French Language Interface for Xml Databases

Hanane Bais and Mustapha Machkour
Team of Engineering of Information Systems, Information Systems and Vision Laboratory Faculty of Sciences, Ibn Zohr University Agadir, Morocco

Keywords: xml database, french language, natural language processing, context-free grammar, machine learning

Abstract: To extract information from a database, it is necessary to formulate queries in terms of the language used by the database, such as Structured Query Language (SQL). However, non-expert users who are not computer skills are unable to write such requests. To address this problem, several solutions are proposed. One of these solutions is to use the natural language that represents the ordinary means of obtaining information. Yet, without any help, computers cannot understand this language. This is why it is very important to develop an intelligent interface capable of translating users queries from natural language to queries in database query language. In this paper, we present a generic French language interface for the XML database. In this paper, we present a generic French language interface for the XML database. The proposed interface translates French language query potentially after reformulation, into an XPath expression that can be evaluated against an XML database. The advantage of this system is that it functions independently of the database domain and automatically improve its knowledge base through experience.

1 INTRODUCTION

The Natural Language Interface for Databases (NLIDB) is an intelligent and flexible system capable of translating a natural language query into a query in database query language [Androutsopoulos et al., 1995]. NLIDB can be considered as one of the fundamental subjects of artificial intelligence and databases. Traditionally, people are used to using forms to access information stored in databases. Nevertheless, their expectations depend strongly on the capabilities of these forms. However, NLIDB provides powerful improvements in the use of this information. Because it offers a large number of users a simple, uniform and unlimited access to the data. Success in this area of research due partly to the benefits that may come from NLIDB [Bais et al., 2016].

Access to information stored in a database by natural language is a very important and motivating method. Numerous research prototypes have been proposed to arrive to process the user's queries in natural language. Most of these works have been designed to interface Prolog and relational databases. For XML database, there is one contribution proposed to extract data from XML databases using the English language.

Our proposed system functions with XML databases. It interprets the user's queries in the French Language Query (FLQ) to an XPath command, and it retrieves the suitable answer from the XML database. The system has the ability to process a very important number of FLQ.

The remainder of the paper is organized as follows Section 2 gives an overview of existing work, showing their advantages and limitations. Section 3 presents a brief description of the proposed system and the architecture of the system. Section 4 reports the experimental results. Finally, Section 5 presents the conclusions and some possible extensions of this work.

2 RELATED WORKS

Over the past fifty years, many attempts have been made to create an intelligent interface in natural language. There has been a great deal of research presenting the theories for the implementation of NLIDB. The first attempts are as old as any other search area in NLP. In fact, NLIDB has been one of the most important successes of NLP since its

Bais, H. and Machkour, M.
French Language Interface for Xml Databases.
DOI: 10.5220/0009772200730081
In *Proceedings of the 1st International Conference of Computer Science and Renewable Energies (ICCSRE 2018)*, pages 73-81
ISBN: 978-989-758-431-2

