Querying Database using a universal Natural Language Interface Based on Machine Learning

Hanane Bais[1], Mustapha Machkour[2], Lahcen Koutti[3]
team of Engineering of Information Systems, Information Systems and Vision Laboratory
Faculty of Sciences, Ibn Zohr University
Agadir, Morocco
[1]baishanan@gmail.com, [2]machkour@hotmail.com, [3]lkoutti@yahoo.fr

***Abstract*— Extracting information from a database system becomes a primary obligation. More and more we are forced to recognize the importance of providing easy access to information stored in a database system. However existing tools that allow users to query database using database query languages such as SQL (Structured Query Language) are difficult for non-experts users. Wherefore asking questions to databases in natural language is a very simple method that can provide powerful improvements to the use of data stored in databases.**

This paper presents the Architecture and the implementation of a generic natural language interface based on machine learning approach for a relational database. The advantage of this interface is that it functions independently of the database domain and automatically improves through experience its knowledge base.

Keywords—Relational Databases; Natural Language Processing (NLP); Intermediate XML Logical Query (IXLQ); Extended Context Free Grammar (ECFG); Machine Learning

I. INTRODUCTION

Over the last fifteen years, the Natural Language Processing (NLP) has become one of the most active areas in human-computer interaction. The purpose of NLP is to enable communication between people and computers without needing the memorization of commands and complex procedures [1]. In that the user doesn't require programming language skills. One of the classic problems in the field of NLP that has recently attracted the attention of the research community is the Natural Language Database Interface (NLDBI). The objective of NLDBI is to extract information from Database using natural language [2]. In this sense the user of database does not need to have expertise in database language to access the data, but only the natural language is enough. Traditionally, people are used to working with a form; but their anticipations strongly dependent on the capabilities of this form. Wherefore the using of NLDBI offers to a large number of users of database system a simple, uniform and unlimited access to data without learning any database query languages.

The remainder of the paper is organized as follows. In section 2 we give an overview of existing work showing their advantages and limitations. Section 3 presents a brief description of our proposed system. Section 4 details the architecture of our system. Section 5 shows some results of this work. Finally, section 6 presents the conclusions and possible extensions of this work.

II. RELATED WORK

The earliest research in the field of NLDBI has been started since1960s [3, 4]. Since this date several systems have been created. BASEBALL [4] and LUNAR [5, 2] were the first usable NLDBI, which appeared in the late sixties. The BASEBALL system was created to answer questions about American baseball games. LUNAR involved a system that answered questions about rock brought back from the moon. However, these systems were Non-Reconfigurable systems, since they were designed for a particular domain and thus could not be easily updated to interface other deferent database domains. By late 1970s, various research prototypes were implemented, like PLANES [6, 18] and LADDER [7, 18]. Some of these systems use semantic grammars inherent to database domain. So it is necessary to develop other grammar to interface other application domains. TEAM [21, 22], CHAT-80 [8] and DIALOGIC [9] systems were developed by the mid-1980s. CHAT-80 formed the basis of several other experimental systems such as MASQUE [10] and MASQUE/SQL [11]. Recently, many NLDBI have been developed such as PRECISE [12, 2] and NALIX (Natural Language Interface for an XML Database) [13]. PRECISE system is developed at the University of Washington to interface relational database systems. Precise introduces the initiative of semantically tractable question which are question that can translate to unique semantic interpretation using some lexicons and semantic constraint. NALIX is a NLIDB system developed at the University of Michigan. This system interfaces XML (extensible markup language) database with Schema-Free XQuery as the database query language.

In this paper we cope with tree issues present in many NLIDB. The first one is that some NLIDBs are based on intermediate representation language [14, 11, 16]. In these the Natural Language Query (NLQ) is translated into a logical query and this latter is translated into database query language. But not all forms of logical query are independent of database language, for that many NLIDB are designed for a particular database model (Relational, Relational-Object, Object, XML, etc.). The second problem exists in systems that use syntactic

978-1-4673-7689-1/16/$31.00 ©2016 IEEE

parsing in which the syntax rules describing the terminal symbols are domain-dependent [15, 14, 10].So to be ported to another domain these rules must be manually changed. The last problem is that many NLDBI don't improve the waiting time for translating the questions already processed. In the next sections we propose some solutions to resolve these problems.

