A NATURAL LANGUAGE PROCESSOR FOR QUERYING CINDI

NICULAE STRATICA

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTREAL, QUEBEC, CANADA

September 2002

Abstract

A Natural Language Processor for Querying Cindi

In this thesis we present our work in designing and implementing a Natural Language Processor for querying Cindi, the Concordia Virtual Library System. The Natural Language Processor, named NLPQC, semantically parses natural language questions and builds corresponding SQL queries for the database. This makes the NLPQC system a Natural Language interface to relational databases. Our contribution to the field of the Natural Language interfaces is done through the reuse of WordNet and of the Link Parser, which are two proven tools from the Open Source domain, and through the introduction of a pre-processor that generates rules and templates for the semantic parsing. The NLPQC system is designed to be platform independent and can accept any database schema.

Acknowledgments

There are many people, both personal friends and professionals from academia, to whom I owe this thesis. I would like to take this opportunity to show my appreciation to them.

First and foremost I would like to thank my supervisor, Prof. Dr. Desai for his guidance and for his encouragement throughout the many phases of this thesis. The various challenges during the elaboration of the work have been overcome with his constant support.

I am also grateful to Prof. Dr. Leila Kosseim for the support in the Natural Language Processing area. Without the help I received from Dr. Kosseim, based on the extensive knowledge of the domain and on previous projects in the Natural Language processing area, the present thesis would not have been as advanced.

I would like to express my sincere recognition to Concordia University, whose excellence in academic teaching has attracted some of the finest professors from academia. Concordia provided me with excellent training.

I hope that in the future I will find more opportunities to work with people that helped me earn the Master degree in Computer Science.

Contents

Acronyms

ADO	ActiveX Data Object
ASCII	American Standard Code for Information Interchange
ASF	Apache Software Foundation
ASP	Active Server Pages
ATN	Augmented Transition Network
CGI	Common Gateway Interface
COM	Common Object Model
DARPA	Defence Advanced Research Projects Agency
DBMS	Database Management System
DCL	SQL Data control language
DDL	SQL Data definition language
DML	SQL Data manipulation language
DB	Database
DBMS	Database Management System
ELF	English Language Front End System
FTP	File Transfer Protocol
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol (World Wide Web protocol)
IR	Information Retrieval
JAVA	A platform independent language developed by Sun Microsystems
JDBC	Java Database Connectivity
MFC	Microsoft Foundation Classes
ML	Machine Learning
MRR	Mean Reciprocal Rank
MSDEV	Microsoft Development Environment
MSVC	Microsoft Visual C compiler and linker
NIST	National Institute of Standards & Technology
NL	Natural Language
NLP	NL Processor or Processing
NLPQC	Natural Language Processor for Querying Cindi
ODBC	Open Database Connectivity
OLAP	Online Analytical Processing
OO	Object Oriented
QA	Question Answering

ODBC	Open Database Connectivity
RDBMS	Relational DBMS
SH	Semantic Header
SHDB	Semantic Header Database
SQL	Structured Query Language
SSGRR	Scuola Superiori G. Reiss Romoli in l'Aquila, Italy
TREC	Text Retrieval Conference
WSA	World Site Atlas
WWW	World Wide Web

List of figures

List of tables

1. Introduction

The scope of this thesis is to design and implement a Natural Language Processor named NLPQC, for querying a relational database. The system has been implemented for use with the Cindi library system [Desai, 1999] but it is designed to work with any database. NLPQC works on Windows and has been designed to run on Unix as well. Figure 1 shows the scope of the NLPQC system.

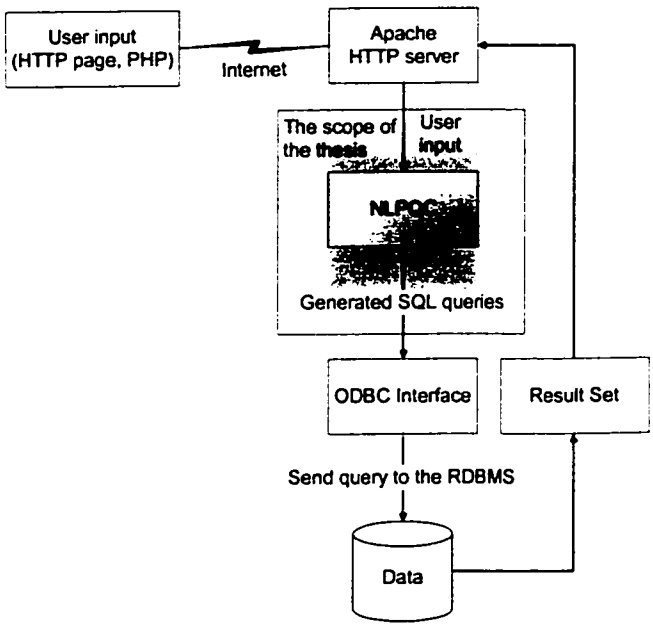

Figure 1. The scope of the NLPQC system

NLPQC is designed to eventually accept the user input in English language through a HTTP page and the ASF Apache[1] server. It generates a SQL query for a relational database engine. The database engine returns the result set to the user through the Apache server. The thesis focuses on parsing the user input and generating the SQL query. The integration with the database base and with the Apache server will be done in future work. Presently, we limit the work to accepting the query through a command line interface and generating the resulting SQL query.

[1] The Apache Software Foundation (ASF) is a non-profit corporation, incorporated in Delaware, USA, in June of 1999. The ASF is a natural outgrowth of The Apache Group, a group of individuals that was initially formed in 1995 to develop the Apache HTTP Server.

The system is composed of two parts: the pre-processor and the run time module (NLP). The pre-processor reads the schema of the database and uses WordNet [Miller, 1995] to generate a set of rules and SQL templates that are later used by NLP. By using WordNet, NLPQC takes advantage of an already proven English dictionary. The rules relate each table to its default attribute, and to other tables. For example, assume that in a database the table *author* has the default attribute *name* and it relates to the table *resource* through the action verb *writes*. The associated SQL template will be:

```
SELECT <attribute_list> FROM author,resource,resource_author
WHERE <condition_list>
AND author.author_id=resource_author.author_id
AND resource.resource_id=resource_author.resource_id
```

WordNet is used to create semantic sets for each table and attribute name in the schema. A semantic set is the set of all synonyms, hypernyms and hyponyms for a given word. The administrator can edit the semantic sets, the rules and the templates, if she is not satisfied with the ones generated by the NLPQC. The manual editing effort associated with this operation depends of the size of the database. The pre-processor is run only once in the beginning before any query is requested. If the system administrator decides to change the data base schema, the pre-processor must be run again. After the pre-processor has finished, the administrator rebuilds the NLP by including the rules and templates generated by the pre-processor.

The end-user runs NLP and types in an English sentence. For example the input might be *Who wrote the Old man and the sea?* NLP uses the Link Parser [Sleator, 1991] to do the syntactic parsing of the input. The Link Parser returns a set of parse trees and the associated cost evaluations[2]. NLP retains the lowest cost link, and uses the rules and the SQL templates generated by the pre-processors to do the semantic parsing. The system tries to identify for each word in the input sentence a corresponding table name. If the word is in the semantic set of a table, the table name is included in the table list. After the table list has been built, NLP tries to match the input with one of the three templates: **<attribute>of<table>**, **<attribute>of<table_1>of<table_2>** and **<table1><action verb><table_2>**. If the user input cannot be matched to one of the three templates, the system fails to produce a valid SQL query. Examples of valid inputs are:

Who *wrote*[3] *The old man and the sea?* (***<table1>*****<action verb>*****<table_2>***)[4]

---

[2] The link cost is a measure of the correctness of the parse tree. It is computed by the Link Parser as it will be shown in the section 3.12.

[3] The underlined word matches one element of the template

[4] The *italicized* template element does not occur in the input sentence. However NLPQC is able to retrieve the missing default table and attribute names.

What is the address of author Mark Twain? (**<attribute>of<table>**)

Show me all the books in the library. (***<attribute>*of<table>**)

Who borrowed the book Algorithms? (***<table1>*<action verb><table_2>**)

And here are examples that are not correctly processed by the NLPQC system:

What is the address of Mark Twain? (Missing attribute determiner, i.e. table *author*)

What books wrote author Mark Twain after 1870? (Missing template, no date processing)

How many books are in the library? (Missing template)

Starting back in the fifties, researchers from the field of Artificial Intelligence have tried to model the language processing capabilities of the human brain [McCarthy, 1959, 1960], [Herbert, 2002]. One of the goals of their work was to create a semantic representation of the sentence. A common technique to solve this task was to use predefined templates. If the template matched the sentence, a corresponding semantic frame was associated with it. Table 1 in section 2.1 shows two template-matching examples. This technique is simple and works well as a first approach. However, due to the many possible input sentences and to the inherent ambiguities of the NL to analyze any form of sentence from any domain, this implementation needs a very high number of templates. The NLPQC system works on a closed domain captured by the content of the database. The number of acceptable input sentences and their possible interpretations is drastically reduced by the limited answering capability of the database. This means that because of the small size of the database schema, the domain of expertise of the NLPQC system is limited to the exiting tables in the database. This limitation is established at the pre-processing time. This is why we chose to use templates as our main approach. Using templates has the advantage of being simple as compared to the more sophisticated approach of the semantic grammars[5]. In our system the semantic interpretation is done depending on the content of the database. By changing the database, the NLPQC could deal with a new discourse domain.

As we will see in Chapter 3, three templates have been developed through an iterative process. We wanted to increase the versatility of the system by increasing the number of templates and their complexity. However we soon noticed that the precision of the system went down rapidly. This was due to the increased risk of matching the user input with the wrong template. To reduce the risk, the templates have been greatly reduced in number and size. The conclusion was that there must be a low number of elementary templates that can be combined for greater flexibility. The present system uses three elementary templates, but it does not combine them yet. This issue should be addressed in the future.

[5] The semantic grammars are presented in the section 2.1.

If the input does not match any of the templates, the system returns a failure message to the user. If not, an SQL query is built. This does not mean that the SQL query is correct. For example, the system can build a syntactically correct query for the database, but the returned result set might not match user's expectation. For example, the user may ask *Show me the address of Mark Twain* and the system wrongly matches *author* with table *user* and build the apparently correct query:

```
SELECT address FROM user
WHERE user.name="Mark Twain"
```

The database engine accepts the query, but the query is not the SQL equivalent of the NL question. This kind of error is caused by an ambiguity in the table name resolution. To address this issue, the NLPQC does not rely on attributes to build the missing table name. In the example above, both *author* and *user* table have the attribute *address* and the system cannot decide which table to use.

The example above shows the following limitation of the system: the user always has to qualify the attribute with the table name. The correct input is *Show me the address of author Mark Twain.* For the time being it was decided that the system should return an error message to the user if the table names cannot be resolved without ambiguity.

NLPQC can produce SQL queries that involve one, two or three tables. It cannot resolve date and place information, and it cannot use more than two attributes per table. This limitation should be addressed in the future by introducing more elementary templates and combinations of them.

The NLPQC design is intended to work with any database schema. The present implementation was done for Cindi, the Concordia automated library system, which is presented in section 1.1.

1.1 The Cindi library system

The Concordia Digital Library System named Cindi [Desai, 1997; Desai, Shinghal, Shyan, Zhou, 1999] has been implemented through the use of Semantic Headers [Desai, Haddad, Ali, 2000]. Semantic Headers are used to facilitate the bibliographic search. The Semantic Header stores useful information, such as author's name and title of the bibliographic references stored in the database. As can be seen in Figure 2, behind the Semantic Headers, there is an expert system, which guides the user in the tasks at hand. The system helps the user ask meaningful questions.

After the user input is processed, the Cindi system uses the Semantic Headers database to retrieve the information from the resource catalog. The focus of the thesis is to design and implement a Natural Language interface for the Cindi system.

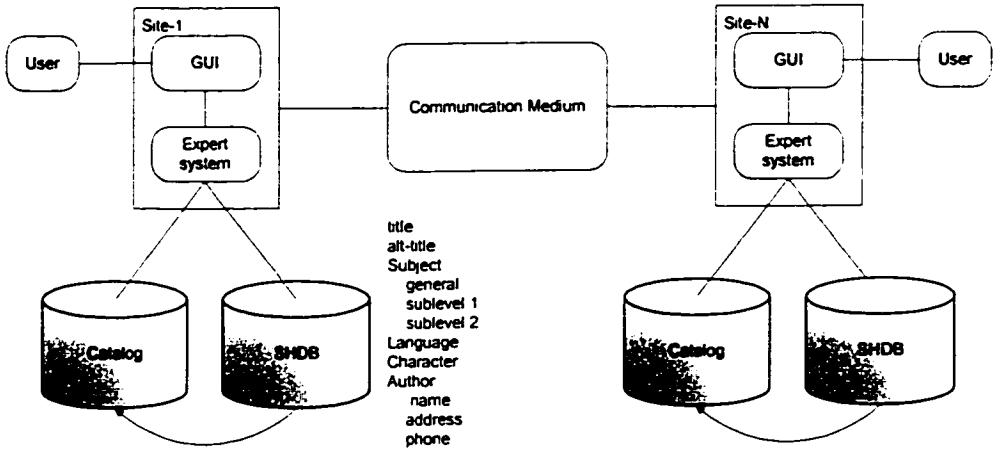

Figure 2. The architecture of the Cindi system

The system will eventually complement the existing Expert System for Cindi. The NLPQC accepts the user queries formulated in English language, and builds the corresponding SQL query for the database. The architecture of the Cindi system with the Natural Language interface is shown in Figure 3.

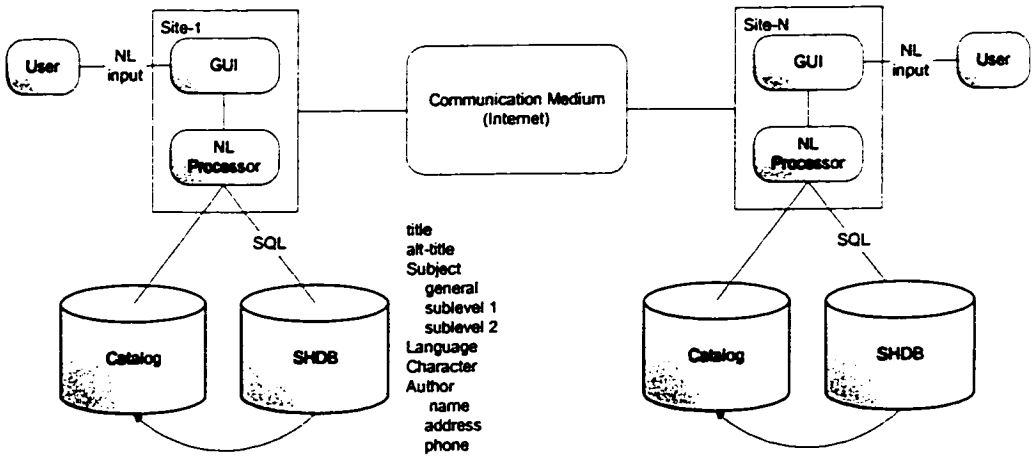

Figure 3. The Cindi system with the NLPQC interface

The NLPQC accepts the user input in the form of a question, such as *Who is the author of the book The Old Man and the Sea?* The system parses the question semantically and builds the following SQL query for the SHDB:

```
SELECT author.name FROM author, resource, resource_author
WHERE resource.title='The Old Man and the Sea'
AND author.author_id=resource_author.author_id
AND resource.resource_id= resource_author.resource_id
```

The *author*, *resource* and *resource_author* are three tables in the SHDB for Cindi. The example shown above uses the tables described in Appendix C.

The NLPQC also accepts requests, such as *Show the books written by Mark Twain.* The corresponding SQL query is:

```
SELECT resource.title FROM author,resource,resource_author
WHERE author_name='Mark Twain'
AND author.author_id=resource_author.author_id
AND resource.resource_id=resource_author.resource_id
```

The SQL could be sent to the database engine, and the returned result set is displayed for the user.

1.2 The foundation of the NLPQC system

The NLPQC is built on top of WordNet [Miller, 1995] and the Link Parser [Sleator, 1991], [Grinberg, 1995] which are two proven tools from the natural language area. The development of the NLPQC system is based on the following requirements:

1. The NLPQC is composed of two modules: a **pre-processor** and a run time **processor**. Figure 4 shows the overall architecture of the system.
2. The NLPQC **pre-processor** reads the schema of the database, and uses WordNet to create semantic sets for each table and attribute name. Example: for the table *resource* from the database schema, the pre-processor uses WordNet to find all synonyms, hypernyms and hyponyms. Then it builds the semantic set for *resource* (*book, volume, record, script).* The **pre-processor** also creates the rules and the SQL templates that can be edited by the system administrator. The schema is described in Appendix C.
3. The NLPQC run-time **processor** is integrated with the Link Parser to do the syntactic parsing of the input. The **processor** accepts English sentences related to Cindi and uses the rules created by the **pre-processor** to analyze the input and generate the SQL query. Here are three examples of accepted inputs: *List all books written by author Mark Twain, What is the phone number of author Leiserson?, Who is the author of the book General Astronomy?.*

4. The NLPQC system is written in C/C++ for the Windows platform. However, the system has been designed to run on Unix as well once the Unix version of the components used is available.
5. The NLPQC system is designed to accept any relational database schema.

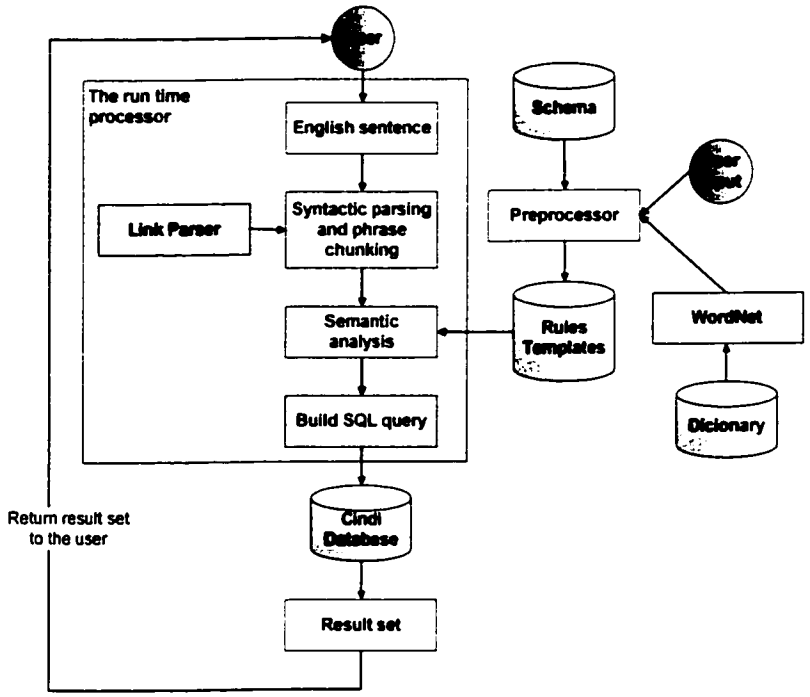

Figure 4. The architecture of the NLPQC system

The requirements implementation is described in detail in Chapter 3. Requirement 4 is true for the NLPQC code only, because the authors of the Link Parser have yet to release the source code for the Unix platform.

1.3 A parallel between QA systems and NL interfaces

The NLPQC system is a Natural Language interface to databases. Some of the tools and ideas it uses have been borrowed from Question Answering systems. In a more general context, Natural Language interfaces and Question Answering systems share common techniques for the semantic parsing.

As shown in Figure 5 both QA systems and NL interfaces do some kind of syntactic & semantic analysis of the English sentence, by using a common set of tools. QA systems build a query for the document collection and then they select the best answer from the result set. NL interfaces build the SQL query for the database, and the result of this query is then sent to the user. A NL interface currently works on a relational database but it does not yet have a mechanism for selecting the most probable answer from the

result set, as in a QA system. This is so because the selection is done when the SQL query is built. The SQL query is sent to the database engine, and the returned result set is considered to contain only valid answers of the SQL query which is considered to be correctly built.

Due to their similarities, a literature survey has been done in both Question Answering (QA) and Natural Language (NL) interface fields. Our literature review in the Natural Language Interfaces field is presented in Chapter 2. The most relevant systems have been analyzed. As a result of the survey, the most useful techniques and tools have been selected and incorporated into the NLPQC system. During this step, we ensured that the present work has been done by using and integrating the results obtained by other contributors in the field. A new bottom-up approach has been developed in order to increase the precision of the semantic parsing of the user input. This is our contribution to the NL semantic parsing domain, and it is described in Chapter 3. This approach has been presented in July 2002 at the SSGRR conference in l'Aquila, Italia [Stratica, Kosseim, Desai, 2002].

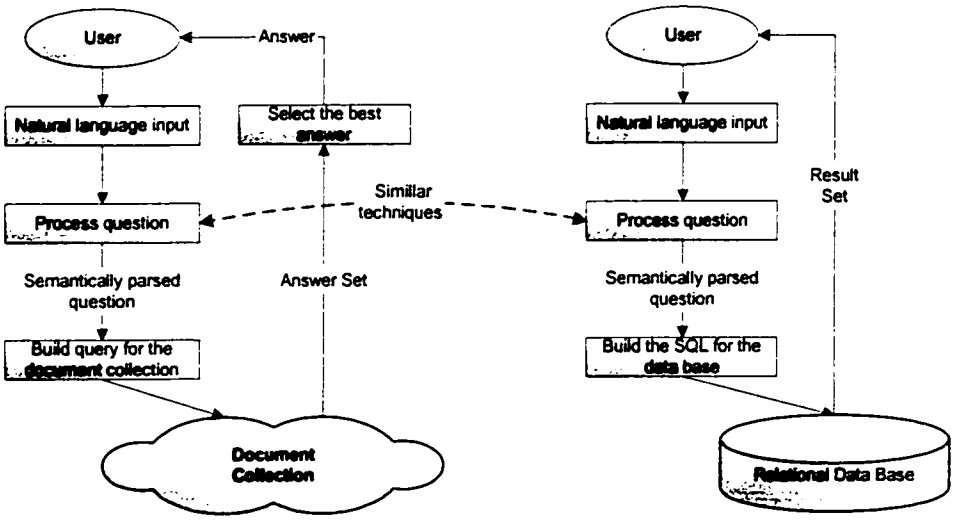

Figure 5. Comparison between QA systems and NL interface architectures

Chapter 4 shows the implementation details of the NLPQC system. Some of results are listed in Chapter 5 and the conclusions are given in Chapter 6.

