semantic templates are used:

```
⟨attribute⟩ of ⟨object⟩
⟨attribute⟩ of ⟨object⟩ of ⟨object⟩
⟨action verb⟩ ⟨object⟩ ⟨attribute value⟩
```

The system tries all parse trees generated by the Link Parser until one matches a semantic template. As objects are mostly mapped linguistically as noun phrases, to match an ⟨object⟩ in a template, we specifically look for tag sequences of the Link Parser that correspond to noun phrases. To match an ⟨attribute⟩, any string will be considered, as long as it forms a full constituent. And to match ⟨action verb⟩, we look for verb phrases in the parse tree. Let us now show each template in detail.

2.2.2.1. The attribute of object template. The ⟨attribute⟩ of ⟨object⟩ template deals with a simple form of input question involving only one attribute in only one relation associated with the semantic set of the object.

In order to translate a question based on this template to its corresponding SQL query, we have associated the following fixed SQL template:

```
SELECT attribute
FROM relationl
WHERE default attribute = ⟨value of relationl.default_attribute⟩
```

Table 1 shows two examples of the ⟨attribute⟩ of ⟨object⟩ template. In the first example, the question *List the address of the writer Mark Twain* is parsed by the Link Parser. Only one parse tree is found (see Figs. 4 and 5). The Link Parser identified *List* as a verb phrase, the *address* as a noun phrase and so on. In order to build the corresponding SQL query, the system tries to apply the ⟨attribute⟩ of ⟨object⟩ template and identifies the noun phrase *the writer* as the

[1] see http://bobo.link.cs.cmu.edu/link/

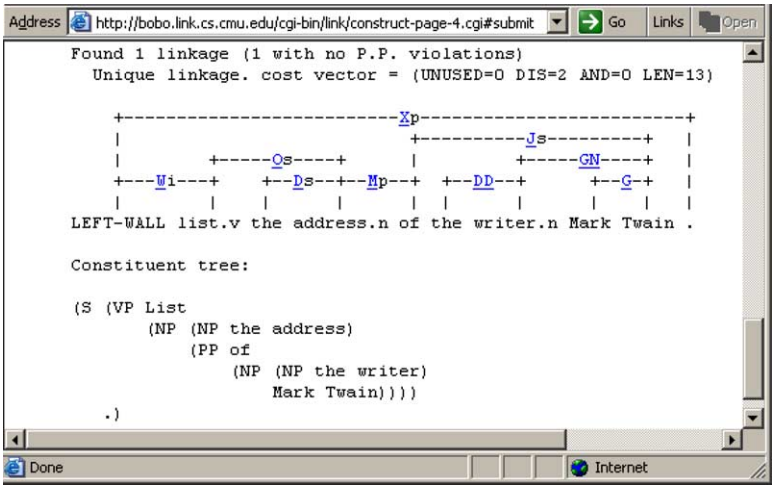

Fig. 4. The output from the on-line Link Parser for the sentence *List the address of the writer Mark Twain*.

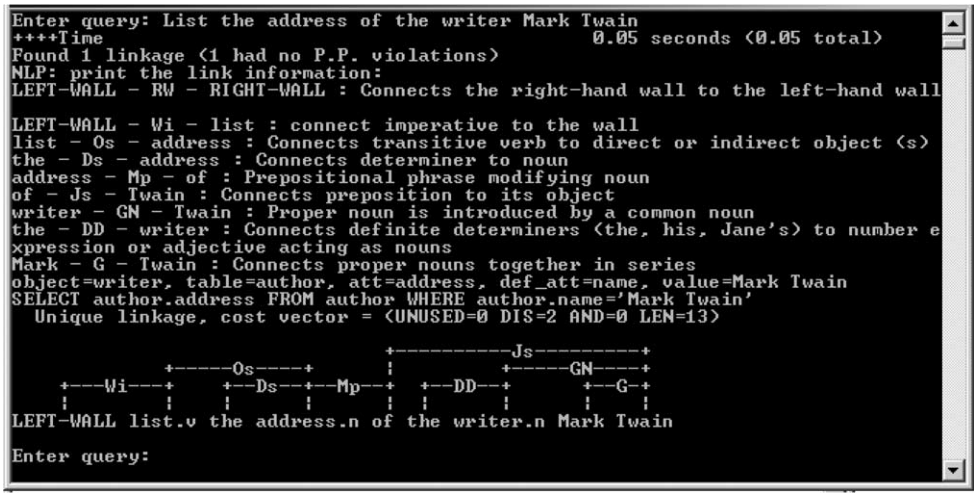

Fig. 5. Actual screen output from the NLIDB system running the same input sentence as the one shown in Fig. 4.

