PRAJÑĀ-Journal of Pure and Applied Sciences, Vol. 19: 41 - 44 (2011)
ISSN 0975 - 2595

NATURAL LANGUAGE INTERFACE FOR STUDENT INFORMATION SYSTEM (NLSIS)

Amisha Shingala*, Rinku Chavda and Paresh Virparia[1]

**Department of MCA, SVIT, Vasad, Gujarat, India*
[1]G. H. Patel Computer Science Department, Sardar Patel University, Vallabh Vidyanagar, Gujarat, India

ABSTRACT

Any computer system that interacts with user has a high utility value and ease. Natural Language Interface is a concept for making computer interface with a user who wants to retrieve the information from a computer database with its own language rather than learning a specialized language. The challenges in Natural language arise due to difficulty in correct interpretation, disambiguation and context resolution.

We present our work in designing and implementing the Natural Language Interface for querying Student Information System. It semantically parses the natural language question and built corresponding structured query from the database. The data which is to be retrieved can be in the form of a spreadsheet or database. The system is developed in Java using JDBC for Student Information System.

Keywords: *Language, Database Interface, Structured Query Language, Data Retrieval*

INTRODUCTION

The computer era has began from last few years and the phase of bringing awareness into different walks of society regarding use and benefits of computer increased successfully. Not only industry but educational institute, service sectors, manufacturing sectors etc are using computers to store, process and update the information. The huge amount of data are stored in repository called database and in order to query or retrieve information from a database by general public, a Natural Language (English) will provide correct and precise information without knowing the depth of SQL query language. The idea of using Natural language with database prompted the development of new type of processing method called Natural Language Interface to Database.

Some of the earlier attempts in providing Natural Language Interface are given below:

1) MASQUE/SQL by Androutsopoulos et al [4] and Anuxeree, P [5] can answer English question by generating SQL code.
2) LUNAR [13] that answered questions about rock samples back from the moon.
3) LIFER/LADER describe by Hendrix [9] was designed as a natural language interface to database about US Navy ships.
4) Warren el al's CHAT-80[12] and Auxerre et al's transform written English questions into prolog queries which are executed against Prolog database.

Information related to overview of various Natural Language Interfaces and the difference between Natural Language Interface and Question Answering could be found in reference [1].The approach used in most of the recent systems is to handle the entire query as one entity for transformation to SQL either through conversion to intermediate representation or directly. We had overcome many limitations which were given by earlier researchers [11].

MODEL AND METHODOLOGY OF NATURAL LANGUAGE STUDENT INFORMATION SYSTEM (NLSIS)

The following goals were set for the proposed NLSIS:

- ***Fault Tolerance***– An algorithm [2] is developed for tolerating spelling errors, which uses degree of phonetic match and degree of spelling match to correct misspelled words. Since the semantics of the query for purpose of conversion to SQL statement within a limited domain does not use the grammatical structure of sentence formation, it is implicitly tolerated. However, use of grossly, inappropriate words (particularly prepositions e.g. using " by" in place of "of", etc and interrogative pronouns) may lead to wrong or failed interpretation of the user query.
- ***Better Context Resolution and Disambiguation*** – We first partition the query into intermediate language and then to SQL clause. A part of query with associate SQL along with domain-specific lexicon leads to an improved context resolution and disambiguation.
- ***Multiple database tool support:*** A tool is developed which can covert data from spreadsheet to database and database to relational database.
- ***History Log Maintenance:*** The query once asked, was stored in separate log which can be help in retrieving the same information again without processing whole algorithm.

Domain-Specific Ontology

Figure A present below ontology of Student Information System (SIS) domain.