2. Previous work in Natural Language processing

Question Answering and Natural Language interfaces share many techniques used for sentence parsing, such as part-of-speech tagging, phrase chunking and the use of templates for semantic interpretation. However, QA focuses on finding answers from general knowledge stored in a collection of documents, while NL interfaces are specialized and restricted to the particular database system they access. Our system, tries to reduce this limitation by first pre-processing the target database to adapt to a new discourse domain. In order to acquire this kind of flexibility and implement the most efficient techniques for sentence parsing, several QA systems and NL interface architectures have been analyzed and compared.

The evolution of the NL interface systems and the most relevant academic ideas are presented in the section 2.1. In the remaining sections, several working systems have been analyzed and the solutions most adapted to our project have been used in the architecture of the NLPQC system.

2.1 Literature review

The early efforts in the NL interfaces area started back in fifties [McCarthy, 1959]. Prototype systems had appeared in the late sixties and early seventies. Many of the systems relied on pattern matching to directly map the input to the database [Abrahams, 1966]. Table 1 shows two pattern-matching examples from *Formal LIst Processor* or *FLIP*, an early language for pattern-matching on LISP structures [Teitelman, 1966]. If the input matches one of the patterns, the system is able to build a query for the database.

Table 1. Pattern-matching examples

User input	Matching Pattern
What is the capital of Spain?	"capital"...<country>
What is the capital of each country?	"capital"..."country"

The system implementation and the database details were inter-mixed into the code. Because of that the coverage of the pattern-matching systems was limited to the specific database they were coded for and to the number and complexity of the patterns.

Later systems introduced syntax-based techniques. The user input is parsed and then it is mapped to a query for the database. The best-known NL interface of that period is *Lunar* [Woods, 1972], a natural language interface to a database containing chemical analyses of moon rocks. Lunar, like the other early natural language interfaces, was built for a specific database, and thus it could not be easily modified to be used

with different databases. The system used an ATN syntax parser coupled with a rule-driven semantic interpretation procedure to translate the user input into a database query. Here is an example processed by the *Lunar* system which shows the user input and the resulted query for the database:

```
LUNAR (Woods 1971)
Natural language system based on ATNs.

Request: (DO ANY SAMPLES HAVE GREATER THAN 13 PERCENT ALUMINUM)
Parsed into query language:
     (TEST (FOR SOME X1 / (SEQ SAMPLES) : T ; (CONTAIN X1
                  (NPR* X2 / 'AL203) (GREATERTHAN 13 PCT))))
Response: YES
```

In contrast to the *Lunar* system, NLPQC uses a syntax-based tree parser (Link Parser) and a semantic interpreter based on rules and templates.

The next evolutionary step has been taken by the semantic grammar systems. By the late seventies several more NL interfaces had appeared [Hendrix, 1977], [Codd, Limbie, 1974]. The user input is parsed and then mapped to the database query. This time though the grammar categories do not have to correspond to syntactic concepts. The semantic information about the knowledge based is hard-coded in the semantic grammar, and thus the systems based on this approach are strongly coupled with the database they have been developed for. Gary Hendrix et al. [Hendrix, 1977] presented a NL interface to large databases named *LADDER* that is based on a three-layered architecture and is representative of the NL interfaces built in the seventies. The three layers operate in series to convert natural language queries into calls to DBMS's to remote sites. *LADDER* can be used with large databases, and it can be configured to interface to different underlying database management systems. *LADDER* uses semantic grammars, a technique that interleaves syntactic and semantic processing. The question answering is done by parsing the input and mapping the parse tree to a database query. The grammar's categories, i.e. the non-leaf nodes that will appear in the parse tree do not necessarily correspond to syntactic concepts. Figure 6 shows an example of a semantic parse tree.

LADDER does spelling correction and can process incomplete inputs. The first layer is the Natural Language system. It accepts questions in a restricted subset of Natural Language and produces a query to the database. For example the question *What is the length of the Mississippi?* is translated into the query *((NAM EQ MISSISSIPPI)(?LENGTH))* where *LENGTH* is the name of the length field, *NAM* is the name of the river field, and *MISSISSIPPI* is the value of the NAME field. The other two layers are not relevant to us because they deal with database details. The system has been implemented in the LISP programming language and can process a database which is the equivalent of a relational database with 14 tables and 100

attributes. At that time the database systems were not relational. The Natural Language layer uses a fragmented approach. This means that although the system cannot process any input, it accepts major fragments of language pertinent to particular application area.

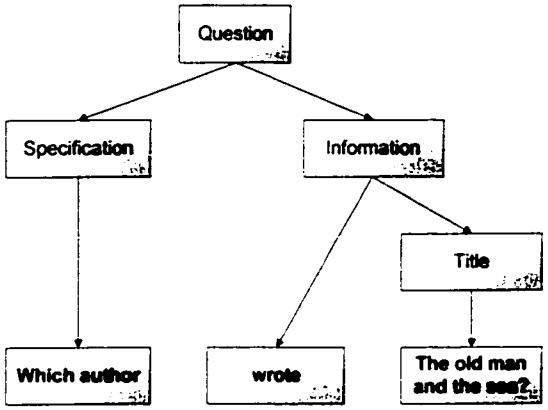

Figure 6. Example of a parse tree in a semantic grammar.

This approach is extremely important because it uses the basic idea of *divide and conquer* to a then new domain. The authors acknowledge that the open-domain Natural Language processing is too complex for computers and decided to *specialize* the application to a specific domain. This strategy goes in line with the limited answering capability of a database. Our own system adopted a similar strategy. However, NLPQC addresses the limitation issue through the database independent implementation.

LADDER uses Augmented Transitions Networks (ATN) and templates to do the parsing of the user input. For example *WHAT IS THE <ATTRIBUTTE> OF <TABLE>* is a simple template that is a natural language representation of the <table, attribute> pair. The system also uses more general patterns such as *<NOUN-PHRASE><VERB-PHRASE>*. Our system uses templates too, for the reasons explained at the beginning of Chapter 1. *LADDER* groups together all the attributes related to the same table. This idea will be reused by NLPQC in the future, to allow the creation of complex SQL queries.

Another concept introduced by *LADDER* is the *generalization*. The system tries to match the input words to general classes, in order to take into account the semantics. For example, if the database contains a table named *resource*, and the user types the words *book* and *article*, the system maps *book* and *article* to the table *resource*, through *generalization*. NLPQC implements a similar generalization mechanism through the semantic sets, as shown in Chapter 3.

LADDER is limited to a finite set of sentences and it does not deal with syntactic ambiguities. In the example *NAME THE SHIPS FROM AMERICAN HOME PORTS THAT ARE WITHIN 500 MILES OF NORFOLK* the system favors shallow parsing and associates distance with *SHIPS*. Although *LADDER* was

revolutionary in the ways presented above, it lacked portability. Like many other system at that time, it inter-mixed data structure and procedures thus making the system dependent on the specific database that it was implemented for.

Other systems used semantic grammars as well. *CHAT-80* [Warren, 1982] is one of the reference systems of the eighties and was the basis for systems in other languages, such as *PRATT-89* [Teigen, 1988]. *CHAT-80* transforms English questions in Prolog expressions, which are evaluated against a Prolog database. *CHAT-80* was developed for a geographical database and could answer questions like these:

```
Does America contain New_York?

Does Mexico border the United_States?

Is the population of China greater than 200 million?

Does the population of China exceed 1000 million?
```

Figure 7 shows the hierarchy of geographical concepts used by the *CHAT-80* to do the semantic interpretation of the user input and to map it to the database fields.

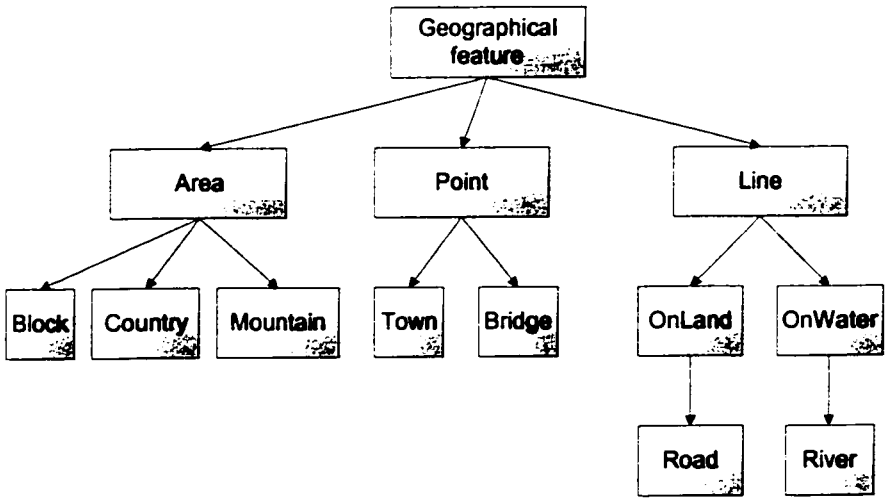

Figure 7. The geographical concept hierarchy

A major inconvenience of the system was that the implementation was heavily related to the specific database application. The next generation of NL interfaces was based on an intermediate representation language. The natural language question is transformed into a logical representation, which is independent of the database structure. The logical representation is then transformed into a query for the database. Grosz

[Grosz et al., 1983] implemented the *TEAM* system as a transportable Natural Language interface to databases. Grosz's vision was ahead of her time. Grosz realized that the Natural Interface must be isolated from the implementation details of the database. Today this idea might look like as an obvious choice, due to the existence of the SQL, but at that time it was not. *TEAM* was innovative also because it introduced concepts that are today known under the generic name of *object-oriented approach*. NLPQC, which also has an object-oriented architecture, inherited from *TEAM* the idea of database independence. This has been implemented through the introduction of the NLPQC pre-processor, which can accept any schema.

At the time *TEAM* was implemented, there was no standard data description language or DDL to describe the database schema. Because of that, the authors had to talk to the database management personnel to obtain the information required for the adaptation of the system to a new database.

TEAM is based on a three-domain approach. It separates information about language, about the domain and about the database. The system uses a lexicon of words, much like the semantic sets produced by NLPQC. *TEAM* stores information about possible relations between words. These would be the equivalent to the pre-defined NLPQC templates. *TEAM* translates the user input into a logical form, based on syntactic parsing, semantic parsing and basic pragmatic rules. The rules are domain and database independent.

Another novelty introduced by *TEAM* is the concept of logical schema. The system takes the user input and builds the logical schema. This is composed of generic classes, such as *student*. If the user types in *sophomore*, the logical schema will refer to the word *student*, which is more general than *sophomore*. This approach has been adopted by NLPQC too, by using hyponyms of WordNet..

The *TEAM* system can distinguish between three different kinds of fields: arithmetic, boolean and symbolic. They are used in matching the logical schema to the database schema. The system is limited to a small range of databases and accepts questions having transitive verbs only.

Although some of the NL interface systems developed in the eighties showed impressive characteristics in specialized areas, they did not gain wide acceptance. NL interfaces were regarded more like experimental systems due to the fact that some other options were offered to the database end users, such as graphical and form-based interfaces. Another negative factor in the development of the NL interfaces was the limited linguistic coverage. The early systems faced numerous problems, such as modifier ambiguity resolution (all *cars* made by *GM* near *Montreal*), quantifier scooping (*all*, *one*, *few*, *many*), conjunctions-disjunctions (*and*, *or*), noun composition (*research book*, *sport facility*). The NL systems developed in the nineties tried to overcome some of these issues. The *JANUS* system [Hinrichs, 1990] has an interface to multiple underlying systems and it is one of the few systems to support temporal questions. *JANUS* addresses the limited database capability by allowing the user to access more than one database.

As we saw in the examples presented above, the NL interfaces went through an evolutionary process. As we will see later on in this chapter, today's systems use various techniques and integrated tools to do the syntactic and the semantic parsing of the English query. Some of these techniques are also used in the QA domain.

Our system is based on a series of QA systems and NL interfaces that are presented in the following sections. NLPQC adopted the intermediate representation approach through the use of semantic sets, schema rules and semantic templates. Our system uses syntactic parsing techniques that have been borrowed from the QA field.

QA systems use various techniques to interpret the user input, build a query for the document collection and extract the most probable answers from a set of candidate answers. There are two classes of QA systems: linguistic and statistic. The linguistic approach consists of syntactically and semantically parsing of the user input in order to obtain a query for the knowledge base. A linguistic system focuses on the semantic of the sentence.

A statistic system processes the input text in order to determine a list of phrases that occur with reasonable frequency in the collection of documents forming the knowledge base.

A typical QA system uses a collection of documents from which it extracts the set of answers. Then it chooses the most probable one, based on the implementation of specific algorithms.

A NL interface uses a database as knowledge base. The NL interfaces use some of the parsing techniques used by the QA systems, but they are more limited due to the nature of the database knowledge as compared to the document set for QA. On one hand this restriction allows for little flexibility in interpreting the user input, but on the other hand, there is an advantage to it. The NL interfaces can successfully use templates for the semantic parsing, as explained in the beginning of the Chapter 1.

Another difference between NL interfaces and QA systems is that the NL interfaces do not select the most probable good answer from the answer set. This is due to the fact that the goal of a typical NL interface is to build a good SQL query for the database engine. Once the query is successfully built, the system's job is over.

The NLPQC system is a linguistic NL interface to the databases. However, its architecture is based on ideas borrowed from both QA linguistic and statistic systems, and from some NL interfaces, such as:

- Code reuse (WordNet[6], LinkParser[7])
- Tool integration (Webclopedia[8], Prager[9], Falcon[10])
- Use of templates (START[11], QUANTUM[12], QA-LaSIE[13])
- Use of pre-processed knowledge (ELF[14])
- The generation of the SQL query (SQ-HAL[15], NLBean[16])
- Use of relationships based on verbs (English Query[17])
- Use of table rules (EasyAsk[18])

Table 2 shows the main attributes of the systems mentioned above that have been considered for the NLPQC architecture. They are not compared against each other. Instead, the main features that inspired the NLPQC system are highlighted.

Table 2. Some QA and NL systems used in the NLPQC architecture

System	Type	Semantic parsing	Information Retrieval	Ideas borrowed by NLPQC
START	QA linguistic	Use of templates <subject relation subject>	Through annotation from other sites	NLPQC uses templates <attribute of table> <attribute of table of table> <table verb table> The use of templates is more appropriate for the NL interface, as shown in Chapter 1.
QUANTUM	QA linguistic	Use of templates <question focus discriminatnt>	Uses the Okapi system [Robertson, 1998]	NLPQC uses templates <attribute of table> <attribute of table of table> <table verb table>
Webclopedia	QA linguistic ML	Uses CONTEX parser for the syntactic parsing	Through complex queries	Use Link Parser to do the syntactic parsing instead of CONTEX. The NLPQC queries are less complex and work on fewer tables.
QA-LaSIE	QA linguistic	Uses Eric Brill's tagger [Brill,	Uses the IR system	NLPQC uses a name matcher to

[6] Appendix A
[7] Appendix B
[8] Section 2.2.3
[9] Section 2.2.5
[10] Section 2.2.6
[11] Section 2.2.1
[12] Section 2.2.2
[13] Section 2.2.4
[14] Section 2.3.2
[15] Section 2.3.1
[16] Section 2.3.4
[17] Section 2.3.3
[18] Section 2.3.5

		1995], a name matcher and a discourse interpreter	from the University of Massachusets.	link the words in the sentence with table names.
Prager	QA statistic	No parsing	Passage-based search with help from WordNet	NLPQC does not do any statistical operation, but uses WordNet to do name matching[19].
Falcon	QA linguistic	Semantic parsing with help from WordNet	It stores results for later re-use	NLPQC uses WordNet and will store questions and SQL queries in the future.
SQ-HAL	NL linguistic	Poor syntactic and semantic parsing, but it is open source. It uses name matching.	Produces SQL queries for the database	NLPQC uses name matching with help from WordNet, and thus gives better semantic coverage.
ELF	NL linguistic (?)[20]	Apparently uses a pre-processor for the database schema.	Produces SQL queries for the database	NLPQC uses a pre-processor. Unfortunately, there is no independent evaluation of ELF on the WWW, and it cannot be compared against other systems, NLPQC included.
NLBean	NL linguistic	Poor syntactic and semantic parsing, but it is open source. It offers a simple mechanism to produce SQL queries. Can be deployed over the WWW.	Produces SQL queries for the database	The NLPQC prototype has been developed on top of the NLBean. The prototype tested the SQL generation techniques and had only limited parsing capabilities.
EasyAsk	NL linguistic (?)	It has its own parser and spell check capability, uses WordNet. It has a two-way dialog with the user to correct the ambiguous words. Uses table rules extracted from the schema.	Produces SQL queries for the database	NLPQC uses table rules as well. It does not do the spell check, but it allows the system administrator to correct the pre-processor errors. EasyAsk goes a step further and allows the user to correct the ambiguous input words.

After analyzing numerous QA systems and NL interfaces, it appears that a good NL interface must work on both ends: the user input and the schema of the database. The user input is semantically parsed. The resulting parse tree must then be matched with table and attribute names from the schema.

SQ-HAL and NLBean try to match the words from the input sentence to the table and attribute names. If they are not meaningful English words, the match fails. Prager and the authors of Falcon improved this phase by using WordNet. NLPQC also uses WordNet to build the semantic set of the schema elements, i.e. table and attribute names. The ELF system claims a very high success rate, which is probably due to the fact that it prepares in advance the information related to the schema, and it uses it at run time to do the semantic parsing. On the other hand, ELF and other commercial systems do not publish details about the architecture and design used, and thus cannot be fully evaluated. In the NL interfaces domain there is a

[19] The system matches the input tokens to the semantic sets of the table and attributes names.
[20] These are supposedly linguistic systems. The design details are not known for the commercial systems.

need for standard test sets such as in the TREC conferences for QA [TREC-9, 2000], [TREC-X, 2001]. Until then, the only way to understand and build on top of the existing commercial systems is to use them intensively and compare them against academic systems such as SQ-HAL and NLBean.

In contrast to the QA domain, we have noticed that the NL interfaces are poorly represented on the WWW. This is due to the fact they have very practical and immediate application in the WWW commerce, and thus the creators are not willing to publish the architecture of their systems. Instead, they publish test results, often compared against other systems. Because of this situation, we found few open source interfaces, and retained only two of them: SQ-HAL and NLBean. They might not be as advanced as ELF, English Query or EasyAsk, but at least one gets the source code and can start to think to do something better. As a matter of fact, the NLPQC system has been built on this idea.

Another valuable idea is the use of templates. QUANTUM, START and other systems use templates for the semantic parsing. As already mentioned in Chapter 1, the best results with templates are obtained by the NL interfaces. Despite this, the SQ-HAL and NLBean systems do not use templates for the semantic parsing. However, NLBean uses SQL templates to build the SQL query, and it gives better results than SQ-HAL. NLPQC uses templates for the semantic parsing and for the SQL query generation, thus it takes the best ideas for the reviewed systems.

One feature that is under-represented in the QA and NL interface domains is the learning mechanism. Falcon addresses this problem through storing the user input and the resulted query for later reuse. Webclopedia goes even further down the road and implements a learning mechanism at the semantic parsing level, through dynamic template generation. NLPQC does not have learning nor memorizing mechanisms. The issue should be addressed in future work to implement a Falcon-like approach.

We have noted that the flexibility and the precision of a system are contradictory requirements. If the number of templates for the semantic parsing increases, the precision goes down due to failure in the ambiguity resolution. If the ambiguity is resolved through the use of a limited number of elementary templates, the system coverage is greatly reduced, and many input sentences will be dropped. From our experience it resulted that there is a need for a minimal number of elementary templates to reduce the ambiguities. The elementary templates must be further combined for more flexibility. This process is similar to dividing a complex sentence into elementary phrases and then using the elementary templates on them. This step has not been implemented in the NLPQC and it is part of future plans. For the time being, the system uses three templates. This number has been found through trial and error in an effort to get acceptable precision and good coverage. However, the system can deal with a relatively small schema of up to 20 tables and 20 attributes per table and cannot build complex queries involving more than 3 tables and two attributes per table. These limitations should be re-worked in the future.

In the following sections we present the systems that have been studied and used in the development of the NLPQC architecture.

2.2 Question Answering systems

In the following sections, we present the systems that inspired us in the development of the NLPQC system. The focus is on the architecture and the implementation of each system.

2.2.1 The START system

The START server [Katz, 1997] is a QA system that answers questions in English about the MIT AI Laboratory, geography and other topics.

Figure 8. An actual screen output from the START system

The Server is based on the START natural language system developed by Boris Katz. START has been available to World Wide Web users since December 1993. The system can answer simple questions such as *Show me a map of Quebec.* START finds the answer on the World Sites Atlas site [WSA, 2002] and returns it to the user as shown in Figure 8. The architecture of the START system is shown in Figure 9.

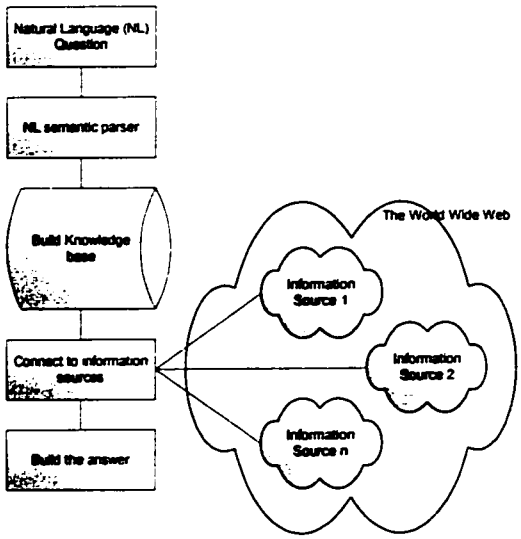

Figure 9. The Architecture of the START system

The Natural Language sentence is semantically parsed. Based on the result of the parsing, the system decides where to look for the answer.

The START system uses T-expressions for the semantic parsing, where T stands for Template. The system uses the pattern **<subject relation object>**. This approach has the advantage of being intuitive but has a limitation. Sentences with different syntax and close meaning are not considered similar by the system. For example, the phrase 'The student surprised the teacher with his answer' is parsed into <student surprises teacher> while the phrase 'The student's answer surprised the teacher' is parsed into <answer surprises teacher>, which is a different interpretation of the input sentence.

The NLPQC system uses 2 and 3 element templates. The NLPQC pattern **<table$_1$ action table$_2$>** resembles the START's **<subject relation object>** pattern. This pattern is used in phrases such as 'List the books written by author Mark Twain' **Written** is the action verb that relates **Book** and **Author** in the Cindi library system. One difference between START and NLPQC is that our system uses more than one pattern for the semantic parsing. The NLPQC templates are detailed in Chapter 3.

2.2.2 The QUANTUM System

The QUANTUM system [Plamondon, Kosseim, 2002] tries to find the answer to a natural language question in a large collection of documents. QUANTUM relies on computational linguistics as well as information retrieval techniques. The system analyzes questions and then selects the appropriate answer extraction function. In the parsing phase, QUANTUM decomposes the question in three parts: a *question*

word, a *focus*, and a *discriminant*. The NLPQC system borrowed the idea of having the question split in two or three part templates. Table 3 shows examples of the question decomposition in the NLPQC system.