Table 1
Two examples of the ⟨attribute⟩ of ⟨object⟩ template

Natural language question	SQL query
List the address of the writer Mark Twain *Attribute Relation*	*SELECT address* *FROM author* *WHERE name = Mark Twain*
What is the size of the book "Algorithms"? *Attribute Relation*	*SELECT size* *FROM resources* *WHERE title = 'Algorithms'*

Table 2
Examples of questions matching the ⟨attribute⟩ of ⟨object⟩ of ⟨object⟩ template

Natural language question	SQL query
List the address of the author of the book 'Astronomy'. *Attribute Relation1 Relation2 Value of Default Attribute*	SELECT author.address FROM author, resource, resource_author WHERE resource.title = 'Astronomy' AND resource.resource_id = resource_author.resource_id AND author.author_id = resource_author.author_id
What is the name of the author of the book 'Physics'? *Attribute Relation1 Relation2 Value of Default Attribute*	SELECT author.name FROM author, resource, resource_author WHERE resource.title = 'Physics' AND resource.resource_id = resource_author.resource_id AND author.author_id = resource_author.author_id

reference to the relation of interest. Using the semantic set built by the pre-processor, the word *writer* is mapped to the relation author. The noun phrase *the address* is processed similarly and the attribute address of the relation author is identified. The next step is to match *Mark Twain* to the default attribute of the target relation, which is `name`. The final SQL query is indicated in the right-hand column of Table 1.

2.2.2.2. Attribute of object of object template. The ⟨attribute⟩ of ⟨object⟩ of ⟨object⟩ template is meant to take into account questions that involve one explicit attribute from a relation and the default attribute from a second relation. Examples of questions involving this template are given in Table 2.

If the system identifies two consecutive target relations, it tries to match the input question with the ⟨attribute⟩ of ⟨object⟩ of ⟨object⟩ template. The associated SQL template is shown in the following box:

```
SELECT relation1.attribute
FROM relation1, relation2
WHERE relation2.def_att = ⟨value of relation2.def_att⟩
```

Table 2 shows two examples of the ⟨attribute⟩ of ⟨object⟩ of ⟨object⟩ template. For example, with the question *What is the name of the author of the book 'Physics'*? Following the syntactic parses generated by the Link Parser, the system identifies author and resource as the two target relations. The relation author has an attribute which is found in the semantic set of the noun name. For the relation resource the system uses the default attribute, which is title. The generated SQL query is shown in the right-hand column of Table 2.

2.2.2.3. Action verb template. In the previous examples, the verb in the question carried no information as to what database relation to look for. However, some verbs do carry important clues

for the interpretation of the question (e.g. *write*, *borrow*, *call*, ...). The action verb template is used when the verb in the question gives clues as to what relation to look for.

The simplest example of the action verb template involves one action verb and the value of one default attribute, as in *Who wrote the Old Man and the Sea*? In this example, although the system finds no relation name in the query, it does find the action verb *wrote*, which is semantically related to the relations author and resource (see Section 2.1.4). From the default attributes for these CINDI database relations, namely name and title, NLIDB derives the following criteria:

```
Attribute1 = Relation1.default attribute = author.name = result—to be
determined
Attribute2 = Relation2.default  attribute = resource.title = 'the  Old
Man and the Sea'
```

From the schema rules, NLIDB also determines that for the pair of CINDI database relations (`author, resource`) must also involve the database relation (`resource author`). Using these along with the criteria, NLIDB generates the following SQL query:

```
SELECT author.name
FROM author, resource, resource author
WHERE resource.title = 'the Old Man and the Sea'
AND resource author.resource id = resource.resource id
AND resource author.authorid = author.author id
```

3. Implementation

The NLIDB system is being evaluated for use on the CINDI system. The database for CINDI currently contains 15 relations. The system has been developed in C and C++ and uses a command-line interface; but we intend to make the interface available over the Internet through a more convenient interface. Figs. 6 and 7 show two sample outputs of the system. With the input *List books*, the system created the SQL query SELECT `resource.title` FROM `resource`. However, with the input *List the phone of Mark Twain*, no corresponding SQL was generated because the system could not find the relation name.