inception. The first NLIDB appeared in the late sixties and early seventies [Waltz., 1978]. Some of this system like LUNAR [Akerkar et al., 2008] and BASEBALL [Gauri et al., 2010] were designed for a particular database domain and thus could not be easily adapted to interface other different databases. By the end of the seventies, several other NLIDB had appeared. Some of these systems like LIFFER / LADDER (Hendrix, 1978) are used semantic grammars inherent to the database domain. The problem of these systems was that the semantic grammar used was adjusted to a certain domain and that reduce the portability of these systems since the rewriting of the grammar was necessary whenever the domain of application is modified. In the mid-1980s, several NLIDBs appeared. This research area had become very popular and numerous research prototypes were put in place with different approaches to handle user queries in natural language. The CHAT-80 system [Warren et al., 1982] is one of the most referenced systems in the 1980s. The system has been implemented in Prolog. The major problem in this system is that it can be used only for a specific database domain. The CHAT-80 code has been widely distributed, and it is the basis for several other systems such as MASQUE (Modular Answering System for Queries in English) [Auxerre et al., 1986]. MASQUE was also intended for Prolog databases. It answers questions written in English that are related to certain areas such as geography and airplanes. Contributions in the area of the NLIDB are continuous in the nineties. The majority of these contributions have focused on the querying of relational databases by the use of natural language instead of SQL. However, these systems are always designed for a specific of domain application. Androutsopoulos, et al developed an extended version of the MASQUE system, called MASQUE/ SQ [Androutsopoulos et al., 1993]. This system can be interfaced with all commercial databases that support the SQL language. MASQUE/ SQL answers the questions of the user in English by generating an SQL code that is executed by the relational DBMS. After the 1990s, there have been interesting approaches operate to design NLIDB independently of the database domain. These systems can efficiently handle requests for different domains without any reconfiguration. One of the best examples of this approach is PRECISE [Popescu et al., 2004]. PRECISE is developed at the University of Washington by Ana-Maria Popescu et al in 2004 and it targets relational databases. The majority of these works interface Prolog and relational databases. For the XML database, the only proposition is NALIX [Li et al., 2005]. It is a natural language interactive natural interface developed at the University of Michigan by Yunyao Li et al in 2006. The database used for this system is XML database (Extensible Markup Language) with 'Schema-Free XQuery' as databases query language. Schema-Free XQuery is a language designed primarily to extract XML database information. The idea of this language is to use keyword search for databases. The transformation processes take place in three steps, generation of the parse tree, validation and then translation of this parse tree into an XQuery query.

Generally, most of the NLIDBs are translating English language queries. Other systems translate queries written in Spanish [Rangel et al., 2002], Urdu [Ahmad et al., 2009], Chinese [Li et al., 2015] and Hindi [Kataria et al., 2015] languages. For the French language, the first system process this language is Edite [Reis et al., 1997]. Edite is a multi-langue natural language interface to relational databases. It answered user's queries expressed in Portuguese, French, English, and Spanish. It functions independently of database domain. The second system designed for the object-oriented database [Hemerelain et al., 2010]. It presents the semantic analysis of queries written in the French language.

In this study, we focused on the development of another generic interface for querying XML databases by using the French language. This interface operates independently of the domain of application. In addition, the use of automatic learning allows our system to have the ability to automatically improve its knowledge base via experience.

3 PROPOSED ARCHITECTURE

In Figure 1, we present the proposed architecture. The proposed architecture is based on the intermediate representation languages approach. The use of this approach due to the difficulties of directly translating the FLQ into an XPath. The idea of this method is to map firstly the FLQ into a logical intermediate query, expressed in an in XML form. Then, this logical query translated into an XPath query and evaluated against the XML database [1]. This architecture can be divided into two major components.

The first part starts from the natural language query to the logical query. The second part starts

from the logical query until the XPath query is generated. The idea to express the logical query in XML form, adds reasoning capabilities to the system. Moreover, since the logical request is independent of the database, it can be ported to different database query language as well as to other domains.

3.1 Linguistic Component

As shown in Figure 1, the linguistic component composes of three analyses: morphological, syntactic and semantic.