Table 3. Examples for the question template used by NLPQC

Input	Question word	Focus	Discriminant
Who is the author of the book 'The old man and the sea'?	Who is	author	of the book 'The old man and the sea'?
What is the phone number of author Ernest Hemingway?	What is	phone number	of author Ernest Hemingway?

The NLPQC system uses the three elementary templates to decompose the question, as shown in Table 2. For the first row, NLPQC uses the template **<word$_1$><action><word$_2$>** and the rule (author, write, book) to build the SQL query, as we will see in Chapter 3. The second row is resolved with the **<attribute>of<object>** template. In addition to handling questions, the NLPQC system handles imperative sentences. Table 4 shows examples of such sentences that are parsed through the action template **<word1><action><word2>**.

Table 4. Examples for the action template used by NLPQC

Input	Word1	Action	Word2
Show all books written by author Mark Twain	books	written	author
List all annotations written by user Thomas Edge	annotations	written	user

The NLPQ system uses question and action templates to do the semantic parsing. It uses schema rules to relate the focus and the discriminants to tables and attributes. The rules are built by the NLPQC pre-processor and used at run time, as described in Chapter 3.

2.2.3 The Webclopedia factoid system and WordNet

The Webclopedia factoid QA system [Hovy E. et al., 2001], makes use of syntactic and semantic world knowledge to improve the accuracy of its results. The system processes the question with the CONTEX parser [Hermjakob, Mooney, 1997], [Hermjakob, 2000]. The idea of reusing the already existing and proven tools, inspired by Webclopedia, has been incorporated in the NLPQC system architecture. Our system uses the Link Parser in the early stages of sentence processing.

Another tool used by Webclopedia for query expansion is WordNet. The system extracts words used in the term definitions before searching for definitions in the collection of documents. The NLPQC system also uses WordNet to expand the query and the schema of the database.

Another characteristic of the Webclopedia system that has been used in the NLPQC system is learning capability. Webclopedia uses the CONTEX parser to execute the syntactic-semantic analysis of the sentence. CONTEXT is made up of two modules, a grammar learner and a parser. The grammar learner uses machine-learning techniques to produce a grammar represented as parsing actions, from a set of training examples. We borrowed this idea for our system. The NLPQC gets trained by the pre-processor, which generates the rules and templates that are used at run time by the semantic parser.

Finally, the Webclopedia forms complex queries by combining simple queries through Boolean operators. The NLPQC system builds complex SQL queries by using Boolean operators, as it is described in Chapter 3.

2.2.4 The QA-LaSIE system

QA-LaSIE [Scott, Gaizauskas, 2001] finds answers to questions against large collections of documents. The system passes the question to an information retrieval system (IR) that uses it as a query to do passage retrieval from the text collection. The top ranked passages from the IR system are then passed to a modified information extraction system. Partial syntactic and semantic analysis of these passages, along with the question, is carried out to score potential answers in each of the retrieved passages. The system processes the question by using the following steps: question tokenizing, single and multi-word identifying, sentence boundaries finding, tagging, phrase parsing and name matching. QA-LaSIE uses Eric Brill's tagger to assign one of the 48 Penn TreeBank part-of-speech tags to each token in the text [Brill, 1995].

The NLPQC system reuses the idea of integrating the existing tools in the new developments. While QA-LaSIE builds on top of Eric Brill's tagger, our system integrates the Link Parser for the syntax parsing.

2.2.5 Use of WordNet Hypernyms for Answering What-Is Questions

Based on the fact that 26% of the questions are of type 'What is...?' in the TREC-9 competition [TREC-9, 2000], John Prager [Prager et al, 2001] implemented a passage-based search process that looks for both the question term and any of its hypernyms found with WordNet. The lookup algorithm counts the number of occurrences of the question term with each of its WordNet hypernyms in the document collection and

divides this number by the number of "is-a" links between the two. The terms with the best score are selected as the answer. With this strategy, Prager claims a precision of 0.83 on the TREC-9 corpus.

The basic idea that NLPQC system borrowed from Prager's work, is the use of WordNet as a knowledge base in the generalization process. In other words, our system uses WordNet to find semantically related terms that are connected to the schema of the database. For example, as shown in Figure 15, *book* and *volume* are synonyms from the perspective of the Cindi database.

2.2.6 The Falcon system

Falcon [Harabagiu, 2001] is an answer engine that semantically analyses a question and builds the expected answer type. Then it uses an information retrieval system and does the answer identification. Falcon also uses WordNet to help create the expected answer patterns. The system provides a feedback mechanism for query reformulation if the question processing fails, which increases the precision of the system. It also has a mechanism to restore the previously formulated question for faster response time. This is based on storing the questions and their associated queries in a database for further use.

As mentioned before, the NLPQC system also uses WordNet to build the semantic sets of the words which are related to the database schema.

2.3 NL Interfaces to Databases

After the semantic parsing of the input sentence, a NL interface builds the SQL query that is then passed to the RDBMS engine. The RDBMS engine returns the answer set to the user. As opposed to a QA system, a typical NL interface has no mechanism of discriminating among the entries in the answer set. The precision of the system relies exclusively on the correctness of the SQL query. In the ideal case, the NL interface returns one result set, or none if the answer is not found in the database. This problem has been addressed in the NLPQC system, in the case when the zero result set is due to the limited knowledge stored in the database. The system administrator has the possibility of changing the current database, and switch the system from one domain to another.

In the following sections, we describe several NL interfaces, both academic and commercial. These systems have inspired us in the architecture and design of the NLPQC system.

2.3.1 The SQ-HAL system

SQ-HAL [Ruwanpura, 2000] is a NL interface that translates questions into SQL queries. It executes the SQL statements to retrieve data from the database and displays the answers to the user. The system is designed to be database and platform independent with multi-user support.

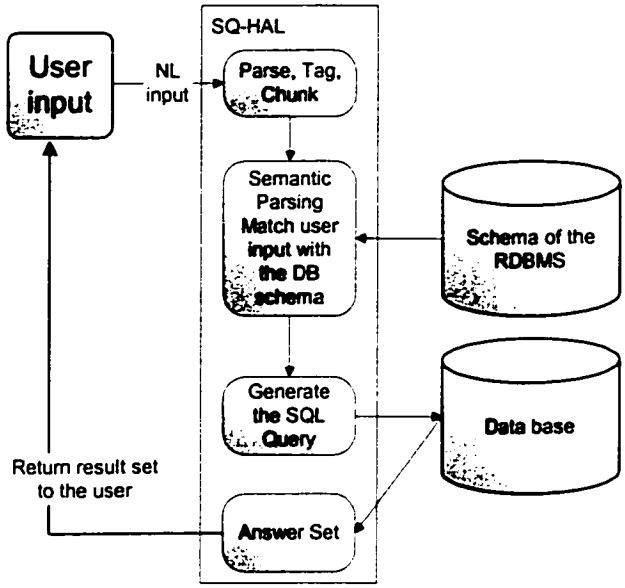

Figure 10. The architecture of the SQ-HAL system

Because the input is in NL, users with no knowledge of SQL are able to address the database through SQ-HAL. The system has the ability to learn a new grammar. The architecture of SQ-HAL is presented in Figure 10. The design can be broken down into three sections: database functionality, natural language translation and user interface. Currently SQ-HAL can translate natural language queries into simple, single and two tables joined SQL statements with or without a simple condition. These statements are then executed in the selected database to retrieve information and display it to the user, thus simplifying the data retrieval process. Also the user has the choice of modifying these SQL statements or creating its own.

SQ-HAL has some limitations and problems. The database table names and column names have to be valid English words. Multiple word names, such as `telephoneNo`, are treated as single words and do not produce the expected results. This is because the keyword `telephoneNo` is not in the English dictionary and it is unlikely that the user will use it as input. SQ-HAL cannot determine synonyms for table names and column names and therefore the user has to manually enter these words in the dictionary. The NLPQC system presented in this thesis solves this problem by using WordNet to create semantic sets of the related terms and allows for the user to edit them. Our system accepts any table and attribute names, regardless

whether or not they are in the English dictionary. SQ-HAL is not capable of determining the relationships between tables, and user assistance is expected to solve this problem.

The NLPQC system uses a mechanism for generating the SQL query, which is similar to the one the SQ-HAL system uses. Our system has a pre-defined library of SQL patterns that contain generic table and attribute names. The names are filled in at run time, based on the rules generated by the pre-processor. After identifying the tables and the attributes that are used in the SQL query, the NLPQC system uses the remainder of the sentence as the value of the default attribute of the last table in the table list. For example, in the question *What is the address of the author Mark Twain?* NLPQC identifies *address* as being an attribute of the table *author*. The default attribute of the table *author* is *name*. The system associates the remainder of the sentence, i.e. *Mark Twain* to the *author.name*. The SQL query is:

```
SELECT address FROM author
WHERE name='Mark Twain'
```

2.3.2 The English Language Front system

The ELF system [Elf, 2002] is the closest to our own approach. Like many other commercial system, the ELF system claims a rather good performance, although how that performance is achieved is not disseminated. The system reads the schema of the database and then creates a set of rules that are used during semantic parsing, when the natural language input is converted into SQL query for the relational database system. The ELF company offers a comparative study between ELF, English Wizard from Linguistic Technology and English Query from Microsoft. English Wizard is described in section 2.3.5

The test, whose results are presented in Figure 11, was performed by the ELF company using the Northwind database sample included with SQL Server 7.0 [Microsoft Corporation, 2000]. The standard eight tables were selected, plus the *Order Subtotals* query. The questions were selected randomly from the ELF regression test suite. Automatic builds were used for the ELF sample and the English Wizard sample; for English Query, the pre-built sample interface that Microsoft ships with English Query was used. The results shown in Figure 11 [ELF, 2002] illustrate the claimed superiority of the ELF natural language database query system over its rivals. A '+' mark indicates a correct response. This condensed view of the results shows the error messages generated by the system, or a comment explaining why the SQL generated was wrong.

Total results: 16 correct answers for English Wizard, 17 correct answers for English Query [Microsoft, 1998] and 91 correct answers for ELF. If the 7 queries that were answered correctly by all 3 systems are

discounted, then on the remaining questions, English Wizard scored 0.096, English Query scored 0.107, and ELF scored 0.903.

The system has not been tested against other sets of questions, partially because in the NL interfaces area there is no standard requirements, such as the QA systems have in the TREC conferences [TREC-8, 2000]. The owner company has conducted the test, thus it cannot be considered reliable or objective. However, we present this system here because of its revolutionary approach of using a pre-processing mechanism for the schema of the database.

ELF vs. English Query vs. English Wizard

where are the suppliers from Germany located	+	Sorry, I didn't understand that	+
show the names and complete address of the biscuit companies	+	Sorry, I didn't understand that Please check your spelling or phrasing If you capitalize proper names, it will be easier for me to understand you	INCORRECT show employees, Discontinued 126 times (crosses with Order Details)
at which company does Ian work	INCORRECT work=>worker=>employee (crosses with Employee table)	Based on the information I've been given about this database, I can't answer "At which companies Ian does works?" I haven't been given any information on companies	INCORRECT No rows returned
who handles the specialty items	+	INCORRECT No appropriate choice	I'm not familiar with the word handles
show the domestic suppliers	+	Based on the information I've been given about this database, I can't answer "How domestic are suppliers?" I haven't been given any information on domesticness	I'm not familiar with the word domestic
show the New Orleans suppliers	+	INCORRECT No answer because New Orleans is part of name, not whole name of company	+
show the New England suppliers	+	INCORRECT Same problem as New Orleans	I'm not familiar with the words New England
which company handles the specialty products	+	INCORRECT No appropriate choice	I'm not familiar with the word handles

Figure 11. Comparing three NL interfaces

Figure 12 [ELF, 2002] shows an actual screen shot from the ELF system. Our NLPQC system has many similarities with ELF. NLPQC uses a similar approach, in a sense that it pre-processes the schema of the database and it generates a set of rules and templates that are used at run time for the semantic parsing. However, the authors of ELF did not publish the details of their research work, and therefore we cannot compare the two architectures.

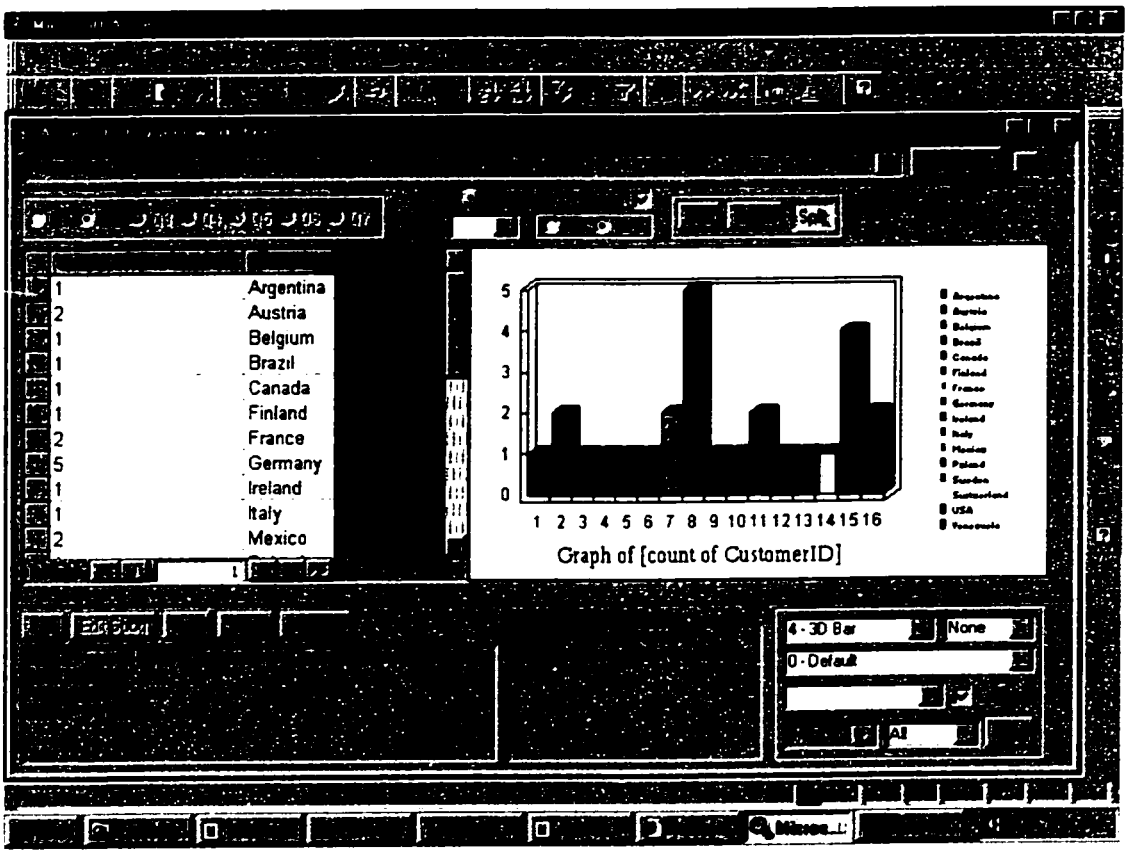

Figure 12. An actual screen output from the ELF system.

2.3.3 The English Query for SQL 7.x

The English Query [Microsoft Corporation, 1998, 2000] was designed and implemented by Microsoft Corporation as part of the SQL server. It consists of two components: a run-time engine to convert English questions to SQL and an authorizing tool for the initial setup of the database.

Before proceeding with English questions, the database administrator uses the authorizing tool to initialize the database structure, create entities, define relationships between entities and create verb phrasing for relationships. All these steps have been implemented in the NLPQC system. Once the database is setup, English Query can translate English questions to SQL with the capability of searching multiple tables and multiple fields. However, we cannot compare the two architectures, due to the lack of information on English Query's design. The following example demonstrates one of the SQL statements translated by English Query. The generated SQL query references three different table and five attributes. The question "What hotels in Hawaii have scuba?" is translated into this:

```
SELECT DISTINCT dbo.HotelProperty.HotelName AS "Hotel Name",
dbo.HotelProperty.USReservationPhone AS "Phone",
dbo.HotelProperty.StreetAddress1 AS "Street Address",
dbo.HotelProperty.CityName AS "City", dbo.HotelProperty.StateRegionName
AS "State or Region" from dbo.HotelProperty, AmenityNames, Amenities
WHERE dbo.HotelProperty.StateRegionName='Hawaii'
AND AmenityNames.Amenity='Scuba'
AND AmenityNames.AmenityID=Amenities.AmenityID
AND dbo.HotelProperty.HotelID=Amenities.Hotelid
```

The run-time engine of the English Query can be integrated with Common Object Model modules, supporting environment such as Visual C++, Visual Basic and Active Server Pages. This enables the English Query to be embedded in custom built software as well as for ASP supported web sites.

English Query however is available only for Win32 platforms and supports only data sources that have OLE Database services such as Oracle and Microsoft Access. Our system is now available on Windows and it will run on Unix as soon as the authors of the Link Parser tool release the source code.

2.3.4 The NLBean System

The NLBean System was developed by Mark Watson [Watson, 1997]. His system is an example of how far an individual can get into the complex area of the NL interfaces to the database systems.

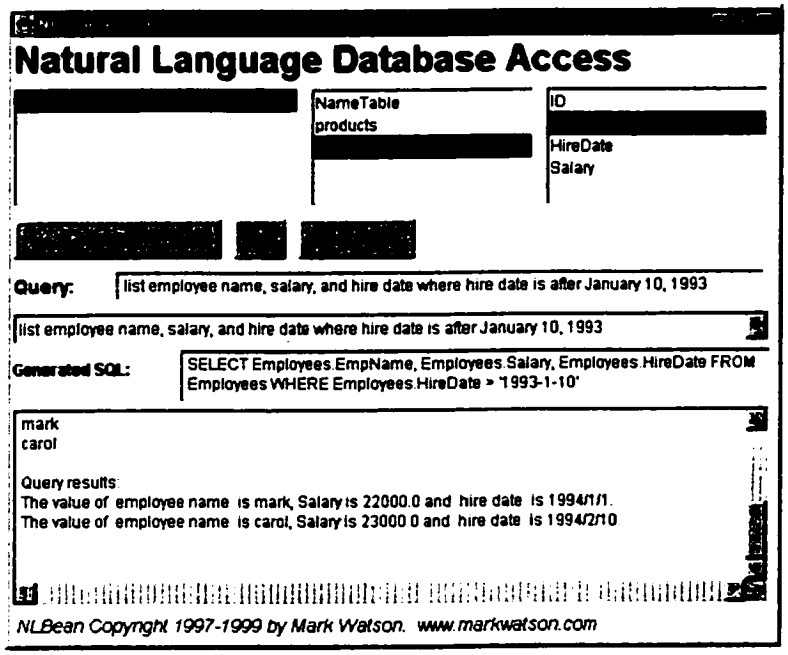

Figure 13. An actual screen output from the NLBean system

The architecture of NLBean is based on a set of objects, which manage the database and the natural language input. The system does the semantic parsing of the sentence and builds an SQL query for the database. Figure 13 shows an actual screen output from the NLBean system, showing the English query, the computed SQL query and associated result set. The test has been done on the Windows platform with Sun JDK 1.3.

The NLBean system is important for the present thesis. The early prototype of the NLPQC system has been built on top of the NLBean library and used the SQL generating mechanism of NLBean. The system parses the user input syntactically and then it tries to match the words to table and attribute names in the database. Then it uses predefined SQL templates to build the actual SQL query for the database.

2.3.5 The EasyAsk System

Developed by EasyAsk Inc. [Dix, 2002], the EasyAsk system is an application with a Natural Language user interface. Dr. Larry Harris initially developed it in 1995, when it was called English Wizard. Users can input their queries as keywords, phrases or complete English questions. The user input is then processed before being translated to SQL. EasyAsk has its own dictionary and thesaurus, which helps to correct spelling mistakes in the user input and to match synonyms. The ambiguous words are then clarified by asking the user for the expected meaning. The EasyAsk generates relatively complex SQL queries. It can generate sub-queries with HAVING clauses and complex ratios. It can match exact values or other SQL specific condition options such as LIKE, EXIST, NOT EXISTS clauses and NULL. It recognizes table join logic by looking at database structure. Some of the other SQL related features include generating sub-select queries (query within another query), handling of date/time yes/no questions and mapping abbreviations into proper values. Once the generated SQL statement is executed, the output of the database results is presented to the user in a number of different forms, such as spreadsheets, charts and graphs. For example: if user input is "Pie chart of sales by region" the output data is represented as a pie chart. The EasyAsk supports various data sources including ODBC databases and Microsoft SQL. The application runs only on the Win32 platform. EasyAsk uses tools to crawl the database looking at structure and nomenclature, and builds a custom dictionary that spells out how the objects are categorized and what attributes are used. The NLPQ system uses a similar methodology of searching the schema and building the table rules based on it. The dictionary is loaded on the Database server and used to translate the language of queries into the language of the database. For example the term 'ladies' is translated to 'women's', 'pleated' to 'pleat', 'cordaroy' to 'corduroy' and 'slacks' to 'pants'. The next step is called triangulation, associating query components with categories (women's, pants) and attributes (corduroy). Anything the system doesn't recognize will be dealt with in a text search. This means that the system could not find the generalizing category of the word, and thus it uses the word in the search, instead of the category. The company says its software can make the site searches accurate up to 9 out of 10. The system learns and improves its

performance by itself. The learning mechanism inspired us in the design of the NLPQC pre-processor. The pre-processor 'teaches' the run time module what to do with the user input and how to relate it to the predefined SQL templates. This mechanism is described in Chapter 3. Our NLPQC system also borrowed the idea of database independence from EasyAsk. The system accepts any database, like EasyAsk does. One major difference between NLPQC and EasyAsk is that EasyAsk does not pre-process the schema and does the semantic parsing based on elements acquired exclusively at run time. Also, the NLPQC has a more limited capability in the generation of the QL query than EasyAsk has. Our system lacks the HAVING, LIKE, EXIST queries. The mechanism for producing these queries has not been implemented yet. However this issue will be addressed in the future work by introducing more templates and by improving the schema rules.

2.3.6 Conclusions

The QA systems and NL interfaces presented in this chapter use various techniques to address the semantic parsing issues. The NLPQC system architecture is based on ideas inspired by these systems. Most QA systems use Information Retrieval (IR) modules to get the answer to a given question, and thus the precision of QA is directly related to the quality of the IR system.