So far, we have evaluated the system informally, with questions ranging from to 20 words in length. At the present time, we estimate a success rate of about 70% for correct questions having one action verb, one direct object and two modifiers. With three modifiers, the success rate decreases significantly because the input question becomes syntactically more complex and the number of possible parses found by the Link Parser increases exponentially. In this case, the system needs to be improved so as to make a better selection of the parse tree offered by the Link Parser. The results seem promising, but a formal evaluation needs to be performed.

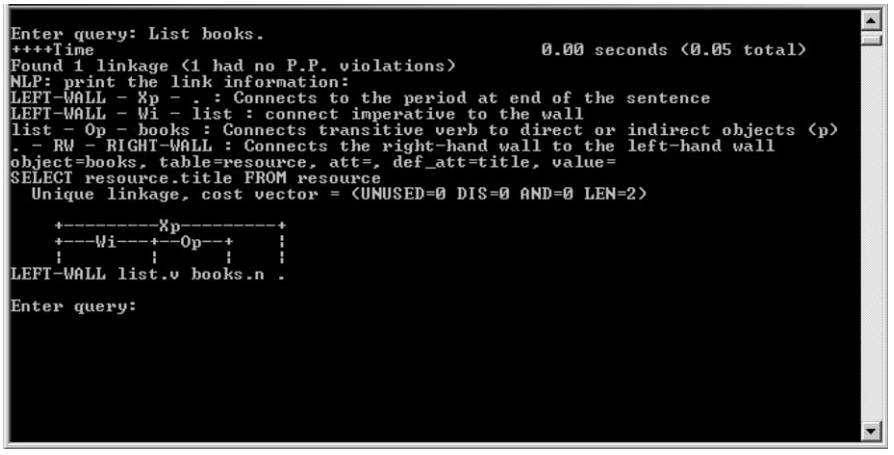

Fig. 6. Example of a correct analysis with the template ⟨attribute⟩ of ⟨object⟩. The NLIDB system constructs the SQL query: SELECT resource.title FROM resource.

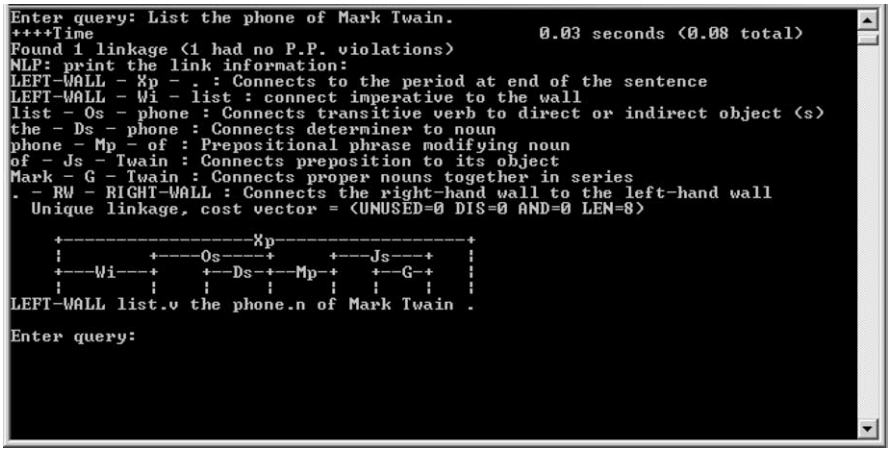

Fig. 7. Example of a failed analysis with the template ⟨attribute⟩ of ⟨object⟩. The relation name was not found and no SQP query has been generated.

3.1. Experimental results

The Natural Language system (NLIDB) was tested on input queries up to 10 words, involving the use of one or two tables from the CINDI database. Fig. 8 shows the actual screen output for the following query: *List books written by Mark Twain*. The Link Parser finds 4 linkages which are processed in order till the resulting SQL returns a non-empty result set, if any.

NLIDB maps the noun books to table `resources` but it cannot map any of the attributes of the table `resources` to the input query. Instead it uses the default attribute which is specified in the rules database i.e. the `resources.title` attribute.