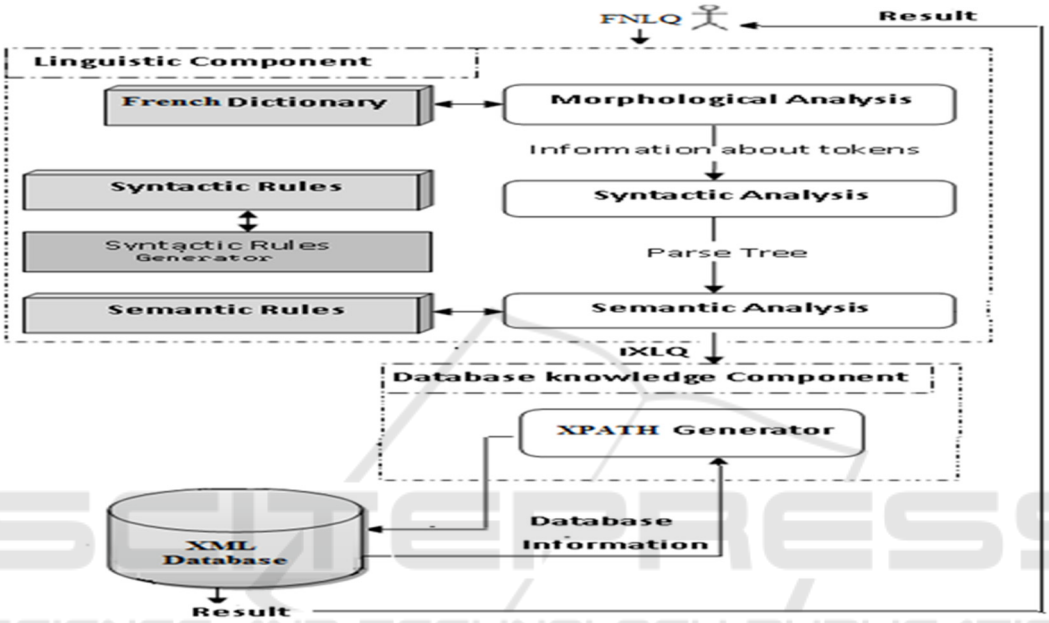

Figure 1: System architecture

3.1.1 Morphological Analysis

The term morphology comes from classical Greek (morpheme) and means the study of form. Morphology is concerned with the study of the external structure of an object. In linguistics, it is the study of how words combine to formulate the sentence. The morphological analysis is responsible for reading the FLQ and dividing it into primitive elements called token. Then, it returns information about each token. This process is performed using the next functions. We note that in this function, we give the example by the FLQ: "Affiche les noms et les addresses des étudiants ayant un age < 19" which means: give me names and addresses of student whose age < 19. The FLQ used in this example contains some spelling errors for shows the using of the function spell checking:

- **Tokenization:** this function used to divide the FLQ into a primitive element, which is considered as a single logical unit.

  Example: < affiche> < les> <noms> < et> <les> < addresses> <des> < étudionts > < ayant > <l> < un > <âge> < '<' > <19>
- **Spelling checker:** used to make sure that each word is in the dictionary. If this is not the case, then spell checking is performed or a new word is added to the system vocabulary.

  Example: < affiche> < les> <noms> < et> <les> < adresses > <des> < étudiants > < ayant > < un > <âge> < '<' > <19>
- **POS Tagger:** Is the process used to associating the grammatical function to each word of the FLQ.

  Example: <affiche , V> <les , DET> <noms , NC> <et , CC> <les , DET> < adresses , NC> <des , P> < étudiants, NC> < ayant, PROREL>

<un , DET> <âge , NC> < <, PUNC > <19 , ADJ>

- **Morpheme:** this function determines the morpheme of each token which is the minimal unit bearing meaning .

Example: < noms : nom> < adresses : adresse> < étudiants: étudiant>

3.1.2 Syntactic Analysis

The syntactic analysis used to show how the tokens in the FLQ are related to each other [Tari et al., 2010]. This function allows our system to generate the parse tree or the derivation tree.

For generating the parse tree, we use a context-free grammar. The context-free grammar used the by our system is presented in Figure 2. The elements of CFG are defined by: CFG=[N,T,R,S] where:

- N a set of non-terminal symbols. T is a set of terminal symbols.
- R is a set of context-free productions.
- S is the start symbol used to represent the ANLQ.