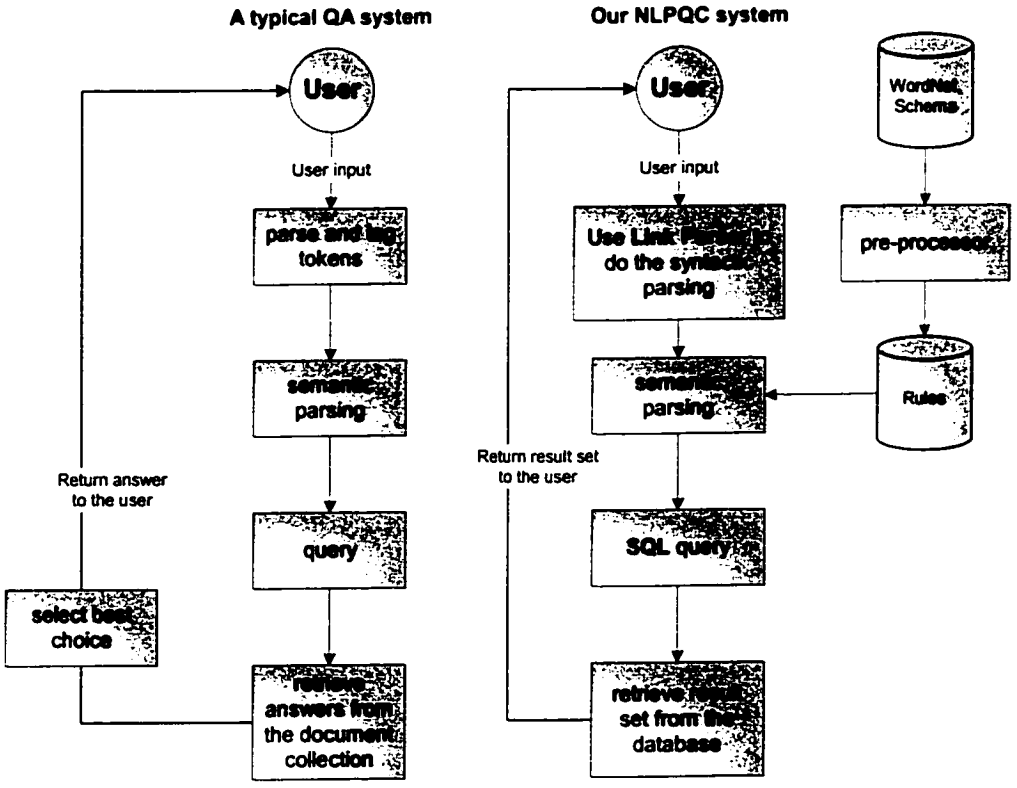

Figure 14. A comparison between QA and NLPQC

Figure 14 shows a comparison between the architecture of a typical QA and our NLPQC system. NLPQC uses a pre-processor, similarly to the ELF system, and to English Query. NL Bean has been used to make the prototype, and SQ-HAL has been our inspiration for generating the SQL queries based on predefined templates. The syntactic parsing tools have been inspired by the previous work in the QA and NL interface fields. The START system uses three-element templates of type **<subject relation object>** for the semantic parsing. The NLPQC system uses two and three element templates. The Quantum system uses a 3-objects mechanism to divide the user question into a question word, a focus and a discriminant. The same technique is used by the NLPQC in the semantic parsing of the sentence. Our system also uses a different 3-objects mechanism for the parsing of the action sentence. The details are presented in Chapter 3.

The Webclopedia system uses WordNet and CONTEX in order to do the semantic parsing of the user input. The NLPQC is also using WordNet.

The QA-LaSIE system finds answers in an incremental fashion. The system passes the user input to an Information Retrieval system that extracts passages from a text collection. The result is sent back the QA-LaSIE system that further processes the results. The NLPQC system uses a similar strategy, in a sense that it relies on the Link Parser, in order to do the token parsing and tagging of the user input.

John Prager used WordNet in order to establish links between words. He built a hierarchy in which one can find the 'best ancestor' of a given word. Our system uses WordNet to retrieve the synonyms, hypernyms and hyponyms of a given word.

The Falcon system uses WordNet and a learning mechanism. The NLPQC uses WordNet for creating semantic sets of related words.

At the end of the research work we were able to build the architecture of the NLPQC system, which is characterized by these features:

- The flexibility of semantic parsing is increased by the introduction of a **pre-processor**. The pre-processor reads the schema of the database and produces a set of rules and templates that are used at run time for the semantic parsing. The system administrator can edit the rules and the SQL templates to correct or otherwise modify them. We consider this approach as a novelty because to our knowledge other authors in previous academic projects did not use it. There is at least one commercial system that apparently does pre-process the schema, but the details are not reported [ELF, 2002].

- The NLPQC system is designed to be database **independent**. This feature is implemented by the use of the pre-processor. To adapt it to a new database, the system administrator only has to pass the new schema through the preprocessor.
- The user does not have to know **SQL** to access the database. The system hides the SQL details from the user.
- The NLPQC system **integrates** proven tools from the NL processing area, such as WordNet and the Link Parser. Instead of implementing a dictionary and a syntactic parser, NLPQC uses proven tools from the NL domain. The code reuse is part of the object-oriented nature of the NLPQC system.
- The NLPQC can run on **Windows** and **Unix** platforms. However, at this date there is a limitation due to the fact that the authors of the Link Parser have yet to release the source code for the Unix platform.

3. The architecture of the NLPQC system

The NLPQC system accepts natural language sentences from the user, parses them semantically and builds an SQL query for the Cindi database. The core functionality of the system is based on **rules** and **templates**. The system administrator can modify or refine the rules.

In the following sections we describe the rationale behind the NLPQC system architecture and the implementation process. Our project overlaps with several domains of expertise, such as relational databases, SQL query language, natural language processing techniques and software engineering. The database knowledge is necessary to understand and manipulate the relations and the attributes. SQL is used by NLPQC to build the formal queries for the database. The implementation is based on an object-oriented architecture and design. As for the code reuse, the NLPQC integrates two already proven resources: WordNet and the Link Parser. Overall, the NLPQC system is a continuation of the research work done by many individuals and institutions in the NL interfaces field, and we hope that our system brings a small, positive contribution to it.

3.1 The NLPQC challenges and how they are addressed

The two major challenges of the NLPQC system are the execution of semantic parsing and the production of the SQL query. Both have a direct impact on the precision of the system. NLPQC addresses the semantic parsing through the use of the rules and the templates that are generated by the pre-processor. The rules are based on the schema of the database, on WordNet and on the system administrator feedback. The system administrator can edit, add and modify the rules. This phase it is shown on the right side of Figure 4 in Chapter 1. At the run time, the NLPQC uses the rules and tries to match the input words with table and attribute names from the database schema. The rules describe the relations between the tables, and the relations between a table and its attributes. In the following sections we assume that the table and attribute names in the schema are meaningful, and they can be found in the English dictionary. If this is not the case, the system administrator has the possibility to specify synonyms. For example, if the table name is *res*, the system administrator adds the following synonyms: *resource, book,* and *volume*. Every time the word *book* occurs in the input sentence, it is associated with the table *res* from the database.

3.2 The Semantic sets

The NLPQC pre-processor uses WordNet [21] to relate words semantically, as shown in Figure 15. Each word is assigned a family of synonyms, hypernyms and hyponyms, which form the semantic set of the word.

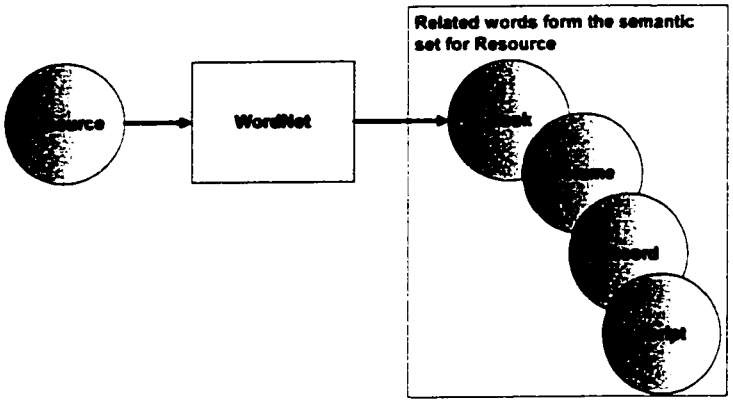

Figure 15. Generating semantically related words with WordNet

The semantic set for *resource* maps *book* to the table *resource* in the database schema. As a matter of fact all of the related words shown in Figure 15 map to the table *resource*. The system administrator may use the words *book*, *volume*, *record* or *script* interchangeably as they will all be associated with the table *resource* by the run time part of the NLPQC system. Here is the actual code generated by the NLPQC pre-processor, that implements the semantic set for the table *resource*:

```
STRING table0[] = {
        "resource","resources","resourcefulness","imagination","book","#",
        "title","statute","championship","conveyance","claim","entitle","#",
        "resource_id","#",
        "version","variant","adaptation","translation","interlingual","#",
        "source","beginning","origin","root","seed","channel","informant","#",
        "size","sizing","of","it","#",
        "abstract","abstraction","outline","synopsis","precis","#",
        "contributor_id","#"};
```

[21] Appendix A gives more details on WordNet

3.3 Rules related to the database schema

The architecture of the NLPQC system was built incrementally having as starting point the schema of the database. Figure 16 shows the most important table names and their relations in the Cindi database.

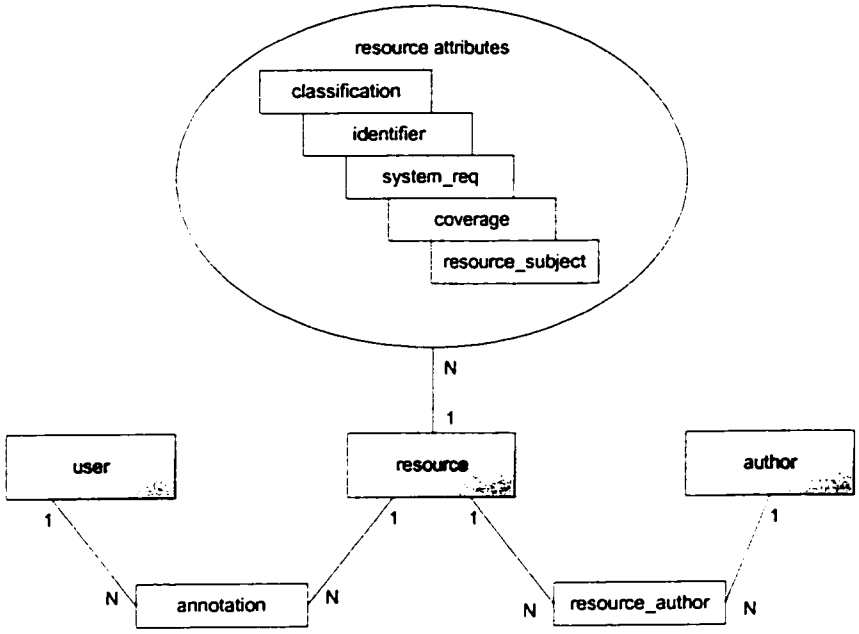

Figure 16. The table representation of the Cindi schema

The table *resource* has a one-to-many relation with the following tables: *classification*, *identifier*, *system_req*, *coverage* and *resource_author*. In other words, one resource can have more than one classification, identifier and so on. Tables *resource* and *user* are in a many-to-many relation, through the use of table *annotation*. One *user* can write more than one *annotation*, and one *resource* can be related to more than one *annotation* as well. Tables *resource* and *author* are in a many-to-many relation, through the use of table *resource_author*. One *author* can write more than one *resource*, and one *resource* can be related to more than one *author*.

An *author* writes a *resource*. A *user* writes an *annotation* of a *resource*. These two relations involve three tables each. The action verb in both cases is the verb *write*, as shown in Figure 17.

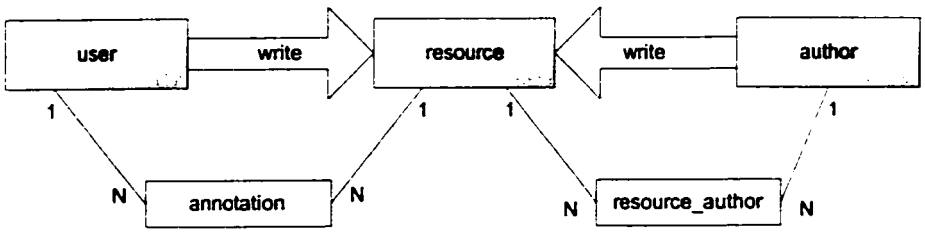

Figure 17. Action verbs and many-to-many relations

The schema of the database gets translated into the rules shown in Table 5. These rules are produced by the NLPQC pre-processor, and they are based on the primary and foreign keys in each relation. The table list is computed after the attribute list and the condition list have been determined.

Example:

```
if (table author and table resource are used) then
      (related table resource_author is used too) and
      (SQL query template includes
            resource_author.resource_id=resource.resource_id
            AND resource_author.author_id=author.author_id)
```

Table 5. Rules related to the database schema

Table list	**Related table**	**Related attribute equations in the SQL query**
author	none	none
resource	none	none
author,resource	resource_author	resource_author.resource_id=resource.resource_id **AND** resource_author.author_id=author.author_id
user	none	None
user, resource	annotation	annotation.resource_id=resource.resource_id **AND** annotation.user_id=user.user_id
resource, classification	none	classification.resource_id=resource.resource_id
resource, identifier	none	identifier.resource_id=resource.resource_id
resource, system_req	none	system_req.resource_id=resource.resource_id
resource, coverage	none	coverage.resource_id=resource.resource_id
resource, resource_author	none	resource_author.resource_id=resource.resource_id

In the code, the rules are stored in this format:

```
STRING cross_reference[] =
        {"author,resource","resource_author","AND
        resource_author.resource_id=resource.resource_id AND
        resource_author.author_id=author.author_id",
        "resource,author","resource_author","AND
        resource_author.resource_id=resource.resource_id AND
        resource_author.author_id=author.author_id",
        };
```

Table 6 shows the default attributes for each of the tables in the Cindi database. The system administrator must produce these values. The default attribute is used when the system is not able to identify the attribute of a table. This is the case in the question *Who is the author of the book Algorithms?* NLPQC identifies the words *author* and *book* as being related to the tables *author* and *resource*, but it does not identify their attributes. The SQL query is built with the default attributes:

```
SELECT author.name FROM author,resource, resource_author
WHERE resource.title='Algorithms'
AND resource_author.resource_id=resource.resource_id
AND resource_author.author_id=author.author_id
```

Table 6. The default attributes

Table name	Default attribute
resource	*title*
author	*name*
user	*name*

In order for the system to understand various formulations of the same query, the pre-processor uses WordNet to build a semantic set for each table and attribute name. The semantic set is formed of synonyms, hypernyms and hyponyms of the table or attribute name. The system administrator can edit the semantic set. Table 7 shows the set for the tables *resource* and *author*.

Table 7. The semantic set for tables *resource* and *author*

Semantic set name	Elements
resource	resource, book, volume, record, script
resource.title	title, name, rubric, caption, legend

resource.language	language, speech, words, source language
resource.keyword	keyword, key word
resource.created_date	publishing date, creation date
resource.size	size, number of pages
author	author, writer, communicator, generator, source, maker, shaper
author.name	name, first name, last name, full name
author.organization	organization, institute, institution, administration
author.address	address
author.phone	phone, phone number

Another set of schema rules are those which relate tables. For example, given that *author*, *resource* and *resource_author* are three tables from the database schema, when tables *author* and *resource* tables are involved in the query, then the cross reference table *resource_author* must be involved as well, as shown in Figure 18. The SQL template associated with the three tables is also shown.

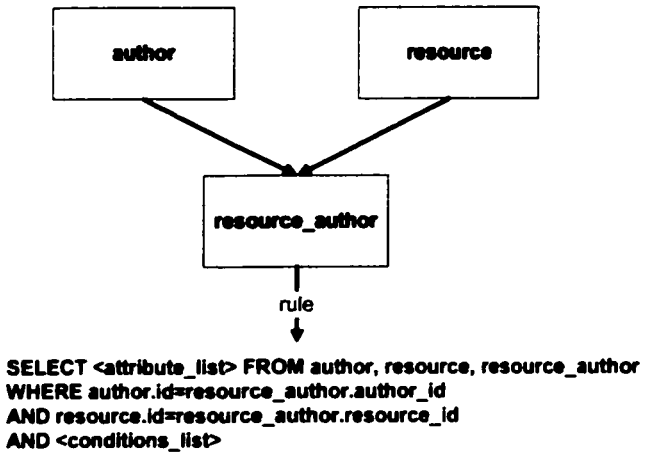

Figure 18. The schema rules

Here is the actual code that stores the table rules. The first element is the list of tables that occur in the SQL query. The second element is the third table that should be involved. The third element shows the SQL template for the partial condition list.

```
STRING cross_reference[] =
{
        "author,resource","resource_author",
        "AND resource_author.resource_id=resource.resource_id AND
        resource_author.author_id=author.author_id",
};
```

3.4 Rules related to the action verbs

Figure 19 shows two words that are related through an action verb. An *author writes* a *book* and thus the verb *writes* establishes a relation between the two words. Some relations are reflexive, such as in the example *The author walks*. The pre-processor uses the template **<word1><action_verb><word2>** and produces the rule **(author, writes, book)**.

In the question *Who **wrote** The old man and the sea?* the NLPQC system identifies the action verb **wrote**, but fails to find any word related to the table names.

The system applies the rule **(author, writes, book)**, and it builds the table list *author,book* for the SQL query:

```
SELECT <attribute_list> FROM author,book
WHERE <condition list>
```

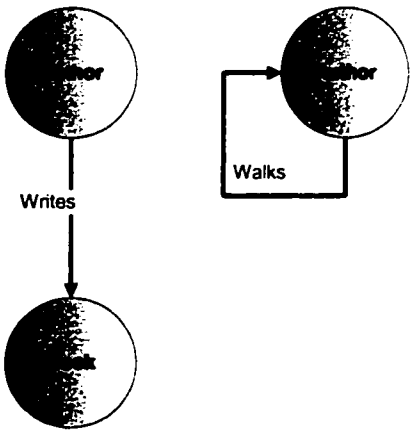

Figure 19. Action related tables

This is the actual code that implements the rules for the action verbs:

```
STRING action[] =   // action, object1, object2 (related objects)
{
      "written", "resource", "author",
      "wrote",   "author","resource"
};
```

The action verb *written* has been related to tables *resource* and *author*.

3.5 The rationale behind the NLPQC templates

The NLPQC system uses two and three-element templates, similar to those used by Plamondon and Kosseim [Plamondon, Kosseim, 2002] in the QUANTUM system. Table 8 shows two examples:

Table 8. Two NLPQC templates: <attribute-of-table> and < attribute-of-table-of-table>

	<attribute-of-table>	**Modifier(s)**	**SQL**
List	*address* of *author*	Mark Twain.	SELECT address FROM author WHERE author.name='Mark Twain';
	<attribute-of-table-of-table>	**Modifier(s)**	**SQL**
What is	the *name* of the *author* of the *book*	Algorithms?	SELECT name FROM author,resource,resource_author WHERE resource.title='Algorithms' AND resource.resource_id=resource_author.resource_id AND author.author_id=resource_author.author_id;

The word *author* is associated with the table *author* and the word *book* is associated with the table *resource*.

The templates are based on simple patterns. The system first tries to match the input on simple **<attribute>**-of-**<table>** and **<attribute>**-of-**<table>**-of-**<table>** patterns. If this fails, it goes to the next level of complexity and tries to parse the sentence through the action template of type **<table1><action><table2>**. The system is built in such a way that it can be continuously improved, by adding more and more complex templates. The templates are described in the following sections.

3.6 The <attribute>-of-<table> template

In the **<attribute>-of-<table>** template only one table is involved, along with one attribute and the value of the default attribute, as shown in Figure 20.

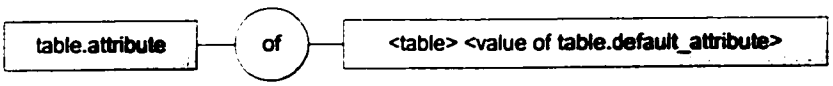

Figure 20. The <attribute>-of-<table> template

The matching SQL template is stored in the system configuration file:

```
SELECT attribute FROM table1
WHERE default_attribute=<value of table1.default_attribute>
```

Here are some examples of NL sentences:

- *Address of author Mark Twain* (table$_1$.attribute = Address, table$_1$= author, value of table$_1$.default_attribute = Mark Twain)
- *Email of author Charles Darwin* (table$_1$.attribute = Email, table$_1$= author, value of table$_1$.default_attribute = Charles Darwin)
- *Publishing date of book 'Evolution of species'* (table$_1$.attribute = Publishing date, table$_1$= book, value of table$_1$.default_attribute = Evolution of species)
- *Role of author Ernest Hemingway* (table$_1$= author, table$_1$.attribute = role, value of table$_1$.default_attribute = Ernest Hemingway)
- *Language of book 'General Astronomy'?* (table$_1$= book, table$_1$.attribute = language, value of table$_1$.default_attribute = General Astronomy)

To better understand how the templates and rules are used, here is an example. If the user types in this question *What is the address of the author Mark Twain?* the NLPQC calls the Link Parser to do the syntactic parsing. The Link Parser returns a parse tree containing the tokens, their part-of-speech tags and their relation with other tokens. The NLPQC scans the token list and identifies the table words. If there is only one, NLPQC tries to confirm the word representing an attribute of the table. In this example, the match is a success:

```
What is the address of the author Mark Twain?
```

In Table 7 *address* is in the semantic set *author.address* and the *author* is in the set *author*. Thus the table name is *author* and the attribute name is *address*. Following the template, the system finds out that *address* is an attribute of the table *author*, whose default attribute *name* has value 'Mark Twain'. We notice that if the semantic set of the attribute does not match the table semantic set, the template is not applicable. If two tables have the same attribute, the system discriminates between them by the table name.

From Table 6, the default attribute for the table *author* is *name*. The NLPQC builds the SQL query based on the associated SQL template:

```
SELECT address FROM author
WHERE name='Mark Twain'
```

3.7 The <attribute>-of-<table1>-of-<table2> template

In the **<attribute>-<of>-<table1>-<of>-<table2>** template, two tables are involved along with the value of the default attribute of the second table, as shown in Figure 21.