III. PROPOSED SYSTEM

The proposed system is based on intermediate representation language approach. First, the NLQ is submitted to many analysis operations (morphological, syntactic and semantic analysis). At the end of this procedure, we obtain an Intermediate XML Logical Query (IXLQ). This expression corresponds to the interpretation of NLQ. Express the logical query in XML form allows the system to be Independent of the database language, content and model (Relational, Relational-Object, Object, XML, etc.). Then logical query translated into database query expression. In our case our system works with Relational and xml databases.

The system works independently of the database domain and automatically improves through experience its knowledge base. The system has this capacity because it use an Auto Generator of Syntactic Rules (AGSR). We will explain the operation of this generator in the next sections

To improve the waiting time for translating questions already processed, we add a Module of Natural Language Query Definitions (MNLQD) This contains a set of classes of IXLQ. Each class represents a logical interpretation of many NLQ. The integration of this module in our system is to improve the waiting time required for translating the questions already processed.

IV. SYSTEM ARCHITECTURE

The main advantage of the system architecture is that it divided into three great modules: the linguistic component module, the database knowledge component module and MNLQD . The first module controls the linguistic aspect, where the natural language Query is submitted a many analysis operations At the end of this procedure, we obtain IXLQ. This expression corresponds to the XML interpretation of the initial natural language Query.

The second module is used to translate IXLQ to database query language expression. This latter is sent to the database jet for producing the answer. In the next we explain in depth how these modules work. By separating linguistic and database knowledge components the system has the ability to be ported to different databases [16].

The third is MNLQD which use to improve waiting time necessary for translate the queries already processed. in the next schema we represent the architecture of our system:

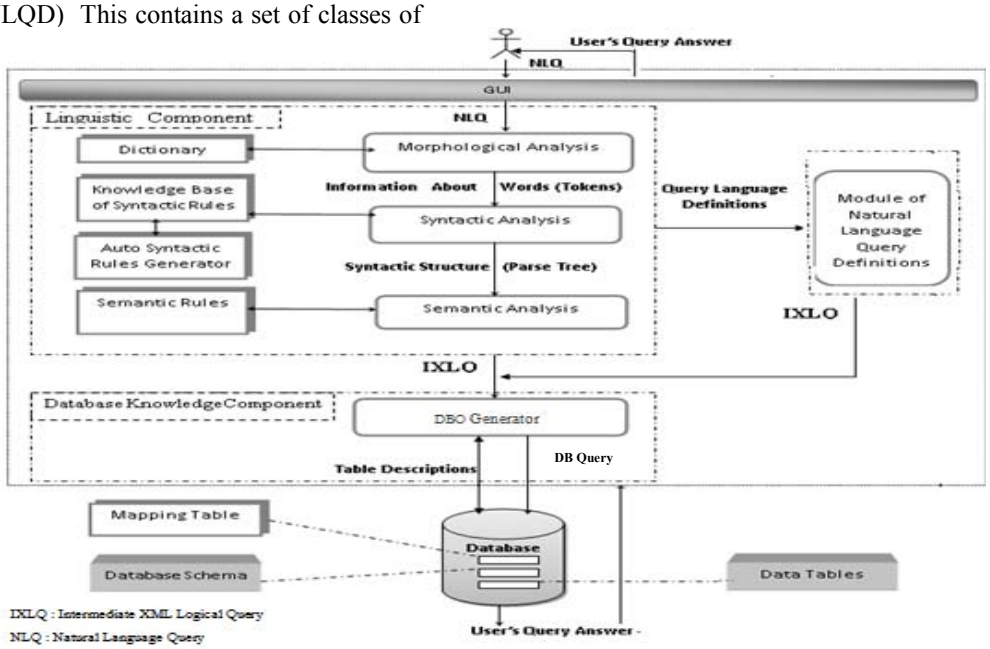

Fig. 1. System Architecture

A. Linguistic component

The linguistic component does three analyses: morphological, syntactic and semantic explained as follows:

1) Morphological analysis: Divides the input sentence in primitive units called tokens. And return information about each token.

a) Token analyzing: this function is used to split the input sentence in primitive units called tokens, which is considered as a single logical unit.

b) Spelling checker: this function ensures that each token is in the system dictionary, if this is not the case, then spell checking is performed or a new word is added to the system vocabulary.

c) Ambiguity reduction: this function minimizes the ambiguity in a sentence to simplify the task of the next analysis by replacing several words or symbols with canonical internal words.

d) Tagger: this function determines the grammatical category of each token.

e) Morpheme: this function is used to determine the morpheme of each token.

2) Syntactic analysis: Returns the parser tree [20] that shows how words relate to each other. That is done by applying a set of syntactic rules described by an Extended Context Free Grammar (ECFG) [17]. In our system the syntactic rules related to non-terminal symbols are domain-independent, these rules are definite previously in the system knowledge base. Whereas the syntactic rules related to terminal symbols are domain-dependent. These latter are generated by ASRG based on machine learning approach [19]. the aim of ASRG is to check whether all the syntactic rules necessary to parse the user query exist in the system knowledge base. If not, it detects automatically the necessary syntactic rules; it creates and adds them to the knowledge base. This part of our system will help to adapt its knowledge base with user requests and therefore it can function regardless of the database domain, so it will be generic system. The following figure displays an example of a parse tree representing the natural language query 'Show me the address of client whose name is "AHMED".

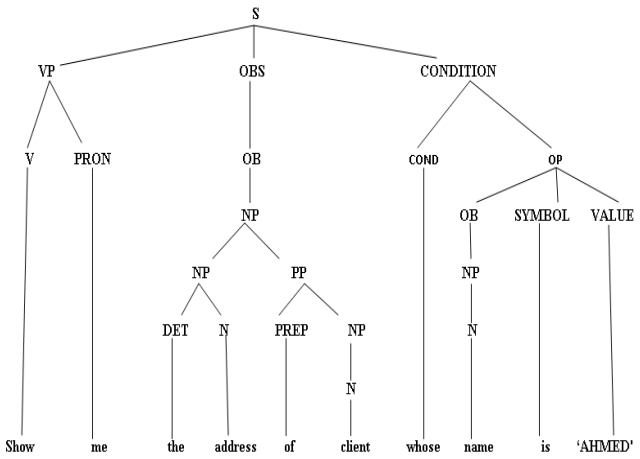

Fig. 2. Parse tree

3) Semantic Analysis: The goal of the semantic analysis is to assign a logical meaning to the parse tree using some semantic rules. Each semantic rule has a corresponding syntactic rule.

TABLE I. SEMANTIC RULES WITH THEIR CORRESPONDING SYNTAX RULES

Semantic rule	corresponding syntactic rule
< attribute 1> cc < attribute 2 > pre <objec1t> cc <object2>	(NP CONJ NP)PREP (NP CONJ NP)
< attribute > pre <object1> cc < object2 >	NP PREP(NP CONJ NP)
<object> cond<attribute> pre <object> symbol <attribute value>	NP CND ((NP PREP NP) SYMBOL VALUE)

For that reason this translation process is called rule-by-rule style [13, 15]. Those rules used to translate the parse tree into an IXLQ. The following XML logical query displays the logical query associated to the parse tree of the natural language query 'Show me the address of client whose name is "AHMED" ' (see Figure 3):

```
<REQUEST>
  <SELECT>
    <OBJECT>
      <NAME> client </NAME>
      <ATTRIBUT>
        <NAME> address </NAME>
      </ATTRIBUT>
    </OBJECT >
  </SELECT>
  <COND>
    <OBJECT>
     <ATTRIBUT>
      <NAME> name </NAME>
     </ATTRIBUT>
    </OBJECT >
    <SYMBOL> is </SYMBOL>
    <VALUE> AHMED </VALUE>
  </COND>
</REQUEST>
```

B. Database knowledge component

This component has two tasks. First, it translates the logical query (i.e. IXLQ) into database query, by mapping each element of the logical query to its corresponding clause in the database query. And then, displays the answers returned by the DBMS in tabular form.