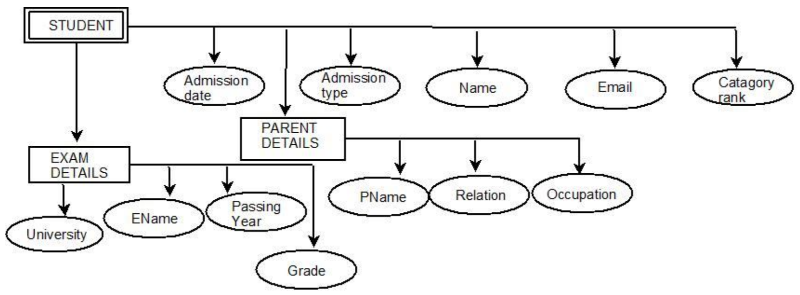


Figure A: Domain ontology for Student Information System

*Corresponding author : am_bt@yahoo.com

A. The **Linguistic model** consists of Lexical Analysis, Parser and Semantic representation as shown in figure B which is explained below:

Lexical Analysis: Here the natural language sentence is divided into smaller fragments called tokens. The elements in natural language query are words or special characters. This process is performed by following function:

i. Token analyzing function: It is used to generate the token which is treated as a single unit.

ii. Spell checker function: It makes sure that the user inputted query with correct word.

iii. Ambiguity reduction: It reduces ambiguity of sentence and simplified the task of parser by substituting multiple words or symbols with base word. E.g. Comma considered as AND, co-occurring word as single word etc.Also, we try to find out that the expected answer would be named entity or not. Figure C represent hierarchy of named entity [7]

Architecture of system

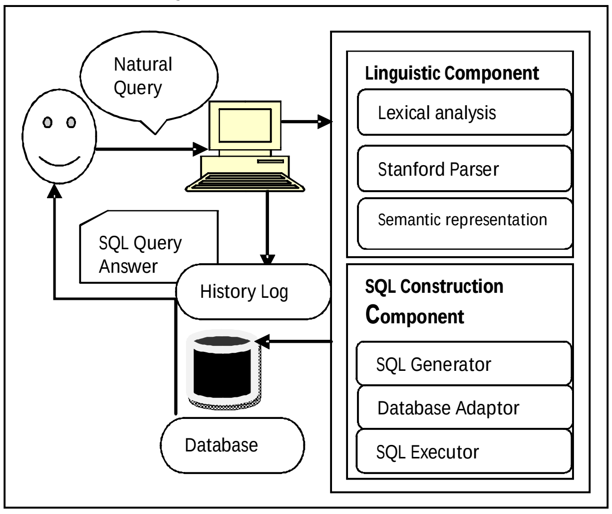


Figure B: Architecture of NLSIS

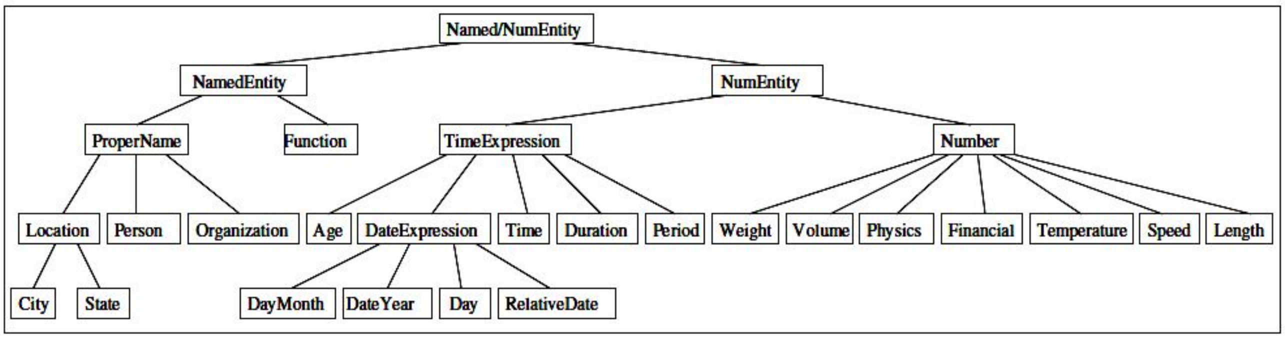


Figure C : Hierarchy of Named Entity

Stanford Parser: The Stanford Parser is a probabilistic parser which uses the knowledge of language gained from hand-parsed sentences to try to produce the *most likely* analysis of new sentences as shown in figure D. This package is a Java implementation of probabilistic natural language parsers.

The Stanford dependencies provide a representation of grammatical relations between words in a sentence for any user who wants to extract textual relationships [3]. The dependency obtained from Stanford parser can be mapped directly to graphical representation in which words in a sentence are nodes in the graph and grammatical relationships are edge labels [6]. In our system, we use the POS tagger and Typed Dependency for an inputted query or sentences.