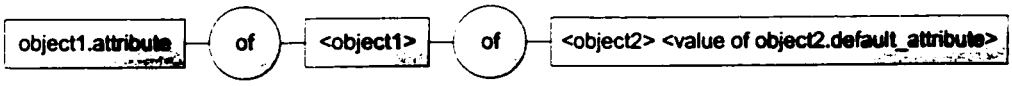

Figure 21. The attribute-of-table-of-table template

The matching SQL template is stored in the system configuration file:

```
SELECT table1.attribute FROM table1, table2
WHERE table2.default_attribute=<value of table2.default_attribute>
```

For example:

- *Name* **of** *author* **of** *book 'Computer architecture'* (table$_1$.attribute = name, table$_1$= author,table$_2$= book, value of table$_2$.default_attribute = Computer architecture)
- *Name* **of** *author* **of** *book 'General Astronomy'* (table$_1$= author, table2 = book, value of table$_2$.default_attribute = General Astronomy)

Example: *What is the name of the author of the book General Astronomy?*

The NLPQC checks the token list that is returned by the Link Parser, and identifies the prepositions of type '**of**'. If there are two of them, NLPQC tries to match the sentence with the **<attribute>-<of>-<table1>-<of>-<table2>** template. In this example, the match is a success:

```
What is the name of the author of the book 'General Astronomy'?
```

The first attribute is *name*, and the first table is *author*. The second table is *book*, and the value of its default attribute is 'General Astronomy'. From Table 7 the system finds out that *name* is related to *author.name*, and *book* is related to *resource*. From table Table 6 the default attribute for *resource* is *title*. From the associated SQL template, the system builds the SQL query:

```
SELECT author.name FROM author,resources
WHERE resource.title='General Astronomy'
```

Because there are two tables involved, the system checks the rules in Table 5 and adds the resource_author to the table list. It also adds this expression to the condition list:

```
resource_author.resource_id=resource.resource_id
AND resource_author.author_id=author.author_id
```

The final SQL query is:

```
SELECT author.name FROM author,resources, resource_author
WHERE resource.title='General Astronomy'
AND resource_author.resource_id=resource.resource_id
AND resource_author.author_id=author.author_id
```

3.8 The action template <table1><action_verb><table2>

If the system fails to match the user input with any of the preceding templates, the next step is to make a check against the action templates. The action template is used by NLPQC when it tries to identify the tables that are in a relation of type **<table1>-<action>-<table2>-<default_attribute_table2>**

Figure 22 shows the action template.

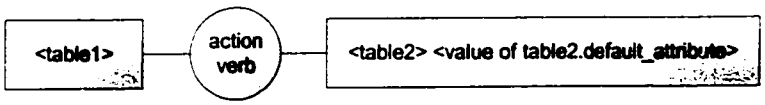

Figure 22. The action template

The system administrator creates the associated SQL template is:

```
SELECT table1.default_attribute FROM table1, table2
WHERE table2.default_attribute=value of table2.default_attribute
```

A word can relate through an action to another word. In the example *Show the books written by author Mark Twain*, *book* receives the action *written* from the *author* 'Mark Twain'.

In our database, there are three possible direct words: *user*, *resource* and *author*. The tables and the associated action verbs are shown in Table 9. The system administrator fills in the active and passive actions, and the counterpart table. WordNet fills in the semantic set for the action verbs. The semantic set for the action verb *write* is formed of *write, compose, pen, indite, publish, mark, spell*. The counterpart to *resource* is *book*, and thus an *author* writes *resource*, or *resource* is written by *author*.

Table 9. Tables and action verbs

Word	Active action	Passive action	Related word	Default attribute
resource		Write, compose, indite, publish, mark, spell	*author*	title
author	Write, compose, indite, publish, mark, spell		*resource*	name

Example:

Show all books written by author Mark Twain
Show all books published by author Mark Twain[22]
Show all books composed by author Mark Twain[23]

The three examples are parsed in the same way, and they eventually produce the same SQL query. The NLPQC system fails to match the input with the **'Of'** templates then it proceeds with the action template:

Show all books **written** by author Mark Twain.

Because the action verb *write* is found in the sentence, NLPQC checks the surrounding tokens:

[22] The NLPQC does not deal with numbers such as in *Show **3** books written by author Mark Twain*
[23] The NLPQC cannot parse sentences such as *Show all books **not** composed by author Mark Twain*

Show all *books* **written** by *author* Mark Twain.

Book is identified as table1 and *author* is table2. From Table 7, the system finds out that *book* is related to table *resource* and *author* is related to table *author*. From Table 6, the system finds the default attribute for *author*, which is *name*. The default attribute for *resource* is *title*. The system fills in the data in the SQL template:

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

From Table 5, the system finds out that *resource, author* are related to *resource_author*, and thus the table list becomes *resource, author, resource_author*. The final SQL query is:

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

The three examples for the three elementary templates presented above are summarized in the process flow chart shown in Figure 23. The user input is syntactically parsed, and then NLPQC tries the first **<attribute>of<table>** template. If the **attribute** and the **table** are found in the semantic sets, the corresponding table and attributes names are used to build the SQL query. The remaining of the sentence is considered to be the value of the default attribute of the table.

If the **<attribute>of<table>** template does not match the input, the system tries the **<attribute>of<table_1>of<table_2>** template. NLPQC uses the semantic sets to associate table names to **table_1** and **table_2**. Then it relates **attribute** to one of the attributes of the **table_1**. If the template is successfully matched with the sentence, the remaining text is considered to be the value of the default attribute of the **table_2**.

If the **<attribute>of<table_1>of<table_2>** template does not match the user input, the system tries the action verb template. NLPQC looks for verbs in the sentence that match the action verbs in Table 9. If a match is found, the corresponding pair of tables forms the table list in the query. The system checks the Table 5 to find if there are any other tables involved. For example, if *resource* and *author* are in the table list, the *resource_author* table must be added as well. The remaining of the sentence is considered to be the value of the default attribute of the second table in the table list.

The default attributes are retrieved by NLPQC from the Table 6.

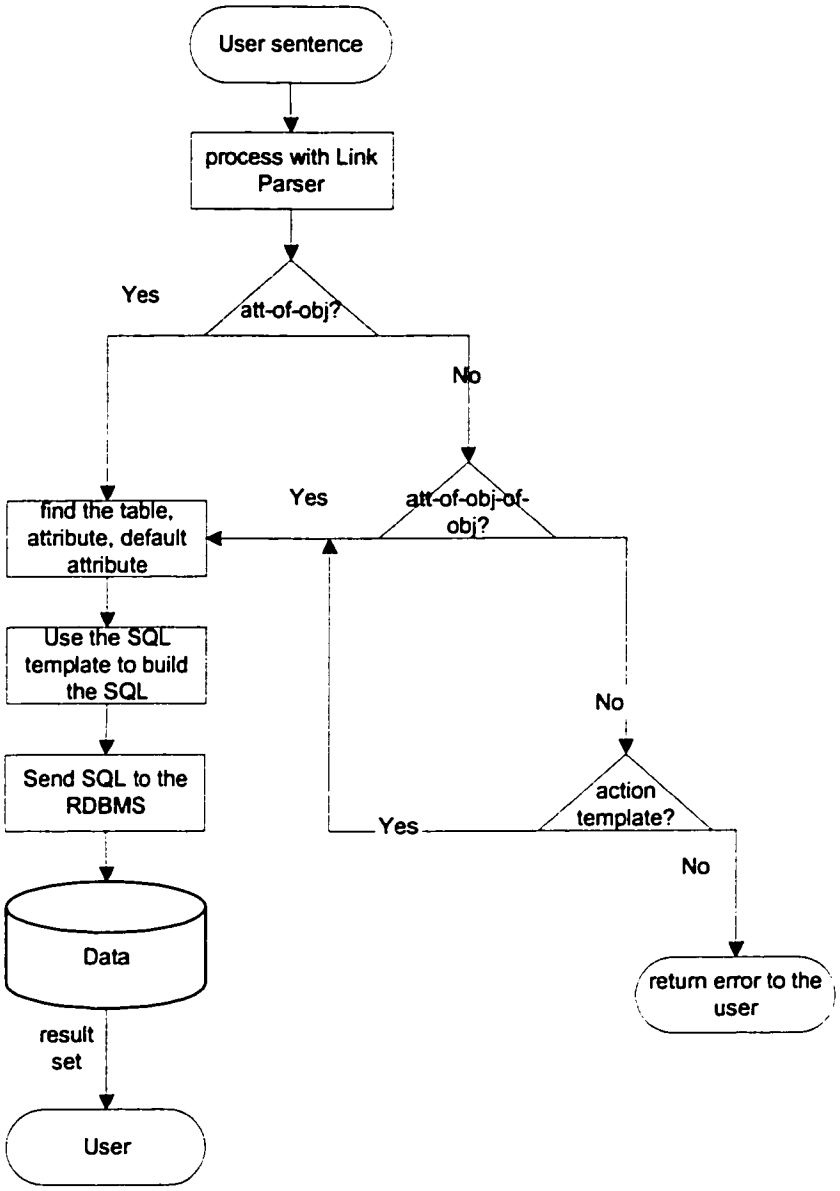

Figure 23. The flow chart for the 3 elementary templates

3.9 The NLPQC detailed system architecture

The system is composed of a pre-processor and a run time processor, as shown in Figure 24:

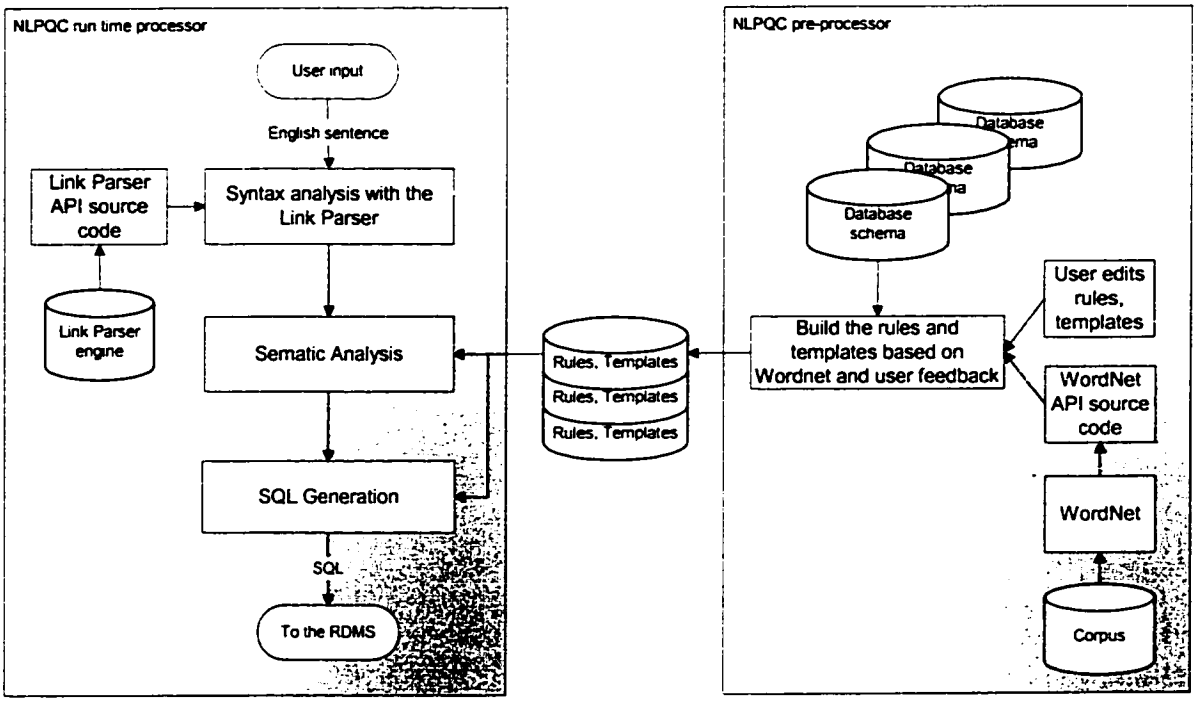

Figure 24. NLPQC detailed architecture

- The pre-processor reads the database schema and generates the system rules. To generate the rules, the pre-processor uses WordNet to extract the synonyms, hypernyms and the hyponyms of each keyword found in the schema file. For example, the schema file might contain words such as *book* or *author*. In this case WordNet returns the semantically related tokens. If the schema contains "Book_ID" as attributes, WordNet fails to find any relatives. In this case the system administrator intervenes and types in the dictionary word that is the closest to the *book_ID*, namely for example, *Identification*. The system administrator may type in more than one token in order to enlarge the capability of the rules generator. The pre-processor can run on any database. Figure 24 shows three different schema files.

- The NLPQC run time processor uses the Link Parser to syntactically parse the user input and then uses rules and templates to do the semantic interpretation and build the SQL query.

3.10 The NLPQC preprocessor details

The NLPQC pre-processor runs only once on each database schema. This is done before run time, in order to prepare the set of rules that are later used by the NLPQC system for the semantic parsing and the building of the SQL query. First the NLPQC pre-processor reads the schema of the database. Then it

identifies the table names, the attributes and the keys in the schema. It sends every token that has been found in the previous step to WordNet, asking for a list of all synonyms, hypernyms and hyponyms. If the token is not found in the WordNet dictionary, the system administrator is asked for input. The process is interactive, and the system administrator validates every token and the related list of terms returned by WordNet. Table 7 in section 3.3 shows two semantic sets. The next step is building the rules. There are several categories of rules and templates, which have been detailed in sections 3.6 to 3.8. The rules are used by the NLPQC system in the semantic parsing process, and in the generation of the SQL query for the database. The schema of the database is illustrated in Appendix C.

3.11 Using WordNet

After reading the schema file, the NLPQC pre-processor identifies each table name and attribute. It uses WordNet to find all the synonyms, hypernyms and hyponyms of the current token. Figure 25 shows the process flow for using WordNet. The following example shows the processing steps of the table *resource* from the database schema.

> table *resource* (resource_id, title, alt_title, language_id, keyword, created_date, upload_date, expiry_date, last_update, version, source, annotation, size, resource_format, abstract, filename, contributor_id,);

The table name is *resource*. The NLPQC pre-processor sends the token to the WordNet interface and receives the following result set:

```
Synonyms: 3 senses of resource
Sense 1: resource
Sense 2: resource
Sense 3: resource, resourcefulness, imagination
Hypernyms: 3 senses of resource
Sense 1: resource
Sense 2: resource, asset, plus, quality, attribute, abstraction
Sense 3:
resource, resourcefulness, imagination, inventiveness, ingeniousness, ing
enuity
Hyponyms: 3 senses of resource
Sense 1: resource,
natural resources, labor resources, support, keep, livelihood, living
Sense 2: resource, aid, assistance, help,
recourse, refuge, resort, resourcefulness
```

Sense 3: resource, resourcefulness, imagination,
armory, armory, inventory

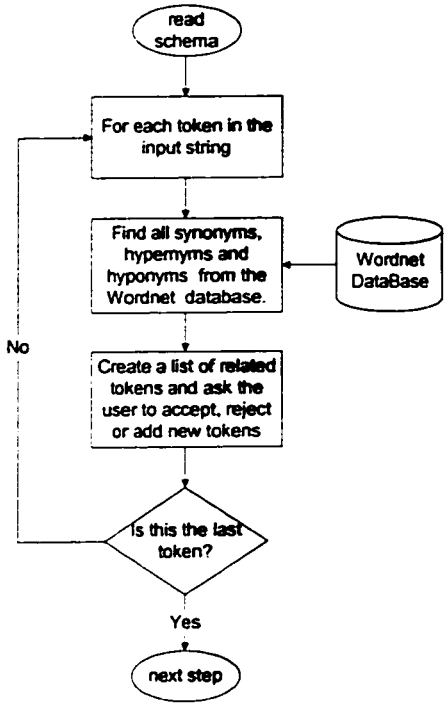

Figure 25. WordNet process flow

The NLPQC pre-processor builds the following list of terms, which are related:

resource = {assets, plus, resourcefulness, imagination, possession, quality, attribute, abstraction, ability, power, cognition, knowledge, support, keep, livelihood, living, sustenance, funding, backing, aid, assistance, aid, help, recourse, refuge, resort, armory, resource, book, record }

If the system administrator is not pleased with the result he can add the following underlined terms, through the interactive process:

resource = {assets, plus, resourcefulness, imagination, possession, quality, attribute, abstraction, ability, power, cognition, knowledge, support, keep, livelihood, living, sustenance, funding, backing, aid, assistance, aid, help, recourse, refuge, resort, armory, resource, book, volume, record, script }

The system administrator may also delete terms. For example the system administrator may retain only the following list that is relevant to the schema of the database:

```
resource = {resource, book, volume, record, script}
```

The result set represents the semantic set for the *resource* token. The process presented above takes place for each table name and for each attribute of the database. The final result is a list of tokens and the associated list of related terms. If the token found in the schema of the database is not an English word, the system administrator provides the necessary entry. For example, in each of the tables *contributor*, *user*, *language*, *subject*, *role*, *author* and *resource* there is an attribute related to *name*. The table *role* has the attribute *role_name*, the table *language* has the attribute *language_name*. In both cases the system administrator will propose the alternative token *name*.

The goal of using WordNet is to allow the NLPQC pre-processor to be more flexible compared to other systems, such as the NLBean and the SQ-Hal systems that have been described in Chapter 2. These systems try to match the tokens identified in the user input with table names and attributes from the schema. If there is no perfect match, the process fails. In the NLPQC system, we allow the user to use a variety of terms that are all related to a table name, or to an attribute of the schema. On one hand this allows more flexibility, on the other there is greater risk to fail the semantic parsing if a token is related to two different table names or attributes. In order to address this issue, the NLPQC system allows the user to intervene at the pre-processing time, and to correct the list of related tokens if necessary. The overall result is a higher precision and more flexibility compared to other systems in the NL interface domain.

3.12 An example of run

In the following sections, we will show a complete example of how the system will create the corresponding SQL query for the sentence *What is the address of author Leiserson?* First, the system administrator runs the NLPQC preprocessor on the Cindi schema. Here is the actual output after using WordNet to build the semantic set for the table *resource*:

```
// Semantic set
STRING table0[] =
{
"resource","resources","resourcefulness","imagination","book","bo
oks","#",
"title","statute","championship","deed","of","conveyance","claim"
,"entitle","#",
```

```
"resource_id","#",
"alt_title","#",
"language_id","#",
"keyword","#",
<skip>
""
};
```

WordNet returns a full set of synonyms, hypernyms and hyponyms for each table and attribute name. The set is also called the semantic set of the name. The cardinality of a set can be as high as 50 elements. Because not all of the semantic set elements are relevant to the name, the system administrator has to edit the result. Some of the elements are removed from the set, and others are added as necessary. For example, WordNet does not return *book* as a synonym for *resource*, as we would like to have for the Cindi database. In this case, she adds *book* to the semantic set of the *resource* table. The administrator also adds the table rules and the SQL templates to the pre-processor output. The following example shows the edited semantic set for the table *resource* and its attributes, and the table and verb rules that are added by the administrator:

```
// Semantic set
STRING table0[] =
{
"resource","resources","book","books","Resource","Resources","Boo
k","Books","#",
"title","Title","#",
"resource_id","#",
"alt_title","#",
"language_id","#",
"keyword","#",
<skip>
""
};

// Schema rules
STRING cross_reference[] =
{
"author,resource","resource_author","AND
resource_author.resource_id=resource.resource_id AND
resource_author.author_id=author.author_id",
<skip>
```

```
};

// Action verb rules
STRING action[] = // action, object1, object2 (related objects)
{
      "written", "resource", "author",
      "wrote",   "author","resource" ,
      "write",   "author","resource" ,
      "writes",  "author","resource" ,
      "writing", "author","resource" ,
};
```

The pre-processor output requires manual editing. For example, it took 3 hours for me to edit the information related to the 14 tables of the Cindi database used in this project.

The system administrator then builds the NLPQC run time module with the rules shown above. The end user can type in the query *What is the address of the author Leiserson?*

The NLPQC system uses the Link Parser to syntactically process the query. Here is the actual output edited for clarity. The **bold** text is NLPQC output and the *italic* text shows the links found by the Link Parser:

Enter query: What is the address of the author Leiserson?
++++Time 0.12 seconds (0.12 total)
Found 4 linkages (2 had no P.P. violations)
NLP: print the link information:
LEFT-WALL - Xp - ? : **Connects to the period at end of the sentence**
LEFT-WALL - Ws - what : **Subject question**
*what - Ss*w - is* : **Question-words like 'who' that can act as noun-phrases**
is - Ost - address : **Connects transitive verb to direct or indirect object (s)**
the - Ds - address : **Connects determiner to noun**
address - Mp - of : **Prepositional phrase modifying noun**
of - Js - Leiserson : **Connects preposition to its object**
author - GN - Leiserson : **Proper noun is introduced by a common noun**
the - DD - author : **Connects definite determiners (the, his, Jane's) to number, expression or adjective acting as nouns**

? - RW - RIGHT-WALL : **Connects the right-hand wall to the left-hand wall**

object=author, table=author, att=address, def_att=name, value=Leiserson

SELECT author.address FROM author WHERE author.name='Leiserson'

The following graphical representation shows the same parse tree as above, in a more readable format:

```
  Linkage 1, cost vector = (UNUSED=0 DIS=2 AND=0 LEN=11)
    +---------------------------Xp--------------------------+
    |             +----Ost---+      +--------Js--------+     |
    +---Ws--+Ss*w+   +--Ds--+--Mp--+  +--DD--+---GN---+     |
    |       |    |   |      |      |  |      |        |     |
LEFT-WALL what is.v the address.n of the author.n Leiserson ?
```

The LEFT and RIGHT walls are added by the Link Parser as helping delimiters. One link in the parse tree relates two tokens from the input sentence. In the example *is - Ost – address, Ost* connects the transitive verb *is* to its direct object *address*. NLPQC uses this information to identify the direct or indirect objects in the input, and to match them to table names in the database, with help from the semantic sets. The Link Parser returns zero, one or many parse trees. If there are more than one parse trees, NLPQC picks the first one, which has the lowest link cost. The link cost is a measure of the precision of the parse tree, and the Link Parser computes it. The link cost is the sum of UNUSED, DIS, AND and LEN parameters, as they are returned by Link Parser. If the minimum cost parse tree returned by the Link Parser is wrong, the NLPQC system will fail to produce a correct SQL query.

Next, the system tries to identify the tables involved. To do this, it checks on each noun in the input sentence if it belongs to one of the semantic sets. It finds that *author* is in the semantic set of the table *author*. Then it finds out the *address* is in the semantic set of the attribute *address* of the table *author*.

The next step is to apply the templates. The first template is **<attribute>of<author>.** The input matches the template. The default attribute of table *author* is considered for the rest of the sentence, i.e. *Leiserson*. The following SQL query is generated:

```
SELECT address FROM author
WHERE author.name="Leiserson"
```

The SQL query is ready to be sent to the database engine.

4. The NLPQC implementation

The language used for the implementation is "C/C++". The prototyping of the user interface for the WWW deployment has been done in Java. The development environment is MSDEV with MSVC compiler.

The NLPQC pre-processor has been implemented in the "C++" language, based on the object-oriented concepts. The main C++ classes of the pre-processor are listed in Appendix E.