Our system generates SQL query for relational database and Xpath for xml database

The generation of database query consists of four phases. Each phase manipulates only one specific part of the database query. The concatenation of the results of the four phases constructs the final database query.

The first phase deals with the part of the logical query that corresponds to the names of attribute for building the select clause. The second phase, builds the from clause by selecting the portion of the logical query that is mapped the table name or a group of table names. The third phase extracts the conditions of selection from of the logical query to construct the condition clause. At the fourth phase, we

select the portions of the logical query that corresponds to the order of presenting the result of database query (i.e. order by clause). Each phase is followed by a test to verify if the name of tables and attributes, extracted from logical query, are valid or exist in dictionary of database. If it is not the case, the system uses a domain specific dictionary (mapping table) which stores the synonyms of table and attribute names. The

mapping table helps the user to write the same query with different natural language sentences.

V. SYSTEM RESULTS

Using our system the user query can be written in different ways (often more than eighty ones) using different query verbs such as: Show, Find, Tell, Search, Give, List and Display. It is also possible to put a natural language question without query verb. He interface represented in figure 5 shows the translation of the NLQ: “give me the names of clients whose ages > 25” into a SQL and Xpath queries:

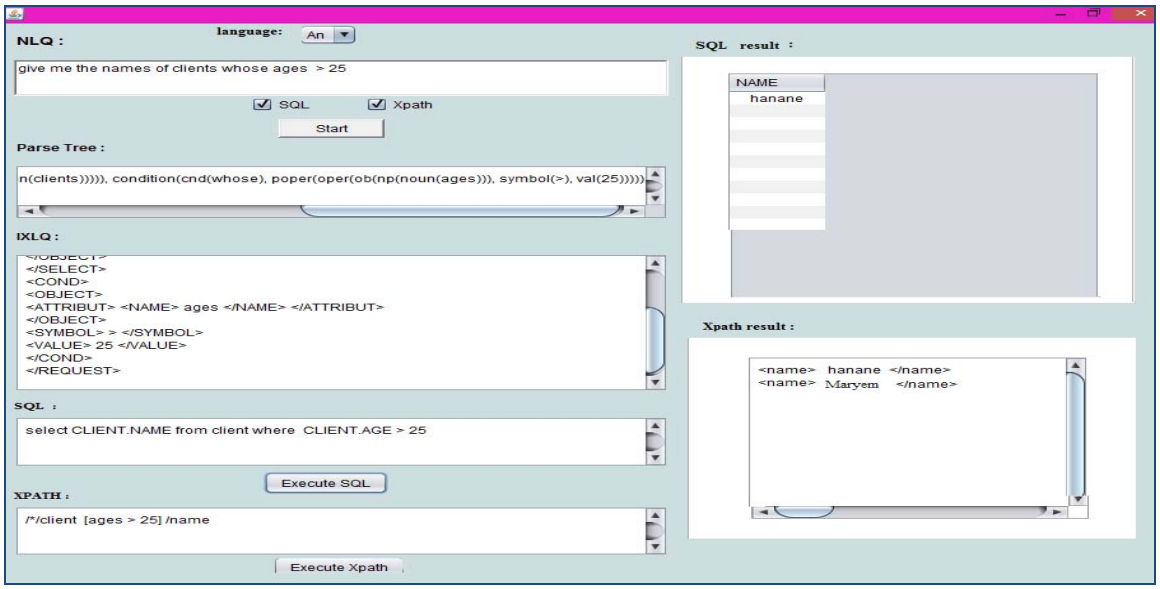

Fig. 3. System interface

The following tables show a list of variety of NLQ that are successfully translated and executed by our system.