R1: SENTENCE → OBJECTS CONDITION ORDER CONJUNCTION SENTENCE
R2: SENTENCE → QUE AUX_VERB OBJECTS CONDITION ORDER CONJUNCTION SENTENCE
R3 :SENTENCE → VERB_PHRASE OBS CONDITION ORDER CONJUNCTION SENTENCE
R4 :VERB_PHRASE → VERB PERSONAL_PRONOUN
R5 :AUX_VERB → EST | SONT
R6: QUE→QUI | QUEL | QUELLE | QUELLES
R7: OBJECTS → OBJECT CONJUNCTION OBJECTS
R8 :OBJECT→ NOUN_PHRASE
R9 :NOUN_PHRASE → QUANT POSSESSIVE_PRONOUN DET NOUN CONJUNCTION NOUN_PHRASE
R10 :NOUN_PHRASE → NOUN_PHRASE PREPOSITION_PHRASE CONJUNCTION NOUN_PHRASE
R11 :NOUN_PHRASE → DETERMINER ADJECTIVE NOUN CONJk9UNCTION NOUN_PHRASE

This grammar is composed of a set of syntactic rules, which corresponds to the possible syntactic structure of the FLQs. We can devise these rules in two categories:

- Domain-independent rules: are the syntactic rules in which the right part content only non-terminal symbols (i.e. rules 25 in Figure 3).
- Domain-dependent rules: are the syntactic rules in which right part content terminal symbols (i.e. rules 9 in Figure 3) [Albert et al., 2011]

In order to be independent of the domain of database, we can't interpret all possible instance of domain-dependent rules. For that we add to our system a generator of syntactic rules. This generator checks whether all the syntactic rules necessary to parse the FLQ ∈ Knowledge Base (KB). If not, it automatically detects the necessary syntactic rules,

R12 : QUANT → TOUS | TOUTE | TOUT| CHAQUE
R13 :PREPOSITION_PHRASE → PREPOSITION NOUN_PHRASE
R14 :PREPOSITION → DE | DES | DU |
R15 :PERSONAL_PRONOUN → MOI | NOUS | VOUS | LUI
R16 :POSSESSIVE_PRONOUN→ MON| VOUS| NOUS |TON| LEUR | LEURS
R17 :CONDITION→COND OP CONJUNCTION CONDITION
R18 :COND → DONT | AVEC | OU
R19 : VERB → DONNER| AFFICHER| MONTRER …
R20 :CONJUNCTION → ET | OU
R21 :OP → OBJECT SYMBOL VALUE
R22 :SYMBOL→ IS | = | > | >= | < | <= | <> | COMME
R23 :ORDER → ORD NOUN_PHRASE CONJUNCTION ORDER
R24 :ORD→ ORDONNER PAR | TRIER PAR
R25: NOUN → CLIENT| FACTURE | PROJET…

Figure 2: The context-free grammar used by the system

creates and adds them to the KB. The addition of syntactic rules generator to our system, help to adapt its KB with the FLQ. So, the system automatically improves its knowledge base through experience. The function of the generator of syntactic rules is described in the next algorithm.

Algorithm Syntactic_Rules _Generator
Input
FLQ a French Language Query
Output
a set of syntactic rules R = {(SRk), 1≤ k≤ m }
Begin Divide the FLQ into a set of words W = {(wi : GFi) , 1≤ i ≤m} where GFi is the Grammatical Function of each word wi ;
For each word (wi : GFi) ∈ W loop
Generate the syntactic rule SRk correspond to wi;
If SRk ∉ KB
add SRk to KB;

```
  End if
 End loop
Return R;
 End Syntactic_Rules _Generator
```

Figure 3 displays the parse tree corresponded to FLQ:
affiche les noms et les adresses des étudiants ayant un âge < 19

3.1.3 Semantic Analysis

The semantic analysis produces an XML logical query which used to assign a logical interpretation to the parse tree created by the syntactic analysis. This is done by using a set of semantic rules. Each syntactic rule defined in the CFG has an equivalent semantic rule. For that this process called rule-by-rule style [Reis et al., 1997]. Table 1 displays some instances of semantic rules with their equivalent syntactic rules:

Table 1: Semantic rules with their equivalent syntax rules.