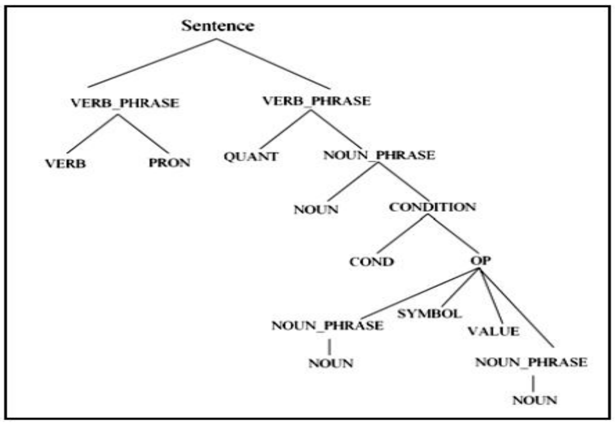


Figure D: Hierarchal structure of Stanford Parser

Semantic Representation: We had used two types of lexicon semantic representation in the form of grammar for terminal words, non terminal words and terminal symbols.

Table 1: Example of Terminal Words

Terminal Words	*Semantics/Base Word*
girls, girl, female, madam	Female
Percentage above 70	Distinction

Table 2: Example of NonTerminal Words

NonTerminal Words	*Semantics/Base Word*
Greater than or equal to	>=
Less, lower or lesser	<

Table 3: Example of Terminal Symbols

Terminal Symbols	*Semantics/Base Word*
Greater, larger, bigger, biggest	Greater
Equal to, Like, equals, same	Equal

B. SQL Constructing Component

This component consists of SQL generator, Database Adaptor and SQL execution.

SQL Generator: The task is to map element of the Natural Language sentence into an actual element of SQL. We consider only SELECT as a statement to view the data in various forms. We used BNF form of SQL grammar [10]

<select query> ::= <select> [UNION [ALL | DISTINCT] <select query>]

<select> ::= SELECT [ALL | DISTINCT] <select list> <from clause>

[<where clause>]

<select list> ::= '*'| <column element> [{',' <column element> }...]

<column element> ::= <column>| <aggregate function> '(' <column> ')'

<from clause> ::= FROM <table reference>[{','<table reference>}...]

<table reference> ::= <table schema> [[AS] <correlation name>]| <table reference>

[INNER | LEFT | RIGHT] JOIN <table reference> ON <search conditions>

<where clause> ::= WHERE <search conditions>

The <search condition> is a logical predicate composed of logical conditions with AND and OR operator. The <aggregate functions> like max, min, sum, average and count transforms set of rows into scalar statement. The <condition> is a expression in form $X_R Y$ where X and Y are the set of values representing column, aggregate function, constant or NULL and R can be operated like {<,>,<=, >=, !=,==}.

Database Adaptor: The data repository can be in the form of a spreadsheet or database. Our algorithm converts these data into MySQL database and retrieve the answer from MySQL database tool.

SQL Executor: The task is to map the SQL generated query to Database Adaptor and retrieve the relevant information or answer by connecting to the appropriate database tool.

3. Algorithm for NLSIS

```
(i)    Read statement or user query S
(ii)   Search S from History Log H
       If found, call query generate algorithm and retrieve the query Q
       Execute the Query Q and Display the answer Ans.
(iii)  If S not successful, perform all the steps from step iv.
(iv)   For each word $W_i$ from S do
       If $W_i$ base word then
       Add Wi to symbol table ST
       End if
       Endfor
(v)    For each Wi from ST do
       Add Wi to Parse tree
        End for
(vi)   Display POS tagger and Typed dependency and get relationship between words PDR
(vii)  For each PDR do
       If metadata = PDR then
       Add table name, attribute and condition to OUT
       Endif
       Endfor
(viii) For each OUT do
       Generate the SQL query Q1
       Call Data conversion algorithm
       Execute SQL query Q
       End for
(ix)   Display SQL query Ans.
```

RESULTS and DISCUSSION

The software has been subjected to test with a number of volunteers phrasing the queries differently. In around 70% cases, the system could correctly interpret and process the query. Currently, the system fails in some cases to handle very complex queries coupled with fuzzy terms. We had paraphrased different queries in the form of simple query, condition query, order by query and join query. Figure E shown implemented version of the system.