TABLE II. NLQ WITHOUT PROJECTION AND SELECTION

Natural language query	SQL	Xpath
Give projects	SELECT * FROM project	/*/client/*
Give all projects		
Give me our projects		
Give me all projects		
Give me projects		
Give our projects		
Give me all our projects		
Give all our projects		
Projects?		
What are our projects?		
All our projects?		
list all our clients	SELECT * FROM customer	/*/client/*
list all our customers		
Show me all our clients and projects	SELECT * FROM customer	/*/client/* /*/project/*
	SELECT * FROM project	

TABLE III. NLQ WITH PROJECTION

Natural language query	SQL	Xpath
give me the names of our client	SELECT employee.name FROM employee	/*/client/name
give me all our clients names		
Names, ages and addresses of clients	SELECT client.name, client.age, client.sex FROM client	/*/clients/name|/*/clients/age|/*/clients/address
display me all our clients names and ages and addresses		
Find all the names of customers and projects	SELECT client.name FROM customer	/*/client/name /*/projects/name
	SELECT project.name FROM project	
Tell me the client name and invoice sum	select CLIENT.NAME from CLIENT	/*/client/name /*/invoice/sum
	select INVOICE.SUM from INVOICE	

TABLE IV. NLQ WITH PROJECTION AND SELECTION

Natural language query	SQL	Xpath
display all customer whose names are "Hanane"	select * from client where client.name = 'hanane'	/*/client[name = “hanane”] /*

all our customer whose names are "Hanane" or "Mustapha"	SELECT* client FROM client where client.names in('Mustapha', 'Hanane')	/*/client[name = "hanane"]/*| /*/client[name="Mustapha"]/*
What are our Clients names and age whose address is "tan tan hay ljadid "?	SELECT client.name, client.age FROM client where client.address ='Hay Dakhla Agadir'	/*/client [address = "tan tan hay ljadid"] /name|/*/client [address = "tan tan hay ljadid"] /age
display all sum of invoice where sum is more than 1000	SELECT invoice.amount FROM invoice where invoice.amount > 1000	/*/invoice [sum > 1000] /sum
Give me the addresses of clients where age is greater than or equal to 26 and name is "Hanane"	select CLIENT.ADDRESS from client where CLIENT.AGE >= 26 and CLIENT.NAME = 'hanane'	/*/client [age >= 26] [name = "hanane"] /address

TABLE V. QUERY WITH AGGREGATE FUNCTION

Natural language query	Generated SQL	Xpath
Give me the number of clients whose names are "Hanane"	SELECT COUNT (*) AS NB_client FROM client where client.names ='Hanane'	/*/count(client [name = "hanane"])
Count me all our project	SELECT COUNT (*) AS NB_projec t FROM project	/*/count(project)
show the max amount of invoice where sum is less than 1000	SELECT MAX (invoice.amount)AS MIN_INVOICE_A MOUNT FROM invoice where invoice.amount < 1000	max (/*/invoice [sum < 1000] /sum)
What is the invoice with the max amount?	SELECT * FROM invoice where invoice.amount in (SELECT MAX (invoice.amount) FROM invoice)	/*/invoice [sum=max (/*/invoice/sum)]/*

We have experienced the performance of system by a set of 13493NLQ, which created using a program. The different results obtained by our test were tabulated in table 6.