Syntactic rule	Semantic rule
NOUN_PHRASE PREPOSITION_PHRASE	<attribute > <object>
NOUN_PHRASE NOUN_PHRASE PREPOSITION_PHRASE	< attribute 1> < attribute 2 > <object>
NOUN_PHRASE PREPOSITION_PHRASE PREPOSITION_PHRASE	< attribute > <object1> < object2 >
NOUN_PHRASE SYMBOL VALUE	<attribute> <symbol> < value>

We have already mentioned in the previous paragraph that the application of semantic rules on the parse tree produces an XML logical query. In XML, we can define the structure of XML logical query by the following Document Type Definition [DTD]:

```
<?xml version="1.0" encoding="UTF-8"
standalone="yes"?>
<!DOCTYPE logical query [
<!ELEMENT REQUEST [SELECT, COND*,
ORDER* ]>
<!ELEMENT SELECT [OBJECT+]>
<!ELEMENT CND [OBJECT, SYMBOL, VALUE+
]> <!ELEMENT ORDER [OBJECT]>
<!ELEMENT OBJECT [NAME, ATTRIBUTE+]>
<!ELEMENT ATTRIBUTE [NAME, AGGREGA +]>
<!ELEMENT NOM [PCDATA]>
<!ELEMENT VALUE [PCDATA]> <!ELEMENT
SYMBOL [PCDATA]> <!ELEMENT AGGREGA
[PCDATA]>
]>
```

The following XML logical query displays the logical query associated to the parse tree of the FLQ:

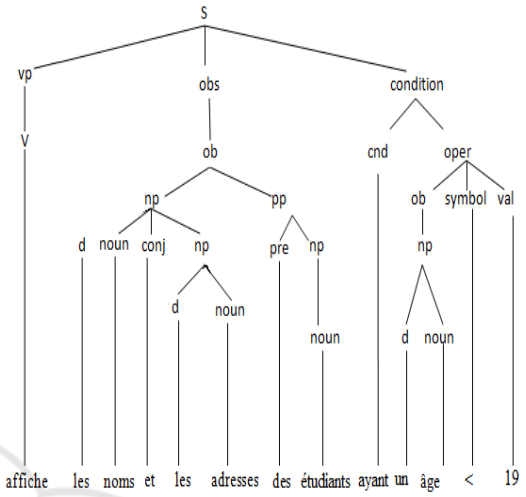

Figure 3: Example of parse tree

```
<QUERY>
  <SELECT>
      <OBJECT>
         <NAME>étudiants</NAME>
         <ATTRIBUT>
           <NAME>noms</NAME>
         </ATTRIBUT>
         <ATTRIBUT>
           <NAME>adresses</NAME>
         </ATTRIBUT>
      </OBJECT>
  </SELECT>
  <CONDITION>
     <OBJECT>
         <ATTRIBUT>
            <NAME>âge</NAME>
         </ATTRIBUT>
      </OBJECT>
      <SYMBOL> < </SYMBOL>
      <VALUE>19</VALUE>
  </CONDITION>
</QUERY>
```

3.2. Database Knowledge Component

The function of the database knowledge is to translate the XML logical query into XPath query. This is done by mapping the elements of the XML logical query into its clause in XPath query. This process done in many steps. Each one manipulates particular part in the logical query.

- Step1: extract the names of the attribute from the logical query.
- Step1: select the portion of the logical query that is contented the table name or a group of table names
- Step1: form the conditions of selection from the logical query.

Each step is followed by an examination, to validate if the name of tables and attributes exist in the XML database. If it's not the case, a mapping table will use. This later stores the synonyms of table and attribute names. After the generation of the XPath queries, the system executes it and then, shows the answers in XML form.