The action verb `written` is mapped to the authors table in the database through the action verbs list. The default attribute for table authors is `author.name` which is given the value

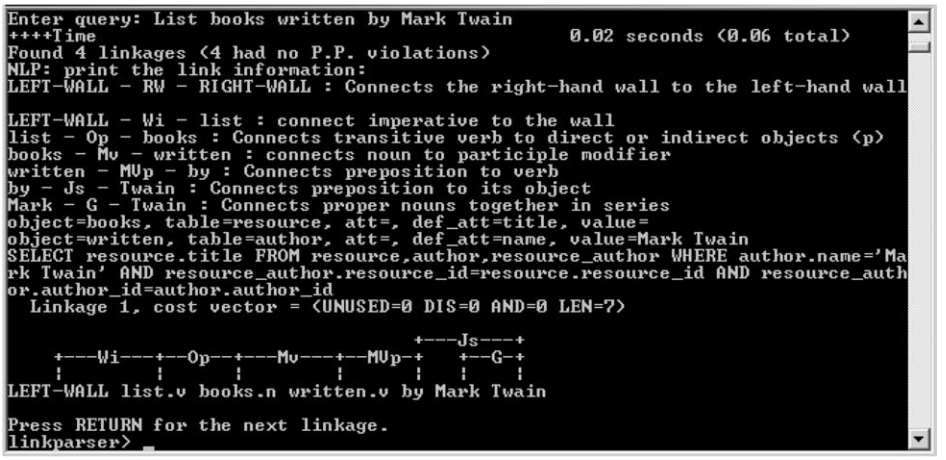

Fig. 8. Actual screen output for the input sentence *List books written by Mark Twain*. The resulting SQL query is: SELECT resource.title FROM...

obtained from the remained of the input query, that is `author.name` = '*Mark Twain*'. The tables `resources` and `author` are related to each other through the `resource_author` relation. This dependency is captured in the rules database. The final SQL query is:

```
SELECT resource.title FROM resource, author, resource_author
WHERE author.name = 'Mark Twain'
```

In the example shown in Fig. 8, all 4 links returned by the Link Parser are valid and generate the same SQL query. This detail is counted for when computing the degree of confidence level for the result.

Table 3 shows actual results returned by the system on several input queries for the CINDI library system.

4. Conclusion and future work

In the early stages of input parsing, the NLIDB system described above decomposes the user question and tries to match it to a semantic template. For the semantic parsing phase the system improves the quality of the parsing by using pre-computed information. The pre-computed information is based on the actual structure of the target database.

If the data is highly organized, as is the case in most relational database implementations, the search domain can be reduced, and thus the precision of the question analysis can be improved. The reduction of the search domain is made possible because the system knows in advance, through the interpretation rules, what it can answer.

Table 3
Experimental results

No.	Input sentence	Result	Note
1	Show the address of Mark Twain	Fail	Table was not identified. No word in the sentence matches any of the table names, nor their corresponding semantic sets.
2	Show the address of author Mark Twain	Pass	Table = author, template ⟨attribute-of-table⟩ **SELECT** address **FROM** author **WHERE** name = 'Mark Twain'
7	Show the name of the author of book 'General Astronomy'	Pass	table1. attribute = name, table1 = author, table2 = book, default attribute for table2 = title, template ⟨attribute-of-table-of-table⟩ **SELECT** author. name **FROM** author, resource, resource_author **WHERE** resource. title = 'General Astronomy' **AND** resource. resource_id = resource_author. resource_id **AND** author. author_id = resource_author. author_id
8	Who wrote 'General Astronomy'?	Fail	Table was not identified. No word in the sentence matches any of the table names, nor their corresponding semantic sets
11	What is the name of the author who wrote the book 'General Astronomy'?	Pass	table1. attribute = name, table1 = author, table2 = book, default attribute for table2 = title, template ⟨attribute-of-table-of-table⟩ **SELECT** author. name **FROM** author, resource, resource_author **WHERE** resource. title = 'General Astronomy' **AND** resource. resource_id = resource_author. resource_id **AND** author. author_id = resource_author. author_id
13	Show all books written by author Mark Twain	Pass	Table1 = book, table2 = author, template ⟨table-action-table⟩ **SELECT** book. name **FROM** author, resource, resource_author **WHERE** author. name = 'Mark Twain' **AND** resource. resource_id = resource_author. resource_id **AND** author. author_id = resource_author. author_id
13	Which are the books written by author Mark Twain?	Pass	Table1 = book, table2 = author, template ⟨table-action-table⟩ **SELECT** book. name **FROM** author, resource, resource_author **WHERE** author. name = 'Mark Twain' **AND** resource. resource_id = resource_author. resource_id **AND** author. author_id = resource_author. author_id

To evaluate the accuracy of the system, we need to ask real users to test the system and evaluate four measures:

- The number of correct SQL queries generated (e.g. Fig. 6).
- The number of incorrect SQL queries generated.
- The number of correct silences (e.g. Fig. 7).
- The number of incorrect silences.