TABLE VI. THE RESULTS OBTAINED BY THE FIRST TEST

NLQ	Answered Queries		Unanswered Queries
13493 (100 %)	12622 (93.54%)		871 (6.45 %)
	DBQ Correctly generated	*DBQ Incorrectly generated*	
	11956 (94.79 %)	666 (5.20 %)	

The Table 6 presents that our system gave an answer of 94.79 % in the 13493 queries. For 6.45% of queries, the system didn't give any answer. 11956 NLQs are correctly converted into DBQ. While 666 queries generate incorrect database queries. We said that query is correctly generated if the Database queries produced is syntactically correct To find clarification for these errors, we have examined the output of the NLQ and we obtained the result presented at Table 7

TABLE VII. THE RESULT OF ERRORS EXPLANATION

	Error in Incorrectly generate (666)	Error in Unanswered queries (871)
Error in Morphological analysis	151 (22.98 %)	64 (7.34 %)
Error in syntactic analysis	111 (15.51 %)	807 (92.65 %)
Error in Semantic analysis	389 (58.20 %)	0 (0 %)
Error in Generation of database query	15 (2.28%)	0 (0 %)

From Table 7, we observe that the Morphological analysis produced 7.34 % of errors in unanswered queries and 22.98 % of errors in incorrectly generated queries. Generally, one of the reasons of these errors is that in some NLQ the system considers some verb like noun. For Example in the query "count all invoice whose amount >= 5000" the word "count" is considered as a noun not a verb.

The syntactic analysis has resulted around 15.52 % of errors in the 666 queries that are incorrectly generated and 92.65 % of Errors in unanswered queries. These errors occurred because the parser generates some parse trees that don't match the initial NLQ. For instance, in the query "show me the max of ages and addresses of employees", the parser considered the chunk "max of age" refers to the age object and the chunk "names of client" refers to the client object, whereas both of them refer to one object: "clients".

The errors in semantic analysis represent 58.20 % of the total errors found in database queries that are incorrectly generated. The reason of these errors is that the semantic analysis doesn't convert correctly the parser tree into IXLQ. As example in the sentence: "list me the ages and addresses of employees and clients", the parse tree demonstrates that the "addresses" and "ages" refer to "client" and "employee", however the IXLQ result of this parse tree shows that "address" and "age" refer just to "client".

The errors relate to the generation of database query has just been 2.28 % . These errors can be avoided at program level.

For the Queries that are correctly generated, not all of the generated DBQ match NLQs. The Table 8 displays the number of DBQ matches NLQ and the numbers of DBQ don't match NLQ.

TABLE VIII. DBQ MATCHES NLQ

	DBQ matches NLQ	DBQ not matches NLQ
Number of queries (11956)	10900	1056
%	91.16 %	8.83 %

As presented in Table 8, we show that 91, 99 % of database queries that are correctly generated match NLQ, while 8.83 % of answers not match NLQ.

Results for waiting-time minimization of the above queries, when they are executed on a notebook with AMD C-50 processor 1.00 GHz and 2GO of RAM, are shown in the following table

TABLE IX. RESULTS FOR WAITING-TIME MINIMIZATION

Query	runtime of query asked in first time (s)	runtime if query is already asked (s)	performance achieved
queries without projection and selection	7.80	0.26	96.6 %
queries with projection and without selection	8.85	0.43	95.14 %
queries without projection and with selection	9.92	1.55	84.37 %
queries with projection and selection	10.76	1.917	82.24 %
Query with aggregate function	10.452	1.87	82.10 %

VI. CONCLUSION

This paper is a study on constructing a generic natural language query interface for relational database based on machine learning approach. The main objective of system is to allow communication between database and its human users using natural language. The system has the capabilities to translate NLQ into an equivalent database query after processing through many steps and generates answers in tabular form. The primary advantages of this system are that it is independent of the database language, domain and model and it is able to automatically improve its knowledge base through experience.

As future work we intend to continue solving more complex queries and use other mechanism to improve the performance.

REFERENCES

[1] Al. Gauri Rao, "Natural language Query Processing Using Semantic Grammar," in International Journal on Computer Science and Engineering, Vol. 02, pp. 219–223 , 2010.

[2] R. Valencia-Garcia, F. Garcia-Sanchez, D. Castellanos-Nieves, and J.T. Fernandez-Breis, "OWLPath: An OWL Ontology-Guided Query Editor," in IEEE Transactions on Systems Man, and Cybernetics—Part A: Systems and Humans41(1) p. 121–136 , 2011.