4 SYSTEM RESULTS

The interface shown in Figure 4 displays the result of translation of the FLQ:

affiche les noms et les adresses des étudiants ayant un âge < 19.

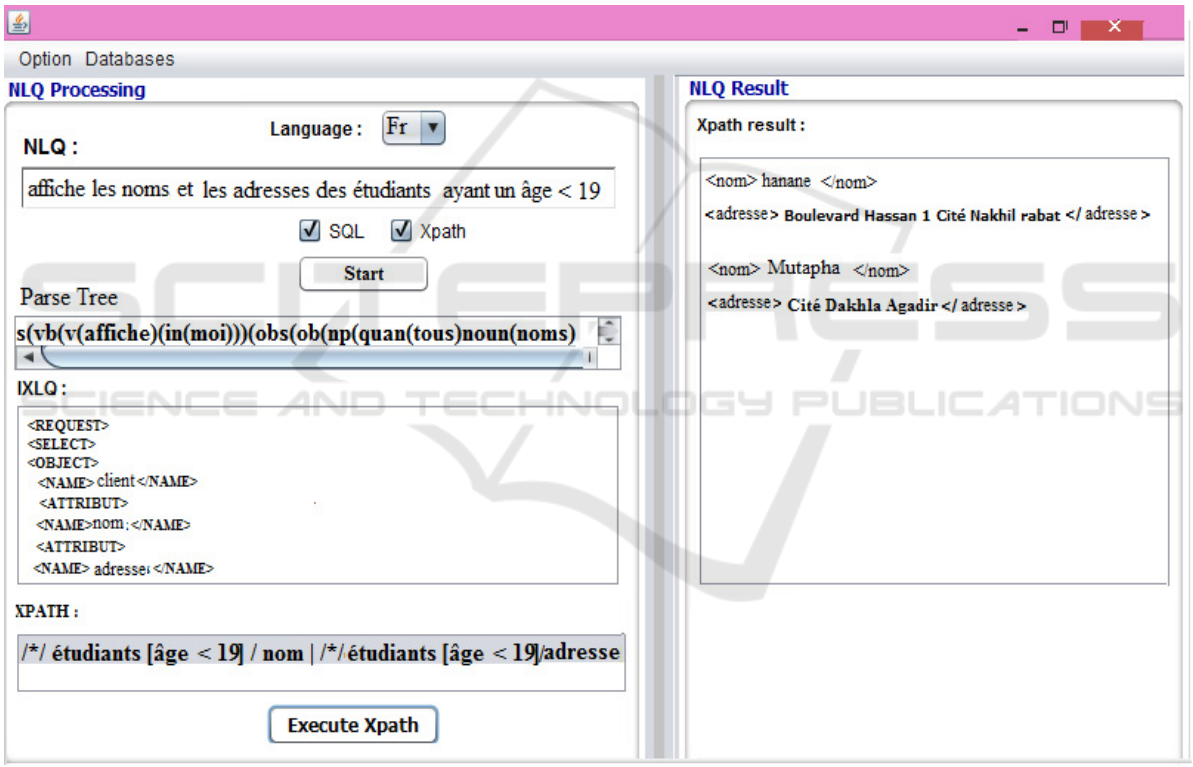

Figure 4: System interface

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

Table 2: FLQ without projection and selection

FLQ	Generated XPATH
affiche les clients	
affiche tous les clients	/*/client/*
affiche moi nos clients	
affiche moi tous les clients	
Affiche moi les clients	
affiche nos clients	
affiche moi tous nos clients	
affiche tous nos clients	
clients?	
quels sont nos clients?	
liste tous nos salaries	/*/ salarié /*
liste tous nos employés	

montre moi tous nos clients et projets	/*/client/* /*/projet/*

Table 3: FLQ with projection

FLQ	Generated XPATH
Donne moi les noms des étudiants	/*/ étudiants /nom
noms, âges et adresses des employées	/*/ employées /nom| /*/ employées /âge
Montre moi tous les clients noms, âges et adresses des employées	/*/clients/adresse
Quels sont tous les noms et les adresses des clients	/*/client/nom|/*/client/adresse
Cherche tous the noms des employés et des clients	/*/client/nom /*/employé/nom
Trouve les noms des clients et les montants des factures	/*/client/nom /*/facture/montant