The NLPQC system has been implemented in the "C" language because of the restriction imposed by the Link Parser source code. It could have been mixed with "C++" but there were many complications on the Link Parser side. However, the same object oriented approach has been used for the NLPQC run time system as for the NLPQC pre-processor.

The NLPQC system has been integrated with WordNet [Miller, 1995] and the Link Parser [Temperley, Sleator, Lafferty, 2002]. The NLPQC system has been primarily developed on the Windows NT 4 platform, but it will work on Unix also, as soon as the Link Parser code for Unix is released[24].

4.1 The integration of WordNet

WordNet® is an online lexical reference system. WordNet has been developed at Princeton University [Miller, 1995]. More details on WordNet can be found in Appendix A. WordNet can be downloaded for both Unix and PC. For the purpose of this thesis, the PC version has been downloaded. Nevertheless the code is platform independent.

WordNet comes with a Database Package that includes WordNet 1.6 database, interfaces, sense index, gloss index, interface and library source code, and documentation. This package requires 37MB of disk space. Once WordNet has been downloaded, it can be installed on the system. There are two major components in the WordNet development system:

- The run time module to accesses the database, which contains the English dictionary. The source code for this module is not published.

[24] The Link Parser binary code has been released for both Windows and Unix platforms. However, the NLPQC system needs the source code to integrate the Link Parser. Only the source code for Windows has been released before August 2002.

- The user interface that is integrated with the NLPQC system, which is written in the "C" language. The source code is available for both Windows and Unix operating systems.

WordNet uses two types of processes to try to convert the input string into one that can be found in the WordNet database. There are lists of inflectional endings, based on syntactic category, which can be detached from individual words in an attempt to find a form of the word that is in WordNet. The WordNet interface to the NLPQC system is implemented in the WN.c source file.

The NLPQC pre-processor uses this command line to call WordNet:

Wn -<option> <token>

There are three WordNet options, which are used by the NLPQC system: synonyms, hyponyms and hypernyms.

The source code for the WordNet interface has been included among the project files for the NLPQC pre-processor. The NLPQC pre-processor includes a link library from the WordNet engine. The engine is in binary format and it is royalty-free.

4.2 The integration of the Link Parser in the NLPQC system

The Link Parser is a syntactic parser of English, based on link grammar, an original theory of English syntax [Temperley, Sleator, Lafferty, 2002]. It is described in detail in Appendix B. Given a sentence, the system assigns to it a syntactic structure, which consists of a set of labeled links connecting pairs of words (Appendix B for details). The system is written in generic C code, and runs on any platform with a C compiler. There is an application program interface to make it easy to incorporate the parser into other applications. The Link Parser has been loaded from http://www.link.cs.cmu.edu/link/ftp.html and it comes with these files:

- link-4.1 - source code for the parser
- translator - source code for the translator
- system-4.1.tar.gz - compressed version of the parser and translator files for Windows only
- link42.zip - compressed version of the Windows version of the parser

The authors have released the source code for Windows and will release the source code for Unix in the future. This situation has been identified as a risk dependency for the NLPQC system. The Link Parser source code has been integrated with the NLPQC system through the MSDEV environment.

The NLPQC processor receives the user input and sends it to the Link Parser for processing. The Link Parser analyzes it to extract the tokens and then it tags them with parts-of-speech and identifies the syntactic constituents. The Link Parser uses a dictionary and a set of rules to identify the relations between the tokens. The relations are based on rules, as explained in Appendix B.

The NLPQC system passes the user input to the Link Parser, which returns the results of the syntax parsing, as shown in Figure 26. The connectors are explained in Appendix B. With the sentence *Show all books by Leiserson*, the Link Parser identifies the action verb *show* and *books* as its direct object. In this case, the Link Parser has related *Leiserson* to the verb *show* which is wrong. It should be related to *all books*. The Link Parser often returns more than one parse tree for a given sentence, along with a confidence level. One way to select the best answer among the set of trees is to use the "link cost" of the result. The "link cost" is a parameter computed by the Link Parser, and expresses the confidence level in the proposed result. In Figure 26 the link cost is LEN=8.

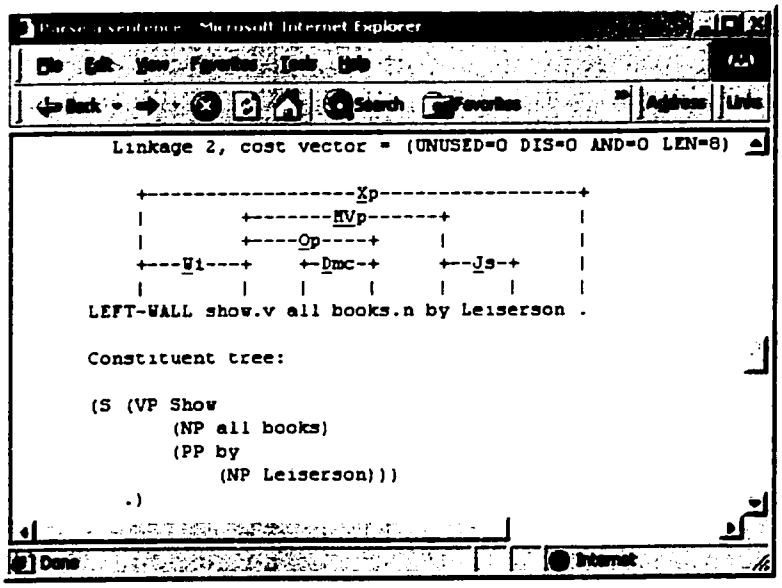

Figure 26. The Link Parser output

Figure 27 shows the result after taking in consideration the "link cost". This time the modifier *Leiserson* is correctly linked to the words *all books*. The cost of the link is LEN=6.

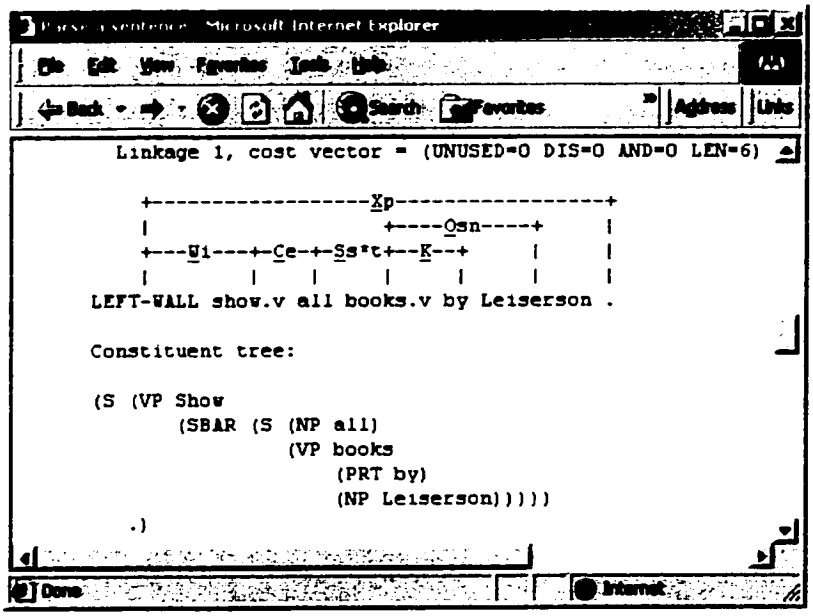

Figure 27. The Link Parser output after correction

The NLPQC system integrates the Link Parser, and it gets the best parse tree from it. The parse tree is used to extract the individual tokens and to match them to one of the three templates presented in Chapter 3. The connectors returned by the Link Parser are used to establish if the current token is an attribute, a table or an action verb. The connectors are further used to link the attributes to their words, and to establish which of the two words in an action template is the passive one.

4.3 The development environment

The development environment is based on the Windows NT 4 platform, and it uses the MSDEV tools to build the NLPQC system. However, the NLPQC system is designed to be platform independent. The only unresolved dependency is related to the Link Parser. The source code for the Link Parser has been released for Windows, and it will be soon released for Unix. The compiler used is MSVC 6.0. Figure 28 shows the MSDEV environment and the four MSVC projects developed for the NLPQC system.

The NLPQC prototype has been written in Java. To compile the Java code, the Sun JDK toolkit 1.3 has been used.

The source file control, or version control, has been implemented through the use of the Visual Source Safe tool from Microsoft. When the NLPQC system will be compiled for Unix, the version control tool will be the PCV system.

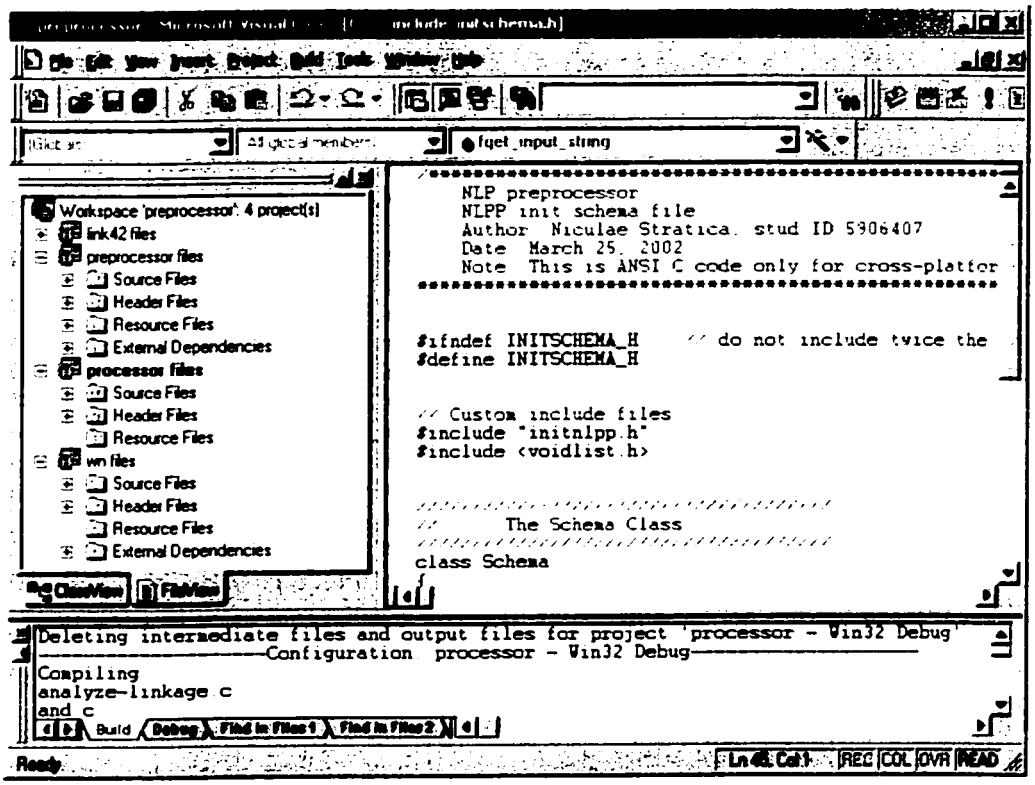

Figure 28. The integrated development environment

4.4 The build process

The build is done through the use of the MSDEV, MSVC and Sun Java SDK. The Sun Java compiler is run at the command prompt to build the Java classes. The NLPQC interface to the database is run at the command prompt. The command prompt accepts the user input, expressed in English. The NLPQC interface uses the Link Parser to returns the most probable links. The NLPQC interface uses the best link cost to determine which parse tree to use for the generation of the SQL query. The resulted SQL query is printed on the screen.

4.5 The run time environment

The NLPQC system has been primarily developed on Windows NT 4. The WordNet source code is available on both Unix and PC platforms. The Link Parser source code is only available on PC today, but

the authors of the Link Parser promised to release the Unix version soon. Given these, the NLPQC system is designed to be platform independent, for use under Unix and Windows.

The NLPQC system can connect to any relational database engine, through the ODBC interface. The NLPQC system is by design database independent. It can accept the schema of any relational database. If the system administrator decides to switch from one database to another, the NLPQC pre-processor must be run before any input can be processed.

5. Experimental Results

The NLPQC system has been tested on the schema of the Concordia virtual library system, named Cindi. The schema is detailed in Appendix C. A set of queries formulated by the author has been run to test NLPQC's performance. The following screen outputs show various queries that have been ran on the NLPQC interface from the command line prompt.

In the following sections we present a series of experiments. Some of them failed, thus showing the limitations of the system. The connectors in the examples shown are detailed in Appendix B. They are also briefly explained in the NLPQC screen output.

5.1 Examples

Example 1, Figure 29: Input = *book*

The system tries to match *book* on a semantic set. It finds the word *book* in the semantic set *resource*. There are no more words in the sentence, and thus the system uses the default attribute of *resource*, which is *title*.

Figure 29. Example 1: *book*

NLPQC tries the <attribute>of<object> template, and produces the following SQL query:

```
SELECT title FROM resource
```

The query is correct from the system's point of view, although it might not return to the user the wanted information.

Example 2, Figure 30: Input = *show books*

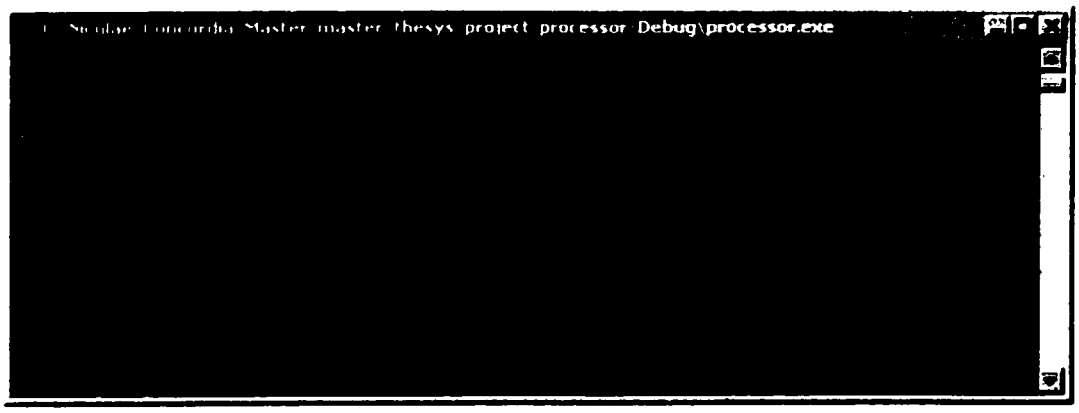

Figure 30. Example 2: *show books*

The NLPQC behaves like in the Example1. It cannot match *show* with any of the table names or attribute and thus it discards it.

Example 3, Figure 31: Input = *Show all books*

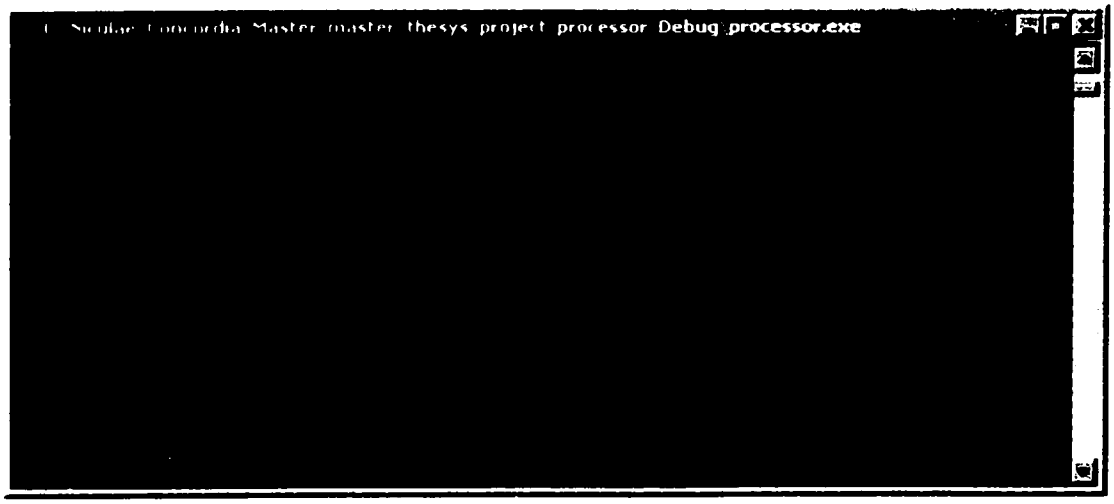

Figure 31. Example 3: *show all books*

The NLPQC behaves like in the Example1. It cannot match *show* nor *all* with any of the table names or attribute and thus it discards them.

Example 4, Figure 32: Input = *What books?*

The system behaves like in Example 1.

Figure 32. Example 4: *What books?*

Example 6, Figure 33: Input = *Books by Mark Twain*. The system cannot build the SQL query.

Figure 33. Example 6: *Books by Mark Twain.*

The NLPQC system related *Books* to *resource*, but it wrongly uses *Mark Twain* as value for the default attribute of the table *resources*.

Example 7, Figure 34: Input= *List books written by author Mark Twain.*

Figure 34. Example 7: *List books written by author Mark Twain.*

The NLPQC system found table1=*resource* which is related to *book* and table2=*author*. The system cannot identify the words with any attribute from table 1 and 2, and thus it uses the default attributes instead. The default attribute for table *resource* is *title* and the default attribute for table *author* is *name*. The remaining of the input sentence is considered to be the value of the attribute *name*.

Example 8, Figure 35: Input= *What is the language of the book Algorithms?*

The system identifies the *book* as being a *resource* and it matches *language* with the table *language*. Next, NLPQC matches the template **<attribute>of<table1>of<table2>** with the input sentence. The **attribute** is not found in the input sentence, and thus the default attribute of table 1 is used. The following SQL query is generated:

```
SELECT language.name FROM language,resource
WHERE resource.title="Algorithms"
AND language.resource_id=resource.resource_id
```

The system uses the default attribute of the table 2, which is *title*. The remaining of the phrase is considered to be the value of the *title*.

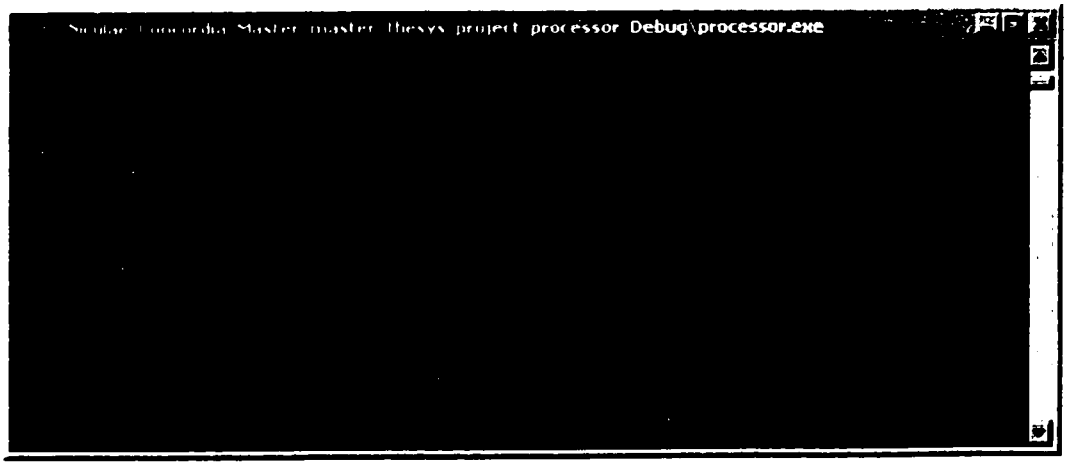

Figure 35. Example 8: *What is the language of the book Algorithms?*

Example 9, Figure 35: Input=Who wrote Algorithms?

Although the NLPQC system does not find any word related to a table or to an attribute in the input, it uses the action verb to find out that tables *author, resource* are in the table list.

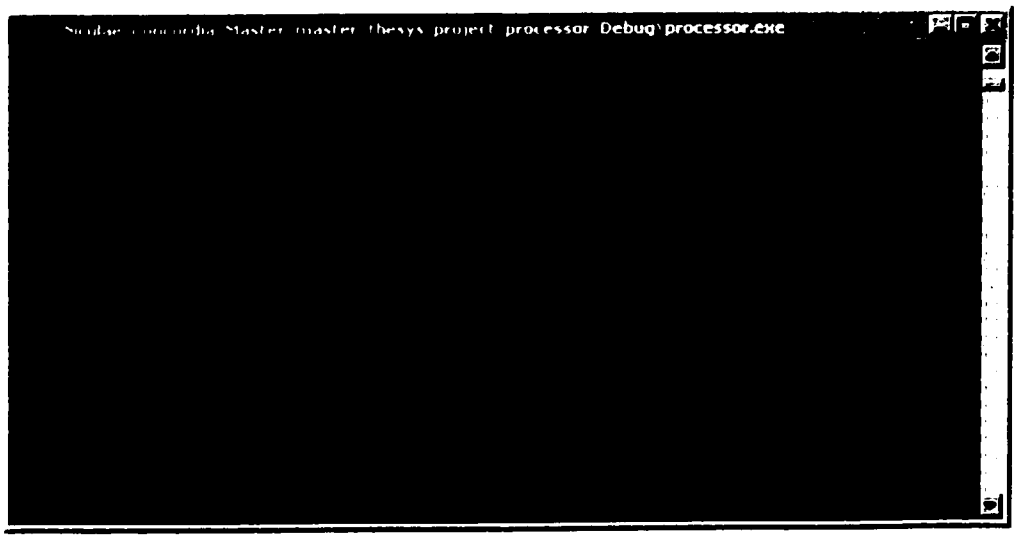

Figure 36. Example 9: *Who wrote Algorithms?*

5.2 Summary

The system has been tested with various sentences. Examples of some tests are shown in Table 10 along with the results and the limitations. The failures can be addressed by introducing more templates in the future but care must be taken because adding more templates may in fact increase the probability of failure due to ambiguities. Some short sentence examples failed, while their more complex variants did pass. This behavior will be fixed by introducing simpler templates in the run time module of the NLPQC. This will not solve all problems, but it will allow us to incrementally raise the system's performance.

Table 10. Various input sentences and 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'`
3	What is 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.
4	What is the address of author Mark Twain.	Pass	Table=author, template <attribute-of-table> `SELECT address FROM author WHERE name='Mark Twain'`
5	Show the author of 'General Astronomy'.	Fail	Table was not identified. No word in the sentence matches any of the table names, nor their corresponding semantic sets.
6	Show the author of book 'General Astronomy'.	Fail	Attribute of table1 was not identified. No word in the sentence matches any of the attributes in table *author*, nor their corresponding semantic sets.
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.
9	Who wrote the book 'General Astronomy'?	Fail	Template missing. None of three templates matches the sentence.
10	Which author wrote the book 'General Astronomy'?	Fail	Template missing. None of three templates matches the sentence.
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
12	Show all books **written** by Mark Twain.	Fail	Table2 was not identified. No word in the sentence matches the table 2 name, or its corresponding semantic sets.
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
14	Which are the books **written** by Mark Twain?	Pass	Table2 was not identified. No word in the sentence matches the table 2 name, or its corresponding semantic sets.
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

14	Which are the books **written** by author Mark Twain after 1870?	Fail	Time template not implemented.

The NLPQC system appears to work correctly on sentences that fit the existing templates. The limitations are due to missing rules and templates. On the other hand, it is relatively easy to improve the system due to the object-oriented architecture and to the rationale behind the system architecture. The system is open and allows for future improvements. Figure 37 shows the supported templates.