[3] N. Nihalani, S. Silakari, and M. Motwani , " Natural language interface for database: a Brief review," in International Journal of Computer Science Issues 8 (2) p. 600–608, 2011.

[4] Androutsopoulos, G.D. Ritchie, and P. Thanisch, "Natural Language Interfaces to Databases – An Introduction," in Journal of Natural Language Engineering 1 p. 29–81, 1995.

[5] Huangi, Guiang Zangi, and Phillip C-Y Sheu, "A Natural Language database Interface based on probabilistic context free grammar," in IEEE International workshop on Semantic Computing and Systems, 2008.

[6] D.L. Waltz., "An English Language Question Answering System for a Large Relational Database," Communications of the ACM, pp. 526–539, July 1978.

[7] Hendrix, G.G., Sacerdotal, E.D., Sagalowicz, D., Slocum, J., "Developing a natural language interface to complex data," in ACM Transactions on database systems, 3(2), pp. 105–147, 1978.

[8] Warren, D., Pereira, F. , "An efficient and easily adaptable system for interpreting natural language queries," in Computational Linguistics. Vol 8, pp. 3–4, 1982.

[9] C.D. Hafner, " Interaction of Knowledge Sources in a Portable Natural Language Interface," in Proceedings of the 22nd Annual Meeting of ACL, Stanford, California, pp. 57–60, 1984.

[10] P. Auxerre and R. Inder, "MASQUE Modular Answering System for Queries in English- User's Manual," Technical Report AIAI/SR/10, Artificial Intelligence Applications Institute, University of Edinburgh, June 1986.

[11] I. Androutsopoulos, "Interfacing a Natural Language Front-End to a Relational Database(MSc thesis)," Technical paper 11, Department of Artificial Intelligence, University of Edinburgh, 1993.

[12] Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates, "Modern Natural Language Interfaces to Databases Composing Statistical Parsing with Semantic Tractability," COLING 2004.

[13] Y. Li, H. Yang, and H.V. Jagadish, "NALIX: an interactive natural language interface for querying XML," in Proceedings of the International Conference on Management of Data, pp. 900–902 ,2005.

[14] I. Androutsopoulos, G.D. Ritchie, and P. Thanisch, "Natural Language Interfaces to Databases – An Introduction," j. Lang, pp. 29-81, Eng.1995.

[15] E.W. Hinrichs. Tense, "Quantifiers, and Contexts. Computational Linguistics," 14(2), pp.3–14, June 1988.

[16] P. Reis, J. Matias, and N. Mamede, "Edit – A Natural Language Interface to Databases: A New Dimension for an Old Approach," in Proceedingsof the Fourth International Conference on Information and Communication Technology in Tourism (ENTER' 97), Edinburgh, 1997.

[17] Jurgen Albert, Dora Giammarresi, and Derick Wood, " Normal form algorithms for extended context-free grammars," in Theoretical Computer Science 267, pp. 35–47 , 2001.

[18] I. Androutsopoulos, G. Ritchie, and P. Thanisch, "Natural language interfaces to databases-an introduction," in Journal of Language Engineering, v. 1(1), pp. 29–81, 1995.

[19] Manning C. and Schütze H., " Foundations of Statistical Natural Language Processing," MIT Press, Cambridge, 1999.

[20] Luis Tari, Phan Huy Tu, Jorg Hakenberg, Yi Chen, Tran Cao Son, Graciela Gonzalez, and Chitta Baral, " Parse Tree Database for Information Extraction," in IEEE transactions on knowledge & data, engineering , 2010.

[21] B.J. Grosz, "TEAM: A Transportable Natural-Language Interface System," in Proceedings of the 1st Conference on Applied Natural Language Processing, Santa Monica,

[22] B.J. Grosz, D.E. Appelt, P.A. Martin, and F.C.N. Pereira, "TEAM: An Experiment in the Design of Transportable Natural-Language Interfaces," Artificial Intelligence, 32, pp. 39–45, California, 1983.

The Waiting time expressed en second (s)