Table 4: FLQ with projection and selection

FLQ	Generated XPATH
Montre tous les étudiants dont le nom est "Hanane"	/*/ étudiant [nom = “hanane”] /*
Tous notre clients dont nom est "Hanane" ou " Mustapha"	/*/client[nom = “Hanane”]/*| /*/client[nom=”Mustapha”]/*
Montre tous les enseignants avec l’âge est entre 28 et 40	/*/ enseignant [âge > 28] [âge<40]/*
quels sont les noms des employées dont l’adresse est "Agadir dakhla " ?	/*/employée [adresse= " Agadir dakhla "] /nom|/*/employée [adresse = "Agadir dakhla "] /âge
Affiche moi tous les adresses des client Qui ont l’âge inférieur à 30	/*/client [âge < 30] / adresse
cherche les adresses des clients avec âge supérieur ou égale à 26 et le nom est "Hanane"	/*/client [âge <= 26] [nom = "hanane"] /adresse

Table 5: Query with aggregate function

FLQ	Generated XPATH
Donne moi le nombre des fournisseurs dont le nom est "Hanane"	/*/count(fournisseur [nom = "hanane"])
compter tous nos projets	/*/count(projet)
Montre moi la moyenne des âges des clients	avg (/*/client/âge)
Donne moi le minimum âge des étudiants	min (/*/étudiant /âge)
Affiche le maximum âge des clients dont l’âge est inférieur ou égale à 40	max (/*/client [âge < 1000] /âge)
Quels sont les employés avec le maximum âge?	/*/ employée [âge=max (/*/employée /âge)]/*

To experience the performance of our system, we use a set of 2000 FLQ. The results obtained by this test were tabulated in Table 6.

Table 6: The results obtained by first test

	Answered Queries	Unanswered Queries
2000 (100%)	1944	56
	97,2%	2,8 %

Table 6 presents that our system translates 97.2% of 2000 FLQ are translated to XPATH.

Table 7: The results obtained by the second test

	XPath Correctly generated	XPath Incorrectly generated
NB (1944)	1930	14
%	99,27%	0,72%

From Table7 we show that 1930 FLQs are generated correct XPath queries. We said that query

is correctly generated if the database queries produced is syntactically correct.

For the queries that are correctly generated, not all of the generated XPath match FLQs. Table 8 displays the number of XPath matches FLQ and the numbers of XPath don't match FLQ.

Table 8: XPATH matches FLQ

	XPath matches NLQ	XPath doesn't match NLQ
NB (1930)	1897	33
%	98.29 %	1.70%

As presented in Table 8, 98.29% of XPath queries that are correctly generated matches FLQ.

5 CONCLUSION AND FUTURE WORK

This research paper presents a model of an architecture of intelligent interface for querying XML database. It proposes a method for translating FLQ into an XPath queries. The main objective of this interface is to allow communication between the XML database and its users using the French language. One of the advantages of this interface is that it functions independently of database domain and it improves automatically its knowledge base through experience. The results of experimentation show that the methods employed in the system have the capabilities to produce an XPath queries for a very important number of FLQ.

As future work, we will continue to solve more complex queries. Also, we intend to apply the methods used in the paper to translating queries in other languages, such as the Arabic language.

REFERENCES

Akerkar, R., Joshi, M.,2008. Natural Language Interface Using Shallow Parsing. In International Journal of Computer Science Applications, Vol. 5, pp. 70-90.