Table 4: List of normal Queries.

Input or Query	*SQL Query generated*
Display details of all students	Select * from students;
Who are our students ? show me list of all students ?	Select * from students;
Show me all branches	Select * from branch
Display various branches	Select bnm from branch

Table 5: List of Conditional Queries

Input Sentence or Query	*SQL query generated*
List of all female students or Display details of girls students List of students who live in Baroda	Select studnm from student where gender = 'F' Select studnm from student where city = 'VADODARA '
Details of Monalisa?	Select * from student where studnm = 'monalisa'
Give me hsc% of aryan or display 12th percentage of Aryan	Select hscper from student where studnm = '%aryan%'

Table 6: List of Aggregate function Queries

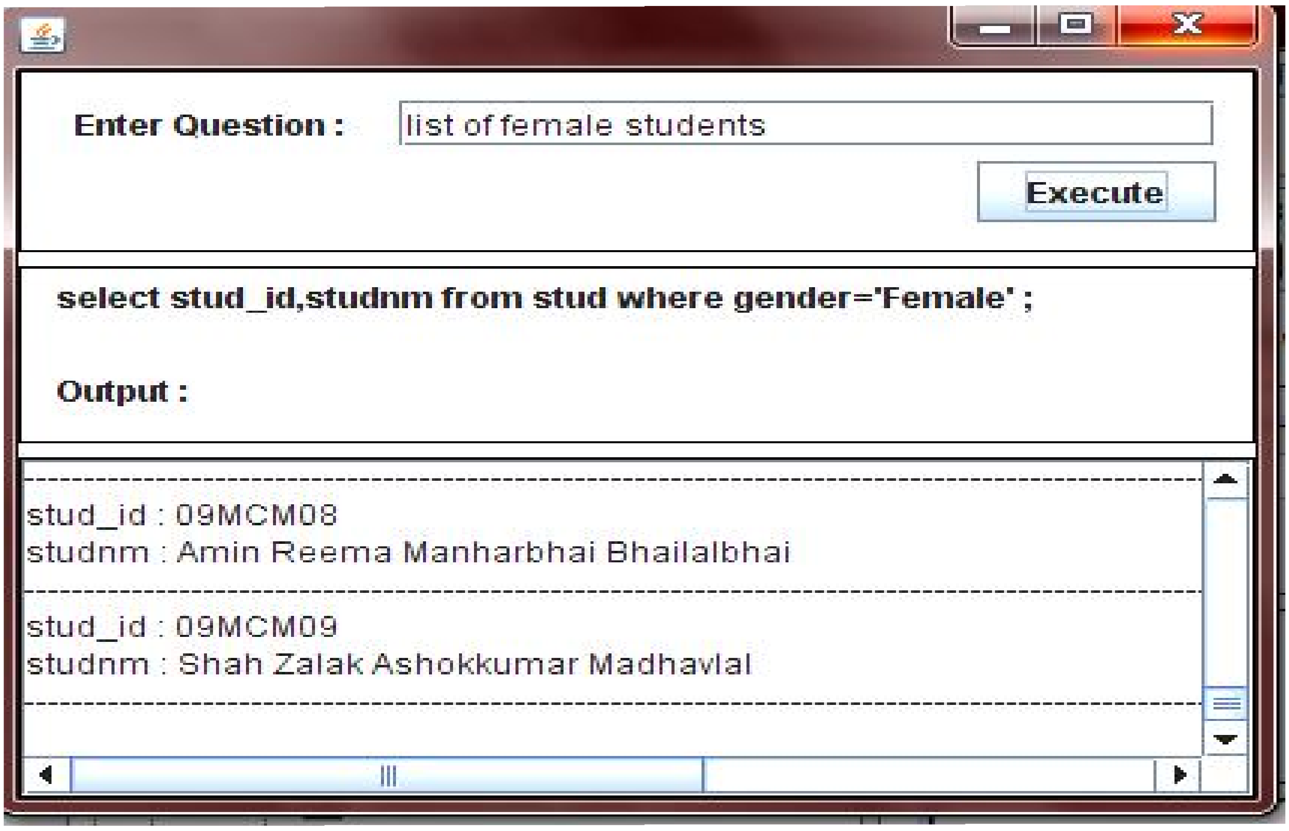
Input Sentence or Query	*SQL query generated*
How many students are in semester 6	Select count(*) from students where sem =6;
List youngest student in sem 1	Select min(age) from students where sem =1;
How many students we have ?	Select count(*) from students