In addition, for each type of error, we need to determine how many were brought about by:

- The results of WordNet (as the number of semantically related terms returned by WordNet increases, the accuracy of the overall system will decrease).
- The accuracy and number of parse trees generated by the Link Parser, and
- The precision of the interpretation rules used in our model.

In addition, we expect that the quality of the results will depend on:

- The size of the database: the larger the schema of the database, the lower the accuracy is expected to be.
- The specificity of the rules: The less specific the rules are, the lower the accuracy is expected to be (but the higher the portability of the system).
- The role of the pre-processor: The more specific rules and code is in the pre-processor, the higher the accuracy is expected to be (but the less portable the system will be).

Our system is not open-domain; it is designed to be tuned to a given database. Once so tuned, there is a dependency between the system and the database. However, this dependency is localized in the pre-processor.

Future work includes the formal evaluation of the system with real system users and the possibility to process queries with more than two sub-sentences. Also, to improve the results, we plan to associate a probability measure to the constructed SQL queries. This can be done by using a confidence measure in the lexical and semantic relations used to create the semantic sets (e.g. the number and type of links traversed in WordNet), and by using the cost vector (confidence level) of the Link Parser. Another goal is to test the system on different flavors of the Unix system (HP-UX, Sun Solaris, Linux) and on the Window system. We plan to port the system to other platforms as well.

NLIDB has been developed for the English language only and it integrates two other existing tools i.e. WordNet [12] and the context-free Link Parser [16]. The system uses the Link Parser in order to have a syntactic parse of the input sentence. Another parser or another type of grammar could also be used, as long as the resulting syntactic tree can be mapped to the semantic rules. For example the system could be further developed around a HPSG parser (Head-Driven Phrase Structure Grammar). In addition, because our approach uses WordNet and a syntactic parser, NLIDB is adaptable to other languages provided that such tools are also available in the new language. Its intended use is for the students in need of information from the library system. The system can be further developed to include dialog capabilities for narrowing down the result set.

Currently NLIDB does not support sub-queries and SQL extensions such as ORDERING, HAVING and COUNTING.

Although NLIDB is not open-domain, it is not restricted to one database. It can work with more than one database. When changing from one database to another, the only change occurs at the pre-processing time when the database schema is interpreted and the rules are created. NLIDB can work with any RDBMS (Relational Database Management System) given the table and attribute names have or can be related to linguistic meanings.

Acknowledgment

This research was supported in part by a grant from NSERC Canada and IDEAS.