Albert, J., Giammarresi, D., Wood, D., 2001. Normal form algorithms for extended context-free grammars. Theoretical Computer Scienc, Vol. 267, pp. 35-47.

Androutsopoulos, I., Ritchie, G.D., Thanisch, P., 1993. MASQUE/SQL - An Efficient and Portable Natural Language Query Interface for Relational Databases. Proceedings of the 6th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems,1_4 June1993 Edinburgh, Scotland, pp.327-330.

Androutsopoulos, I., Ritchie, G., Thanisch, P., 1995. Natural Language Interfaces to Databases – An Introduction. In Journal of Natural Language Engineering 1 Part 1, pp.29-81.

Auxerre, P., Inder, R., 1986. MASQUE Modular Answering System for Queries in English, User's Manual, technical report AIAI/SR/10, Artificial Intelligence Applications Institute, University of Edinburgh.

Bais, H., Machkour, M., Lahcen, K., 2016. Querying database using a universal natural language interface based on machine learning. In International Conference on Information Technology for Organizations Development (IT4OD). IEEE.

Gauri, R., Agarwal, C., Chaudry, S. , Kulkarni, N. , Patel, S.H., 2010.Natural language query processing using semantic grammar'. In International Journal on Computer Science and Engineering, Vol. 2, pp. 219-223.

Hemerelain, B.Belbachir, H., 2010. Semantic Analysis of Natural Language Queries for an Object Oriented Database. In JSEA, Vol. 03, pp. 1047-1053.

Hendrix, G., Sacerdoti, E., Sagalowicz, D., Slocum, J., 1978. Developing a natural language interface to complex data. ACM Transactions on Database Systems, Vol. 3, pp. 105-147.

Kataria, A.Nath, R., 2015. Natural Language Interface for Databases in Hindi based on Karaka Theory. In International Journal of Computer Applications, Vol. 122, pp. 39-43.

Li, Y., Yang, H., Jagadish, H.V., 2005. Nalix: an interactive natural language interface for querying xml. SIGMOD 05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pp. 900-902.

Li, Z., Li, J., Ning, W., 2015. Research on Chinese Natural Language Query Interface to Database Based on Syntax and Semantic. In Applied Mechanics and Materials, Vol. 731, pp. 237-241.

Papadakis, N., Kefalas, P., Stilianakakis, M., 2011. A tool for access to relational databases in natural language. In Expert Systems with Applications, Vol. 38, pp. 7894-7900.

Popescu, A-M., Armanasu, A., Etzioni, O., Ko, D., Yates, A., 2004. Modern Natural Language Interfaces to Databases: Composing Statistical Parsing with Semantic Tractability. In Proc. COLING'04.

Rachide, A., Khan, M.A., Ali, R., 2009. Efficient Transformation of a Natural Language Query to SQL for Urdu. Proceedings of the Conference on Language and Technology, pp. 53-60.

Rangel, R.A.P., Gelbukh, A.F., Barbosa, J.J.G., Ruiz, E.A., Mej_a, A.M., S_anchez, A.P.D., 2002. Spanish Natural Language Interface for a Relational Database Querying System. 5th International Conference, TSD 2002 Brno, Czech Republic, pp. 123-130.

Reis, P., Matias, J., Mamede, N., 1997. Edite A Natural Language Interface to Databases: A New Dimension for an Old Approach. Proceedings of the International Conference in Edinburgh, Scotland, pp. 317-326.

Tari, L., Tu, P.H.,Hakenberg, J., Chen, Y., Son, T.C., Gonzalez, G., Baral, C., 2010. Parse Tree Database for Information Extraction. Proceedings of IEEE transactions on knowledge and data.

Waltz, D., 1978. An English language question answering system for a large relational database. Communications of the ACM 21, pp. 526-539.

Warren, D., Pereira, F., 1982. An Efficient Easily Adaptable System for Interpreting Natural Language Queries. In Computational Linguistics, Vol. 8, pp. 110-122.