Table 7: List of Order by Queries

Input Sentence or Query	*SQL query generated*
Display student record in ascending order of their rollnos	Select * from students order by studid;
Display male students in descending order of their age	Select studnm, age from students where gender = 'm' order by age

Table 8: List of Simple join Queries

Input Sentence or Query	*SQL query generated*
Display branch in which rima study	Select bnm from branch natural join student



list of female students

```
select stud_id,studnm from stud where gender='Female' ;
```

```
----------------------------------------------------------------------------------------------
stud_id : 09MCM08
studnm : Amin Reema Manharbhai Bhailalbhai
----------------------------------------------------------------------------------------------
stud_id : 09MCM09
studnm : Shah Zalak Ashokkumar Madhavlal
----------------------------------------------------------------------------------------------
```

Figure E: A sample screen shot of the system

CONCLUSION AND FUTURE WORK

The degree of correct interpretation and processing of Natural Language Queries relies on the extent of exhaustiveness of domain-specific lexicon. Despite of some limitations, the proposed system provides a robust tool to handle Natural Language Queries allowing fault-tolerance to some degree. The work can be further extended by providing multilingual (Gujarati or Hindi language) query to SQL conversion.

REFERENCES

[1] Amisha Shingala & Paresh Virparia, (2011) A Survey of Natural Language Interface, International Journal of RESEARCH@ICT: International Journal of Information and Computing Technology, Volume 2 , ISSN: 0976 – 5999.

[2] *Amisha Shingala & Sanjay Vij (2007)* Manage and Access Information and Knowledge through Natural Language Queries, Fifth AIMS International conference.

[3] Anjali Ganesh Jivani, Amisha Shingala & Paresh Virparia (2011) The Multi-Liaison Algorithm, International Journal of Advanced Computer Science and Applications, Volume 2 Issue, ISSN 2156-5570 (Online)

[4] Androutsopoulos, 1, Ritchie & Thanisch P (1993), MASQUE/SQL- An efficient and portable Natural Language Query Interface for Relational Database, Proc. of Sixth International Conference on Industrial & Engineering Applications of AI & Expert System, Edinburgh.

[5] Anuxerre, P & Inter R (1993) MASQUE Modular Answering System for Queries in English - User's Manual, AI Applications Institute, Univ. of Edinburgh.

[6] Faraj A. El-Mouadib, Zakaria S. Zubi, Ahmed A. Almagrous, and Irdess S. El-Feghi (2009) Generic Interactive Natural Language Interface to Databases (GINLIDB), International Journal of Computers, Issue 3, Volume 3.

[7] Ferret, B. Grau, M. Hurault-Plantet, G. Illouz, How NLP can improve Question Answering, Limsi-cnrs, BP Orsay Cedex

[8] F.Siasar djahantighi, M.Norouzifard, S.H.Davarpanah, M.H.Shenassa (2008) Using Natural language processing in order to create SQL queries, Proceedings of the International Conference on Computer and Communication Engineering, Kuala Lumpur, Malaysia

[9] Hendrix, C.G; Sacerdoti, E.D; Sangalowicz, D; Slocum J (1978) Developing a natural language interface to complex data ", ACM Trans on Database Systems, 3(2), pp. 105-147.

[10] M^{a} José Suárez-Cabal (2009) Structural Coverage Criteria for Testing SQL Queries, *Journal of Universal Computer Science, vol. 15.*

[11] Niculae Stratica (2002) *Natural language interface for querying Cindi*, Thesis, Master of Computer Science, Concordia university, Quebec, Canada.

[12] Warren D and F. Pereira (1982) An Efficient Easily Adaptable System for Interpreting Natural Language Queries, Computational Linguistics, pp.110-122

[13] Woods, W.A. (1973) Progress in Natural Language Understanding: An Application to Lunar Geology, AFIPS Conf. Proc. 42, pp. 141-450.