References

[1] Lars Ahrenberg, Nils Dahlbäck, Annika Flycht-Eriksson, Arne Jönsson, Pernilla Qvarfordt, Lena Santamarta, Lena Strömbäck, Towards multimodal natural language interfaces for information systems the LINLIN approach, in: Proceedings of the 4th International Conference on Applications of Natural Language to Information Systems (NLDB 99), Austria, 17–19 June 1999.
[2] E. Brill, Transformation based error driven learning and natural language processing: A case study in part of speech tagging, Computational Linguistics 21 (4) (1995) 543–565.
[3] B.C. Desai, Supporting discovery in virtual libraries, Journal of the American Society of Information Science 48 (3) (1997) 190–204.
[4] B.C. Desai, S.S. Haddad, A. Ali, Automatic semantic header generator, in: Proceedings of ISMIS'2000, Charlotte, NC, Springer-Verlag, 2000, pp. 444–452.
[5] B.C. Desai, R. Shinghal, N. Shyan, Y. Zhou, Cindi: A system for cataloguing, searching, and annotating electronic documents in digital libraries, in: Proceedings of ISMIS'99, Warsaw, Poland, Springer-Verlag, June 1999, pp. 154–162.
[6] B.C. Desai, R. Shinghal, N. Shyan, Y. Zhou, Cindi: A system for cataloguing, searching, and annotating electronic documents in digital libraries, Library Trends 48 (1) (1999) 209–233.
[7] F. Dinenberg, D. Levin, Natural language interfaces for environmental data bases, in: Applications of Natural Language to Information Systems: Proceedings of the Second International Workshop, Amsterdam, The Netherlands, 26–28 June 1996, pp. 175–184.
[8] D. Jurafsky, J. Martin, Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics, Prentice Hall, New Jersey, 2000.
[9] C. Manning, H. Schutze, Foundations of Statistical Natural Language Processing, MIT Press, Cambridge, MA, 1999.
[10] Johannes Matiasek, Alexandra Klein, Harald Trost, Tamic-P: A system for NL access to social insurance databases, in: Proceedings of the 4th International Conference on Applications of Natural Language to Information Systems (NLDB 99), Austria, 17–19 June 1999.
[11] M. McTear, The Articulate Computer, Basil Blackwell Oxford, Oxford, England, 1987.
[12] G. Miller, WordNet: a lexical database for english, Communications of the ACM 38 (1) (1995) 39–41.
[13] D. Molla, J. Berri, M. Hess, A real world implementation of answer extraction, in: Proceedings of the 9th International Workshop on Database and Expert Systems, Workshop: Natural Language and Information Systems (NLIS-98), Vienna, 1998.
[14] Dan Moldovan, Sanda Harabagiu, Roxana Girju, Paul Morarescu, Finley Lacatusu, Adrian Novishi, Adriana Badulescu, Orest Bolohan, LCC tools for question answering, in: E.M. Voorhees, Lori P. Buckland (Eds.), Proceedings of the Eleventh Text Retrieval Conference (TREC-2002)—NIST Special Publication: SP 500–251, Gaithersburg, Department of Commerce, National Institute of Standards and Technology, November 2002.
[15] L. Plamondon, L. Kosseim, QUANTUM: A function-based question answering system, in: Proceedings of the 15th Conference of the Canadian Society for Computational Studies of Intelligence (AI 2002), Calgary, Canada, 2002, pp. 281–292.
[16] D.D. Sleator, D. Temperley, Parsing English with a link grammar, in: Proceedings of the Third International Workshop on Parsing Technologies, 1993.
[17] E.M. Voorhees, The TREC-8 question answering track report, in: E.M. Voorhees, D.K. Harman (Eds.), Proceedings of the Eight Text Retrieval Conference (TREC-8)—NIST Special Publication: SP 500-246, Gaithersburg, Department of Commerce, National Institute of Standards and Technology, November 1999.

[18] E.M. Voorhees, Overview of the TREC-9 question answering track, in: E.M. Voorhees, D.K. Harman (Eds.), Proceedings of the Ninth Text Retrieval Conference (TREC-9)—NIST Special Publication: SP 500-249, Gaithersburg, Department of Commerce, National Institute of Standards and Technology, November 2000.
[19] E.M. Voorhees, Overview of the TREC 2001 question answering track, in: E.M. Voorhees, D.K. Harman (Eds.), Proceedings of the Tenth Text Retrieval Conference (TREC-2002)—NIST Special Publication: SP 500-250, Gaithersburg, Department of Commerce, National Institute of Standards and Technology, November 2001.
[20] E.M. Voorhees, Overview of the TREC 2002 question answering track, in: E.M. Voorhees, Lori P. Buckland (Eds.), Proceedings of the Eleventh Text Retrieval Conference (TREC-2002)—NIST Special Publication: SP 500-251, Gaithersburg, Department of Commerce, National Institute of Standards and Technology, November 2002.
[21] E.M. Voorhees, D.K. Harman (Eds.), Proceedings of the Tenth Text Retrieval Conference (TREC-2002)—NIST Special Publication: SP 500-250, Gaithersburg, Department of Commerce, National Institute of Standards and Technology, November 2001.
[22] M. Watson, NLBean(tm) version 4: a natural language interface to databases, Available from: <www.markwatson.com>, Site visited in May 2002.
[23] W. Woods, R. Kaplan, Lunar rocks in natural English: Explorations in natural language question answering: linguistic structures processing, Fundamental Studies in Computer Science 5 (1977) 521–569.

Niculae Stratica holds an engineering degree from the University of Bucharest and a MS in Computer Science from the Concordia University in Montreal. He is currently enrolled in the Ph.D. program at the same university. His research work covers topics in computerized cancer treatment, natural language processing and distributed computing.

Dr. Leila Kosseim is an Assistant Professor at the Department of Computer Science and Software Engineering at Concordia University, where she does research in Natural Language Processing. Prior to that, she was a researcher at the Université de Montréal and an NSERC post-doctoral fellow in the industry. She received her Ph.D. in 1995 from the Université de Montréal working on Natural Language Generation.

Dr. Bipin C. Desai is a professor in the Department of Computer Science and Software Engineering of Concordia University. His current interests include database systems, digital library, medical information systems and NLDB.