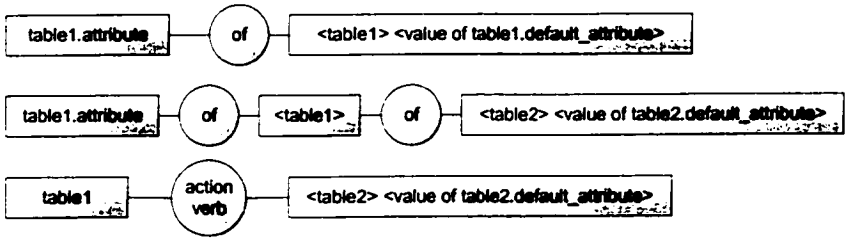

Figure 37. The three supported templates

The future development will include the development of new elementary templates and of the composed templates. Figure 38 shows an example of composed template.

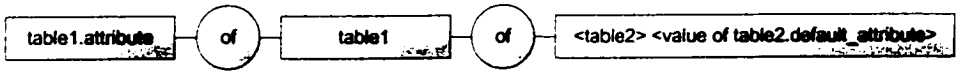

Figure 38. Composed template

The template shown above can handle expressions such as 'What is the address of the author of the book General Astronomy?'

6. Conclusions and plans for future development

The NLPQC system presented in this thesis has the following main characteristics:

1. The system is built on top of the existing proven tools and techniques in order to execute the token parsing and tagging, and to do semantic parsing. In order to achieve these goals, WordNet and the Link Parser have been integrated into the NLPQC system.
2. The system functionality has been divided into two parts: **a.** The pre-processor interprets the database schema and builds the system rules, which are later used into the semantic parsing. **b.** The run time module processes the user input, and does the semantic parsing based on the rules established by the pre-processor.

The NLPQC system uses a combined top-down and bottom-up approach. The top-down approach is used in the early stages of input parsing, in order to decompose the user input into 2 or 3-tuple templates. The system improves the quality of the parsing by using pre-computed information, based on the actual structure of the database. The pre-processor does the bottom-up processing. It starts from the bottom of the process flow chart shown in Figure 23 and tries to build the templates on top of the database schema.

One difference between QA and NLPQC is that QA looks for answers in a collection of unstructured documents in any domain, while the NLPQC is limited to a structured database of limited discourse domain. On the other hand, the database contains highly organized information, and thus the search domain is greatly reduced. A QA system brings in a set of answers, which is close to the original question. NLPQC works more like a pass/fail device. If the SQL query is correctly built, the answer is considered to be true, if not it is false. If the SQL cannot be built, the system does not return any answer.

The NLPQC system relies on the Link Parser to do the syntactic parsing, and then it tries to semantically analyse the input and build the SQL query. The confidence level is related to the link cost returned by the Link Parser. NLPQC always processes the lowest cost link as returned by the Link Parser. Special care has been taken in implementing the mechanism of the rule generation in the pre-processor.

Although our work satisfies our initial goal, it can be improved in several ways. Here is the list of future developments, by order of importance:

- The very first priority in the future is to increase the **processing power** of the NLPQC system. This can be performed by two means:

1. Refining the rule generation process. This will involve the creation of additional word categories, more generalization and more interfaces between words. Each interface will describe the relations among different words. This mechanism is later used for the semantic parsing of the user input. The work will be based on the concept of basic objects introduced by Rosch [Rosch et al., 1976].
2. The user input is presently processed through three templates. In the future, more templates will be implemented to address the limitations of the present system. This will be done in two steps:

   2.1. The attributes will be described by a simple entity, or by a combination of entities

   2.2 The template will be diversified. Example: *table-action-table*, *modifier-table-action*.

   The NLPQC system acquires 'understanding' capability, in a sense that it interprets declarative statements. Example: User: Gabriel goes to school. The system finds 'Gabriel' in the database, and then it finds the name of the school 'Dawson College' and it returns to the user: 'Gabriel studies at Dawson College in Montreal'

- A **memorizing module** should be built: the NLPQC system will store the user queries and the resulting SQL query, if valid. The stored knowledge will be later used in processing similar user queries. The system will follow the strategy introduced by Harabagiu [Harabagiu 2000].
- A **feedback-to-user** functionality should be implemented: after processing the user input, the NLPQC system will construct the corresponding SQL query. Subsequently, the SQL query will be back-processed into English and returned to the user, to evaluate the correctness of the NL processing that was done by the system. The work will be based on the concepts introduced by Gal [Gal, 1885].
- The NLPQC system should be **generalized** to accept any Relational Database that is described through means of a schema. The present NLPQC system is limited to two-table templates and to three-table SQL queries. The system will be redesigned and modularized in order to accommodate more than three tables in a query. The future developer should introduce generic categories of words through word interfaces, and the associated dependencies. The pre-processing will be done in two steps:
  1. Go from the concrete schema to the generic word hierarchies and
  2. Identify relations between words, and incorporate these relations in the word interfaces.
- The NLPQC system should **accept more than one database schema** at a time.
- The NLPQC system should work on **distributed databases** through Internet, over TCP.
- A standard evaluation methodology should be used, having as model the proceedings of the TREC conference [TREC-X, 2001]. Rules and standards should be created in order to compare two different NL interfaces for the RDBMS.

Appendix A – WordNet

WordNet® is an online lexical reference system whose design is inspired by psycholinguistic theories of human lexical memory (http://www.cogsci.princeton.edu/~wn). English nouns, verbs, adjectives and adverbs are organized into synonym sets, each representing one underlying lexical concept. Different relations link the synonym sets. WordNet was developed by the Cognitive Science Laboratory at Princeton University under the direction of Professor George A. Miller.

The WordNet system consists of lexicographer files, code to convert these files into a database, and search routines and interfaces that display information from the database. The lexicographer files organize nouns, verbs, adjectives and adverbs into groups of synonyms, and describe relations between synonym groups.
Information in WordNet is organized around logical groupings called synsets. Each synset consists of a list of synonymous words or collocations (eg. ***"fountain pen"*** *,* ***"take in"*** *), and pointers that describe the relations between this synset and other synsets. A word or collocation may appear in more than one synset, and in more than one part of speech. The words in a synset are logically grouped such that they are interchangeable in some context.*
Two kinds of relations are represented by pointers: lexical and semantic. Lexical relations hold between word forms; semantic relations hold between word meanings. These relations include hypernymy/hyponymy, antonymy, entailment, and meronymy/holonymy.
Nouns and verbs are organized into hierarchies based on the hypernymy/hyponymy relation between synsets. Additional pointers are be used to indicate other relations.
Adjectives are arranged in clusters containing head synsets and satellite synsets. Each cluster is organized around antonymous pairs (and occasionally antonymous triplets). The antonymous pairs (or triplets) are indicated in the head synsets of a cluster. Most head synsets have one or more satellite synsets, each of which represents a concept that is similar in meaning to the concept represented by the head synset. One way to think of the adjective cluster organization is to visualize a wheel, with a head synset as the hub and satellite synsets as the spokes. Two or more wheels are logically connected via antonymy, which can be thought of as an axle between the wheels.[Miller, 1995]

Glossary of lexical & semantic relations in WordNet

In following definitions **word** is used in place of **word or collocation** .

entailment A verb **X** entails **Y** if **X** cannot be done unless **Y** is, or has been, done.

holonym The name of the whole of which the meronym names a part. **Y** is a holonym of **X** if **X** is a part of **Y** .

hypernym The generic term used to designate a whole class of specific instances. **Y** is a hypernym of **X** if **X** is a (kind of) **Y** .

hyponym The specific term used to designate a member of a class. **X** is a hyponym of **Y** if **X** is a (kind of) **Y** .

indirect antonym An adjective in a **satellite synset** that does not have a **direct antonym** has an indirect antonyms via the direct antonym of the **head synset** .

meronym The name of a constituent part of, the substance of, or a member of something. **X** is a meronym of **Y** if **X** is a part of **Y** .

pertainym A relational adjective. Adjectives that are pertainyms are usually defined by such phrases as "of or pertaining to" and do not have antonyms. A pertainym can point to a noun or another pertainym.

subordinate Same as **hyponym** .

superordinate Same as **hypernym** .

synset A synonym set; a set of words that are interchangeable in some context.

troponym A verb expressing a specific manner elaboration of another verb. **X** is a troponym of **Y** if **to X** is **to Y** in some manner.

Appendix B – The Link Parser

The Link Grammar Parser can be downloaded from http://www.link.cs.cmu.edu/link. NSF, ARPA, and the School of Computer Science, at Carnegie Mellon University, supported this research project.

The Link Grammar Parser is a syntactic parser of English, based on link grammar, an original theory of English syntax. Given a sentence, the system assigns to it a syntactic structure, which consists of a set of labeled links connecting pairs of words. As of July 2000, the authors released version 4.0 of the parser. Among the new features of version 4.0 is a system which derives a \"constituent\" representation of a sentence (showing noun phrases, verb phrases, etc.) The thesis implementation integrates the Link Parser at the source code level.

The entire system is available for download on the web. The system is written in generic ANSI C, and runs on all platforms with a C compiler. There is an application program interface (API) to make it easy to incorporate the parser into other applications.

The parser has a dictionary of about 60000 word forms. It has coverage of a wide variety of syntactic constructions, including many rare and idiomatic ones. The parser is robust; it is able to skip over portions of the sentence that it cannot understand, and assign some structure to the rest of the sentence. It is able to handle unknown vocabulary, and make intelligent guesses from context and spelling about the syntactic categories of unknown words. It has knowledge of capitalization, numerical expressions, and a variety of punctuation symbols [Temperley, Sleator, Lafferty, 2002].

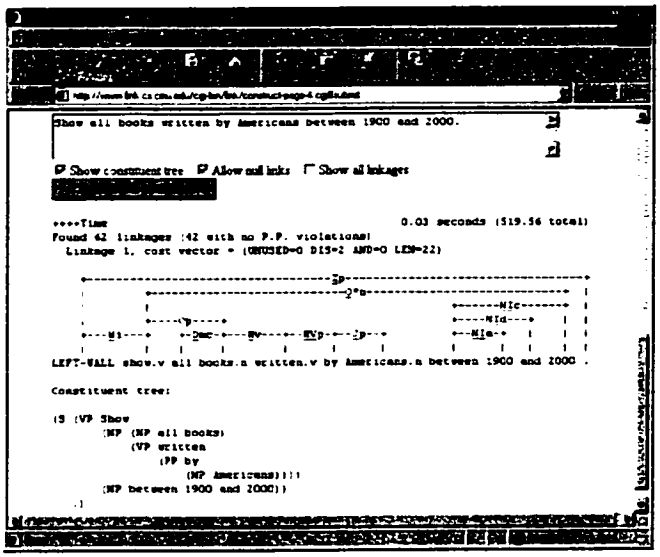

Figure 39. LinkParser example

Figure 39 shows an actual output from the online Link Grammar Parser. The input phrase is *Show all books written by Americans between 1900 and 2000.*

Here it is the actual output from the compiled version (ANSI C)

For our project, the Link Grammar Parser has limited utility, because it fails to correctly associate many of the phrases in our tests. On the other hand, if the user asks the parser to find all possible parse trees (for example they are 62 in the example presented above) at least one will be correct. Figure 40 shows the first two combinations out of 62 that were found by the Link Parser:

Here it is the summary of the links [Temperley, Sleator, Lafferty 2002] found on the WWW at http://www.link.cs.cmu.edu/link:

A connects pre-noun ("attributive") adjectives to following nouns: "The BIG DOG chased me", "The BIG BLACK UGLY DOG chased me".

AA is used in the construction "How [adj] a [noun] was it?". It connects the adjective to the following "a".

AF connects adjectives to verbs in cases where the adjective is fronted, such as questions and indirect questions: "How BIG IS it?"

AL connects a few modifiers like "all" or "both" to following modifiers: "ALL THE people are here".

AM connects "as" to "much" or "many": "I don't go out AS MUCH now".

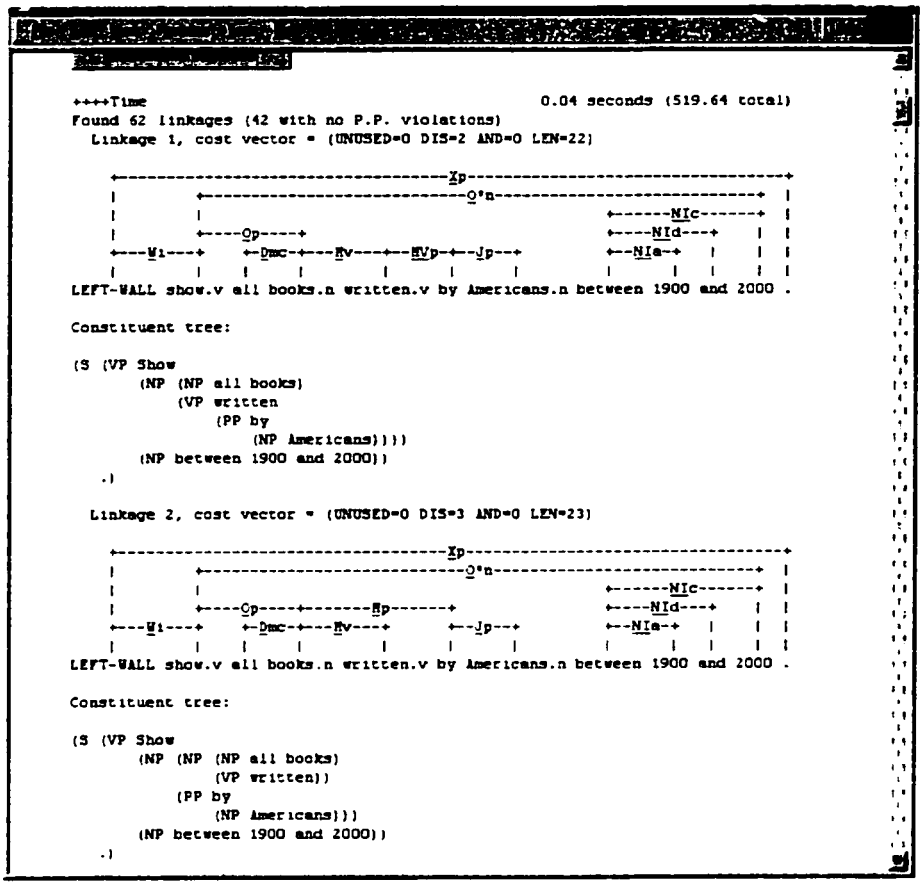

Figure 40. Another Link Parser example showing the link costs for two parse trees.

AN connects noun-modifiers to following nouns: "The TAX PROPOSAL was rejected".

AZ connects the word "as" back to certain verbs that can take "[obj] as [adj]" as a complement: "He VIEWED him AS tall".

B serves various functions involving relative clauses and questions. It connects transitive verbs back to their objects in relative clauses, questions, and indirect questions ("The DOG we CHASED", "WHO did you SEE?"); it also connects the main noun to the finite verb in subject-type relative clauses ("The DOG who CHASED me was black").

BI connects forms of the verb "be" to certain idiomatic expressions: for example, cases like "He IS PRESIDENT of the company".

BT is used with time expressions acting as fronted objects: "How many YEARS did it LAST?".

BW connects "what" to various verbs like "think", which are not really transitive but can connect back to "what" in questions: "WHAT do you THINK?"

C links conjunctions to subjects of subordinate clauses ("He left WHEN HE saw me"). it also links certain verbs to subjects of embedded clauses ("He SAID HE was sorry").

CC connects clauses to following coordinating conjunctions ("SHE left BUT we stayed").

CO connects "openers" to subjects of clauses: "APPARENTLY / ON Tuesday , THEY went to a movie".

CP connects paraphrasing or quoting verbs to the wall (and, indirectly, to the paraphrased expression): "///// That is untrue, the spokesman SAID."

CQ connects to auxiliaries in comparative constructions involving s-v inversion: "SHE has more money THAN DOES Joe".

CX is used in comparative constructions where the right half of the comparative contains only an auxiliary: "She has more money THAN he DOES".

D connects modifiers to nouns: "THE DOG chased A CAT and SOME BIRDS".

DD connects definite modifiers ("the", "his") to certain things like number expressions and adjectives acting as nouns: "THE POOR", "THE TWO he mentioned".

DG connects the word "The" with proper nouns: "the Riviera", "the Mississippi".

DP connects possessive modifiers to gerunds: "YOUR TELLING John to leave was wrong".

DT connects modifiers to nouns in idiomatic time expressions: "NEXT WEEK", "NEXT THURSDAY".

E is used for verb-modifying adverbs which precede the verb: "He is APPARENTLY LEAVING".

EA connects adverbs to adjectives: "She is a VERY GOOD player".

EB connects adverbs to forms of "be" before an object or prepositional phrase: "He IS APPARENTLY a good programmer".

EC connects adverbs to comparative adjectives: "It is MUCH BIGGER"

EE connects adverbs to other adverbs: "He ran VERY QUICKLY".

EF connects the word "enough" to preceding adjectives and adverbs: "He didn't run QUICKLY ENOUGH".

EI connects a few adverbs to "after" and "before": "I left SOON AFTER I saw you".

EL connects certain words to the word "else": something / everything / anything / nothing , somewhere (etc.), and someone (etc.).

EN connects certain adverbs to expressions of quantity: "The class has NEARLY FIFTY students".

ER is used the expression "The x-er..., the y-er...". it connects the two halfs of the expression together, via the comparative words (e.g. "The FASTER it is, the MORE they will like it").

EZ connects certain adverbs to the word "as", like "just" and "almost": "You're JUST AS good as he is."

FL connects "for" to "long": "I didn't wait FOR LONG".

FM connects the preposition "from" to various other prepositions: "We heard a scream FROM INSIDE the house".

G connects proper noun words together in series: "GEORGE HERBERT WALKER BUSH is here."

GN (stage 2 only) connects a proper noun to a preceding common noun which introduces it: "The ACTOR Eddie MURPHY attended the event".

H connects "how" to "much" or "many": "HOW MUCH money do you have".

I connects infinitive verb forms to certain words such as modal verbs and "to": "You MUST DO it", "I want TO DO it".

ID is a special class of link-types generated by the parser, with arbitrary four-letter names (such as "IDBT"), to connect together words of idiomatic expressions such as "at_hand" and "head_of_state".

IN connects the preposition "in" to certain time expressions: "We did it IN DECEMBER".

J connects prepositions to their objects: "The man WITH the HAT is here".

JG connects certain prepositions to proper-noun objects: "The Emir OF KUWAIT is here".

JQ connects prepositions to question-word modifiers in "prepositional questions": "IN WHICH room were you sleeping?"

JT connects certain conjunctions to time-expressions like "last week": "UNTIL last WEEK, I thought she liked me".

K connects certain verbs with particles like "in", "out", "up" and the like: "He STOOD UP and WALKED OUT".

L connects certain modifiers to superlative adjectives: "He has THE BIGGEST room".

LE is used in comparative constructions to connect an adjective to the second half of the comparative expression beyond a complement phrase: "It is more LIKELY that Joe will go THAN that Fred will go".

LI connects certain verbs to the preposition "like": "I FEEL LIKE a fool."

M connects nouns to various kinds of post-noun modifiers: prepositional phrases ("The MAN WITH the hat"), participle modifiers ("The WOMAN CARRYING the box"), prepositional relatives ("The MAN TO whom I was speaking"), and other kinds.

MF is used in the expression "Many people were injured, SOME OF THEM children".

MG allows certain prepositions to modify proper nouns: "The EMIR OF Kuwait is here".

MV connects verbs and adjectives to modifying phrases that follow, like adverbs ("The dog RAN QUICKLY"), prepositional phrases ("The dog RAN IN the yard"), subordinating conjunctions ("He LEFT WHEN he saw me"), comparatives, participle phrases with commas, and other things.

MX connects modifying phrases with commas to preceding nouns: "The DOG, a POODLE, was black". "JOHN, IN a black suit, looked great".

N connects the word "not" to preceding auxiliaries: "He DID NOT go".

ND connects numbers with expressions that require numerical modifiers: "I saw him THREE WEEKS ago".

NF is used with NJ in idiomatic number expressions involving "of": "He lives two THIRDS OF a mile from here".

NI is used in a few special idiomatic number phrases: "I have BETWEEN 5 AND 20 dogs".

NJ is used with NF in idiomatic number expressions involving "of": "He lives two thirds OF a MILE from here".

NN connects number words together in series: "FOUR HUNDRED THOUSAND people live here".

NO is used on words which have no normal linkage requirement, but need to be included in the dictionary, such as "um" and "ah".

NR connects fraction words with superlatives: "It is the THIRD BIGGEST city in China".

NS connects singular numbers (one, 1, a) to idiomatic expressions requiring number modifiers: "I saw him ONE WEEK ago".

NT connects "not" to "to": "I told you NOT TO come".

NW is used in idiomatic fraction expressions: "TWO THIRDS of the students were women".

O connects transitive verbs to their objects, direct or indirect: "She SAW ME", "I GAVE HIM the BOOK".

OD is used for verbs like "rise" and "fall" which can take expressions of distance as complements: "It FELL five FEET".

OF connects certain verbs and adjectives to the word "of": "She ACCUSED him OF the crime", "I'm PROUD OF you".

ON connectors the word "on" to dates or days of the week in time expressions: "We saw her again ON TUESDAY".

OT is used for verbs like "last" which can take time expressions as objects: "It LASTED five HOURS".

OX is an object connector, analogous to **SF**, used for special "filler" words like "it" and "there" when used as objects: "That MAKES IT unlikely that she will come".

P connects forms of the verb "be" to various words that can be its complements: prepositions, adjectives, and passive and progressive participles: "He WAS [ANGRY / IN the yard / CHOSEN / RUNNING]".

PF is used in certain questions with "be", when the complement need of "be" is satisfied by a preceding question word: "WHERE are you?", "WHEN will it BE?"

PP connects forms of "have" with past participles: "He HAS GONE".

Q is used in questions. It connects the wall to the auxiliary in simple yes-no questions ("///// DID you go?"); it connects the question word to the auxiliary in where-when-how questions ("WHERE DID you go").

QI connects certain verbs and adjectives to question-words, forming indirect questions: "He WONDERED WHAT she would say".

R connects nouns to relative clauses. In subject-type relatives, it connects to the relative pronoun ("The DOG WHO chased me was black"); in object-type relatives, it connects either to the relative pronoun or to the subject of the relative clause ("The DOG THAT we chased was black", "The DOG WE chased was black").

RS is used in subject-type relative clauses to connect the relative pronoun to the verb: "The dog WHO CHASED me was black".

RW connects the right-wall to the left-wall in cases where the right-wall is not needed for punctuation purposes.

S connects subject nouns to finite verbs: "The DOG CHASED the cat": "The DOG [IS chasing / HAS chased / WILL chase] the cat".

SF is a special connector used to connect "filler" subjects like "it" and "there" to finite verbs: "THERE IS a problem", "IT IS likely that he will go".

SFI connects "filler" subjects like "it" and "there" to verbs in cases with subject-verb inversion: "IS THERE a problem?", "IS IT likely that he will go?"

SI connects subject nouns to finite verbs in cases of subject-verb inversion: "IS JOHN coming?", "Who DID HE see?"

SX connects "I" to special first-person verbs lke "was" and "am".

SXI connects "I" to first-person verbs in cases of s-v inversion.

TA is used to connect adjectives like "late" to month names: "We did it in LATE DECEMBER".

TD connects day-of-the-week words to time expressions like "morning": "We'll do it MONDAY MORNING".

TH connects words that take "that [clause]" complements with the word "that". These include verbs ("She TOLD him THAT..."), nouns ("The IDEA THAT..."), and adjectives ("We are CERTAIN THAT").

TI is used for titles like "president", which can be used in certain cirumstances without a modifier: "AS PRESIDENT of the company, it is my decision".

TM is used to connect month names to day numbers: "It happened on JANUARY 21".

TO connects verbs and adjectives which take infinitival complements to the word "to": "We TRIED TO start the car", "We are EAGER TO do it".

TQ is the modifier connector for time expressions acting as fronted objects: "How MANY YEARS did it last".

TS connects certain verbs that can take subjunctive clauses as complements - "suggest", "require" - to the word that: "We SUGGESTED THAT he go".

TW connects days of the week to dates in time expressions: "The meeting will be on MONDAY, JANUARY 21".

TY is used for certain idiomatic usages of year numbers: "I saw him on January 21 , 1990 ". (In this case it connects the day number to the year number.)

U is a special connector on nouns, which is disjoined with both the modifier and subject-object connectors. It is used in idiomatic expressions like "What KIND_OF DOG did you buy?"

UN connects the words "until" and "since" to certain time phrases like "after [clause]": "You should wait UNTIL AFTER you talk to me".

V connects various verbs to idiomatic expressions that may be non-adjacent: "We TOOK him FOR_GRANTED", "We HELD her RESPONSIBLE".

W connects the subjects of main clauses to the wall, in ordinary declaratives, imperatives, and most questions (except yes-no questions). It also connects coordinating conjunctions to following clauses: "We left BUT SHE stayed".

WN connects the word "when" to time nouns like "year": "The YEAR WHEN we lived in England was wonderful".

WR connects the word "where" to a few verbs like "put" in questions like "WHERE did you PUT it?".

X is used with punctuation, to connect punctuation symbols either to words or to each other. For example, in this case, POODLE connects to commas on either side: "The dog , a POODLE , was black."

Y is used in certain idiomatic time and place expressions, to connect quantity expressions to the head word of the expression: "He left three HOURS AGO", "She lives three MILES FROM the station".

YP connects plural noun forms ending in s to "'" in possessive constructions: "The STUDENTS ' rooms are large".

YS connects nouns to the possessive suffix "'s": "JOHN 'S dog is black".

Z connects the preposition "as" to certain verbs: "AS we EXPECTED, he was late".

Appendix C – The schema of the database

The following schema describes the Cindi database. It is written in the ASCII file format and it contains the description of the relations in the SQL 92 formal language [IBM DB2, 2002]. For the sake of brevity, only three tables are listed here. The full details are included in the NLPQC project, in the file schema.cfg.

```
# Schema file for the NLPQC preprocessor
# Author: Niculae Stratica, Stud ID:5906407
# Case matters (ABC!=abc)
create table resource (
            resource_id   int(10) unsigned NOT NULL AUTO_INCREMENT,
            title               varchar(100) NOT NULL,
            alt_title           varchar(100),
            language_id         int(10) unsigned,
            keyword             varchar(200),
            created_date        date NOT NULL,
            upload_date         date NOT NULL,
            expiry_date         date,
            last_update         date NOT NULL,
            version             varchar(50),
            source              varchar(200),
            annotation          text,
            size                int(10) unsigned,
            resource_format     varchar(20),
            abstract            text NOT NULL,
            filename            varchar(100) NOT NULL,
            contributor_id      varchar(20)  NOT NULL,
            primary key(resource_id));

create table author (
            author_id           int(10)      unsigned      NOT      NULL
AUTO_INCREMENT,
            name                varchar(50)  NOT NULL,
            organization        varchar(100),
            address             varchar(50),
            apt_no              varchar(20),
            city                varchar(30),
            province            varchar(30),
            country             varchar(30),
            p_code              varchar(20),
            e_mail              varchar(50),
```

```
                phone                varchar(20),
                role                 varchar(20)  NOT NULL,
                primary key(author_id));

create table resource_author (
                resource_id          int(10) unsigned NOT NULL,
                author_id            int(10) unsigned NOT NULL,
                primary key(resource_id, author_id));
```

Appendix D – Relational Databases

A database is a collection of related data stored in a computer. One of the most common models is the relational database model. A relation is mathematically based on set theory. The basic component of a set is an object. An object has a series of attributes, where an attribute is a property of the object. A series of these attributes describe an object. The attribute values are organized in a tuple. A set of objects of the same type forms a set of tuples, which in turn form a relation. Two-dimensional tables represent these logical relations universally. Attributes related to the columns of the table and to the tuples form the table rows. These concepts are represented in Figure 41. The table names and the attributes are from Cindi database.

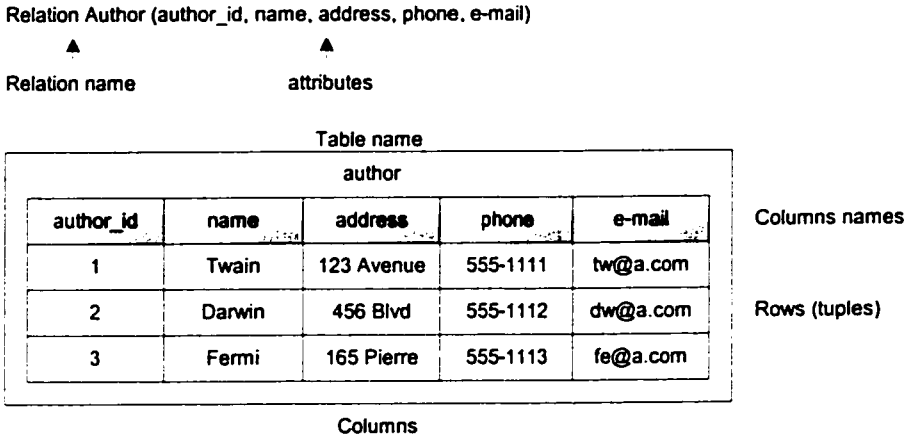

Figure 41. Tables and attributes

A relational database consists of a number of relations or tables. Relational database management systems need to handle data correctly and provide a means of creating, altering, deleting tables and rows in the database. The language used to do these operations is Structured Query Language. SQL was developed in the 70's in IBM's San Jose laboratory where it was used as the relational database language for System-R DBMS [Akeel, 1994]. The outcome of this project made SQL a popular query language among the RDBMS vendors. As a result many relational DBMS came to the market such as Oracle by Relational Software (1979), INGRES from Rational Technology (1981), IDM from Britton-Lee (1982), DG/SQL from Data General Corporation (1984) and Sybase from Sybase Inc. SQL later received certification from American National Standard Institute (ANSI) and from the International Standard Organization (ISO) in 1986. The SQL-86 standard was extended further to create the new SQL-92 standard in 1992. This standard has been used for the NLPQC project. There are a number of advantages in using a standardized relational language such as application portability and cross-system communication. However, SQL is a formal language that is not very easy to learn. NLPQC addresses this issue by hiding it completely from the user.

There are three types of SQL commands: Data definition language, Data manipulation language and Data control language. The SQL commands, or commonly referred to as SQL statements, are issued to the DBMS, which then executes these commands and return the result set. In order to query or retrieve some specific data from the database, the SELECT statement has to be issued. The syntax for SELECT statement is:

```
SELECT column_list FROM table_list
[WHERE conditional_expression]
[GROUP BY group_by_column_lis]
[HAVING conditional_expression]
[ORDER BY order_by_column_list]
```

The expressions in square brackets are optional and the words in capital letters are the SQL keywords. All the lists in the statement are commas separated. The column names and table names are usually case sensitive. The most common SQL statements involve only the SELECT, FROM and WHERE clauses. The following example is related to the schema shown in Figure 42.

show all columns in author table: SELECT * FROM author

show the 'name' column from author table and 'title' column from resource table: SELECT author.name, resource.title FROM author, resource.

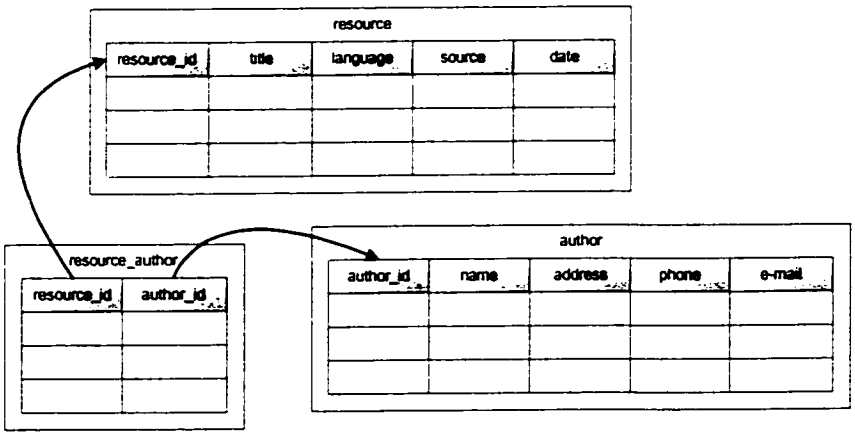

Figure 42. A three-table example in Cindi database

Conclusions

The goal of the semantic analysis is to generate a correct SQL query. Using the SQL to query a database has the advantage of being a universally accepted standard. However, not all users are familiar with it. One of the goals of the NLPQC system is to facilitate the access to a relational database to users that do not know the SQL.

Appendix E – The NLPQC C++ classes

The NLPQC pre-processor reads the database schema through the class Schema. Here is the declaration:

```
class Schema
{
	public:
		// Constructor, Destructor
		Schema();
		~Schema();
		// The class members
		void init_schema ();			// Init schema list with TABLES,
ATTRIBUTES
		void WordNet_schema ();			// Create synonym list for each
element
	private:
		// Class members
		char* skip_spaces (char* pointer);
		char* get_token (char* pointer);
		// Data members
		void_list schema_names;				// store the preprocessed
schema here
		void_list* schema_over;				// store a list of
synonyms for each element
		void store_synonyms (int idx, char* p); // store synonyms in the
schema_over[i]
		int duplicate (int idx, char* p); // check for duplicates
};
```

The string processing has been implemented through the CString class. It resembles the synonym class from the MFC library, but it is not the same. The NLPQC system is designed to be platform independent, while the MFC is not, and this is why is has not been involved in the development of our system.

```
Class CString
{
public:
	/* Various constructors, destructor */
	CString(char *s);
	CString(int s);
	CString(char *s, int start, int howMany);
```

```
        CString(char x, int fill);
        CString(void);
        ~CString(void);
        /* Descriptions for all methods appear at their declaration */
        char   charAt(int index);
        int    compareTo(CString &s1);
        bool   endsWith(char end);
        bool   equalsIgnoreCase(CString &test);
        bool   equalsIgnoreCase(char *test);
        int    indexOf(char ch);
        int    indexOf(char ch, int startIndex);
        int    lastIndexOf(char ch);
        int    lastIndexOf(char ch, int startIndex);
        int    length(void);
        char * operator +(char *s1);
        char * operator +(CString &s1);
        bool   operator ==(CString &test);
        bool   operator ==(char *test);
        void   operator =(char *temp);
        char * replaceOccurence(char oldChar, char newChar);
        void   setCharAt(int index, char setTo);
        bool   startsWith(char start);
        char * substring(int start);
        char * substring(int start, int end);
        char * toChar(void);
        char * toUpperCase(void);
        char * toLowerCase(void);
        char * trim(void);
private:
        int size;           /* The current length of the string */
        char *string;/* Holds the character data of the object */
};
```

The list processing has been implemented through the class Clist. This class only looks like an MFC class but it is not, for the same reasons as for the Cstring.

```
class Clist
{
        struct voidinfo *curr;                  /* only single entry required for
linked list */
        void   *data;
        int    entries;
```

```
        int     curr_entry;
public:
        Clist(void);                        /* Constructor */
        ~Clist(void);                       /* DeConstructor */
        char    add(void *tempdata = NULL);      /* Adds new entry with 'size'
data allocated */
        char    kill(void);                 /* Removes entry from list and frees up
memory */
        void    link(voidinfo *entry);           /* Links linked_list onto another
*/
        voidinfo      *cutfront(void);      /* Cut's off front of list and returns
pointer */
        voidinfo      *cutend(void);        /* Cut's off end of list and returns
pointer */
        char    nuke(void);                 /* Removes all entries */
        void    *get(void);                 /* Gets data from entry */
        char    put(void *info);            /* Puts data to entry */
        char    operator =(void *info);     /* Puts data to entry */
        char    operator--(int x);          /* Uses last entry in list */
        char    operator++(int x);          /* Uses next entry in list */
        char    last(void);                 /* Uses last entry in list */
        char    next(void);                 /* Uses next entry in list */
        char    rw(void);                   /* Goes to start of list */
        char    ff(void);                   /* Goes to end of list */
        int     num(void);                  /* Returns number of entries in list */
        void    count(void);                /* Restores info in list */
        void * addr(void);                  /* Returns pointer to data */
        char    use(int entry);             /* Goes to specified entry in list */
        char    find(void *fdata);          /*  Makes the list with data fdata
current */
};
```

Bibliography

Abrahams, P., W. et al. (1966), *The LISP 2 Programming Language and System*, Proceedings of FJCC 29:661-676, AFIPS, Fall 1966.

Akeel, D. (1994), *Structured query language (SQL) A practical Introduction*. NCC Blackwell Ltd, ISBN: 1855543575, 1994.

Brill, E. (1995), *Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging*. Computational Linguistics 21(4), pages 543-565.

Codd, E., Klimbie, J.,W. (1974), *Seven Steps to Rendezvous with the Casual User*, in Data Base Management, N-H 1974, pp.179-199.

Desai, B.C. (1997), *Supporting Discovery in Virtual Libraries*, Journal of the American Society of Information Science, 48-3, pp. 190-204.

Desai, B.C. (1999), *Concordia INdexing and DIscovery System*, paper presented at the First Canadian Database Research Workshop, UQAM Montreal, August 1999.

Desai, B.C., Shinghal, R., Shyan, N., Zhou, Y. (1999), *CINDI: A System for Cataloguing, Searching, and Annotating Electronic Documents in Digital Libraries*, Proceedings of ISMIS'99, Springer-Verlag, Warsaw, Poland, June, pages 154-162.

Desai, B.C., Shinghal, R., Shyan, N. and Zhou, Y. (1999), *CINDI: A System for Cataloguing, Searching, and Annotating Electronic Documents in Digital Libraries*, Library Trends, Summer 1999, 48(1), pages 209-233.

Desai, B.C., Haddad, S.S. and Ali, A. (2000), *Automatic Semantic Header Generator*, Proceedings of ISMIS'2000, Springer-Verlag, Charlotte, NC, pages 444-452.

Dix, J. (2002) *EasyAsk: How it works*, Network World, March 11, 2002.

Elf (2002), *English Front End*, http://www.elf-software.com, site visited in June 2002.

Freitag D. (1998), "Information Extraction From HTML: Application of a General Learning Approach.", In the Proceedings of the *15th National Conference on Artificial Intelligence (AAAI-98)*.

Gal, A., Minker, J. (1985), *A Natural Language Database Interface that Provides Cooperative Answers*, in Artificial Intelligence Applications, C.R. Weisbin (ed.), IEEE Press, Washington.

Grinberg, D., Lafferty, J., Sleator, D. (1995), *A robust parsing algorithm for link grammars*, Carnegie Mellon University Computer Science technical report CMU-CS-95-125, and Proceedings of the Fourth International Workshop on Parsing Technologies, Prague, September, 1995.

Grosz, B. et al. (1983), *A transportable natural language interface system* in Proceedings of the Conference on Applied Natural Language Processing (Santa Monica, Calif., Feb.), Association for Computational Linguistics, 39-45.

Harabagiu S. et al. (2001), *FALCON: Boosting Knowledge for Answer Engines*, NIST Special Publication 500-249: The Ninth Text Retrieval Conference, Gaithersburg, Maryland, USA (TREC 9), pages 479--488.

Hendrix, G., Sacerdoti, E., Sagalowicz, D., Slocum, J. (1978), *Developing a natural language interface to complex data*. A CM Transactions on Database Systems, 3(2):105-147, June 1978.

Herbert, Stoyan (2002), *Early LISP History (1956-1959)*, University of Erlangen-Nurnberg Lehrstuhl fur Kunstiche Intelligenz Am Weichselgarten 7, D-91058 Erlangen Germany.

Hermjakob, U. and R.J.Mooney (1997), *Learning Parse and Translation Decisions from Examples with Rich Context*. In 35th Proceedings of the Conference of the Association for Computational Linguistics (ACL), 482–489.

Hermjakob, U. (2000), *Rapid Parser Development: A Machine Learning Approach for Korean*, Proceedings of the North American chapter of the Association for Computational Linguistics (NAACL-2000).

Hinrichs, E.W., Ayuso, D.M., and Scha, R. (1990), *The Syntax and Semantics of the JANUS Semantic Interpretation Language*, in Research and Development in Natural Language Understanding as Part of the Strategic Computing Program, Annual Technical Report, BBN Laboratories, Report No. 6522, 1987, pp. 27-31.

Hovy, E. et al. (2001), *Question Answering in Webclopedia*, NIST Special Publication 500-249: The Proceedings of The Ninth Text REtrieval Conference, Gaithersburg, Maryland, USA (TREC 9).

IBM DB2 (2002), *SQL Reference*, http://www-3.ibm.com/software/data/datajoiner/booksv2/djxk5m02.htm#ToC, site visited in August 2002.

Katz, B. (1997), *From Sentence Processing to Information Access on the World Wide Web*, AAAI Spring Symposium on Natural Language Processing for the World Wide Web, Stanford University, Stanford CA.

McCarthy, J. (1959), *LISP Programmers Manual, Handwritten Draft*, MIT AI Lab., Cambridge Oct. 1959.

McCarthy, J. (1960), *Recursive Functions of Symbolic Expressions and their Computation by Machine*, Communications ACM, 3(1960)3, 184-195.

Microsoft Corporation (2000), *Microsoft Servers - English Query*, http://www.microsoft.com/sql/productinfo/english.htm, site visited in August 2002.

Microsoft Corporation (1998), *Add Natural Language Search Capabilities to Your Site with English Query, MIND*, http://msdn.microsoft.com/library/periodic/period98/equery.htm, site visited in August 2002.

Miller, A.G. et al. (1995), *WordNet – A lexical database for the English language*, Communications of the ACM, 38 (1), November, pp. 39-41, ACM Press, New York, ISSN:0001-0782, 1995. Software available at http://www.cogsci.princeton.edu/~wn/, site visited in August 2002.

Plamondon, L., Kosseim, L. (2002), *QUANTUM: A Function-Based Question Answering System*, Proceedings of The Fifteenth Canadian Conference on Artificial Intelligence, May 2002.

Prager, J., et al. (2001), *Use of WordNet Hypernyms for Answering What-Is Questions*, Proceedings of The Eighth Text REtrieval Conference, Gaithersburg, Maryland, USA (TREC 8, 2000).

Rosch, E. et al. (1976), *Basic Objects in Natural Categories*, Cognitive Psychology 8, pages 382-439.

Robertson, S.E. and Walker. S., (1998) *Okapi/Keenbow at TREC-8* in Proceedings of The Eighth Text Retrieval Conference (TREC-8), pages 151-162, Gaithersburg, Maryland.

Ruwanpura, S. (2000), *SQ-HAL: Natural Language to SQL Translator*, http://www.csse.monash.edu.au/hons/projects/2000/Supun.Ruwanpura, Monash University, site visited in May 2002.

Scott S., Gaizauskas R. (2001), *QA-LaSIE: A Natural Language QA System*, Advances in Artificial Intelligence, 14th Biennial Conference of the Canadian Society for Computational Studies of Intelligence, AI 2001, Ottawa, Canada, June 7-9, 2001, Proceedings, 172-182.

Sleator, D., Temperley, D. (1991), *Parsing English with a Link Grammar*, Carnegie Mellon University Computer Science technical report CMU-CS-91-196, October 1991. Software available at http://www.link.cs.cmu.edu/link/.

Stratica N., Kosseim L., Bipin C.Desai (2002), *A Natural Language Processor for Querying Cindi*, to appear in Proceedings of the 2002 SSGRR Conference, l'Aquila, Italy.

Teigen, J. and Vetland, V. (1988), *Syntax analysis of norwegian language*. Technical report, The Norwegian Institute of Technology.

Teitelman, W. (1966), *FLIP, A Format List Processor*, Memo MAC-M-263, MIT.

TREC-9 (2000), *Proceedings of the Ninth Text Retrieval Conference*, November, Gaithersburg, Maryland, USA.

TREC-X (2001), *Proceedings of theTenth Text Retrieval Conference*, November, Gaithersburg, Maryland, USA, available at http://trec.nist.gov/pubs/trec10/t10_proceedings.html, site visited in 2002.

Warren, D.,H.,D. and Pereira, F., C., N. (1982), *An efficient and easily adaptable system for interpreting natural language queries* in Computational Linguistics, 8(3-4).

Watson, M. (1997), *Creating Javabeans Components for Distributed Applications*, ISBN: 1558604766 Publisher: Morgan Kaufmann Publishers, September 1997.

Woods, W.A., Kaplan, R.M. and Webber, B.N. (1972), *The Lunar Sciences Natural Language Information System: Final Report*. BBN Report 2378, Bolt Beranek and Newman Inc., Cambridge, Massachusetts.

WSA World Site Atlas (2002), http://www.sitesatlas.com/, site visited in August 2002.