Turkish Query Engine on Library Ontology

Sadi Evren SEKER
Department of Computer Engineering, Istanbul University, Istanbul, Turkey
academic@sadievrenseker.com

Abstract *Purpose of this project is implementing conversational software to interface dialog based sentences between the user and a library database.*

This software implemented with a special expertise on library dialogs. The number of possible library dialog sentences is limited and this project covers almost all of these possible sentences. The input sentences are accepted as Turkish and a flexible management system added for further additions. For example any sentence missed on this project can be added with a simple entry on the YACC file.

The technology utilized during this project is YACC and LEX implementation on LINUX. Also the database of the project is implemented over MySQL. LEX and YACC produces C source codes and the functionality of semantic processing and the database queries are also implemented in C language.

One of the hardest part of this project is implementing Turkish language capability over C programming environment on LINUX. All the technological modules of this project which are MySQL, C, LEX, YACC and ZEMBEREK created different problems with the Turkish inputs. During these problems I have searched Internet for the Turkish input implementations of LEX and YACC or the MySQL connection through C and as a result of my findings this project is the first time implementation of Turkish characters sets by LEX, YACC and MySQL at the same time on LINUX.

One of the most important achievements after accomplishing this study is the flexibility of the input sentences. Anybody can add a new grammar rule to the YACC file buy obeying the regular expression structure of YACC. After a successful addition the project will search for this new addition in the input sentences and the answers related to this input will be produced.

Keywords: Natural Language Processing, Ontology, Morphological Analysis, Syntactic Analysis.

1 Introduction

Aim of this project is implementing conversational library software, which will be capable of understanding the user inputs and reply back the user with the desired information. The user inputs can be anything with no limitations of course, but the software is only responsible to understand the library based dialog inputs.

The only data source of this project is a database, implemented to keep the information of writers and books. The database implementation kept as simple as possible to make the conversational part of project more challenging.

The project implemented in a flexible manner, so any sentence pattern can be added to the conversational module.

Almost all for the bots are developed for English and the biggest difference of this project and the current bots are the language they have developed. Also they have mostly frozen sentence structures instead of a formal approach with morphological, syntactic and semantic levels. They mostly try to find out a NLP parsing by using the prepared sentence templates and when a user input matches this template the related answer is displayed. This is the most important second part of this project from the projects above.

Also the database of the above bots is very limited and they do not have a chance to measure their successes. Since the world has unlimited number of natural language sentences (theoretically) the above bots can never be measured in a way of all possible inputs.

Contrarily this project has a limited number of input possibilities and the maximum possible sentences can be touched one day. Of course during this study the all possible sentence alternatives are not outlined but this is a possible case for the future studies.

In the case of finding the possible alternatives the performance of success will be able to measure by using the current implementation.

Of course the measurement is an essential element in engineering studies so I have tried to find out a way of implementing these possibilities. The solution is dividing the possible input sentences into two groups. In the first group the sentences will be used for training and the code will be developed. The second group of input sentences will be kept for the testing purposes and the performance of success will be measured by using the outputs of these sentences.

2 Design of Project

All the possible inputs will be covered and a BNF [1] representation will be created in this chapter. Also a finite state machine will be drawn depending on the BNF representation of the input sentences.

The design phase covers the details of technological selections and their adaptations to the project.

This paper has been published in IKE12, Proceeding of Internet Knowledge Engineering, 2012, ISBN:1-60132-222-4, Vol. 1 pp. 26-33

2.1 Layered Approach and Technological Background

The project can be separated into 4 layers as demonstrated below:

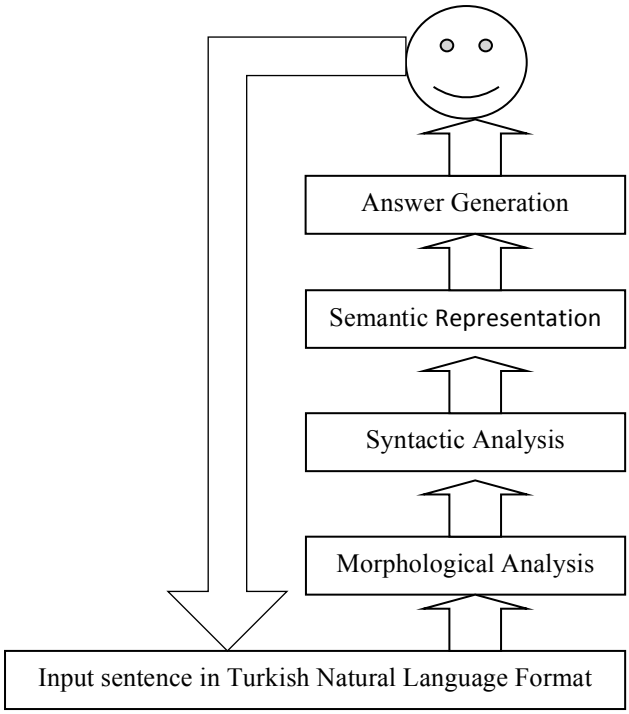

Fig. 1. Layered representation of the Project

Above figure represents the three well-known layers. The natural language processing starts by the morphological analysis the order of words is parsed in the syntactic phase and finally a semantic representation is found out. The semantic representation is of course not a user-friendly representation. The aim of this representation is only keeping track of the meaning of the input sentence. The project gets this semantic representation and tries to find out a possible answer for the user query and generates an answer in natural language again.

2.2 Morphological Analysis

The morphological analysis [2] is the phase of getting the meaning of the words. The word meanings are mostly depends on the root of the word. Of course the suffixes may shift the meaning of the word. During the studies the project has got 3 phases.

1. The pre yacc phase
2. Yacc integration phase
3. Morphological Regular Expressions phase

In the pre yacc phase the integration of lex and zemeberek has taken place and the words have been separated into three categories:

- Frozen words
- Zemberek processed words
- Database queried words

The frozen words was the hard coded words which does not has any suffixes and does not match in the database.

Zemberek processed words was accepted words in the syntax of the project but they do require some morphological analysis because of many possibilities with the words.

For example consider the below sentences:
Beyaz isminde kitabı yazan kimdir
(Who has written the book named "Beyaz")
Beyaz isminde kitabı olanlar kimlerdir
(Whose book has a name of "Beyaz")
Beyaz isimli kitabı olan kimdir
(Who has a book named "Beyaz")
Beyaz ismindeki kitabı yazan kimdir
(Who is the writer of the book "Beyaz")
Beyaz ismindeki kitabı yazanlar kimlerdir
(Who are the writers of the book "Beyaz")
Beyaz isimli kitapları olanlar kimlerdir
(Who are the writers with the book "Beyaz")

All the above sentences have almost same syntactic order with same semantic meaning. So the user entering any of the above sentences means the same thing for each of the above options.

The morphological meaning gets a great variety by the above sample for example all the below words have the same semantic:

Kitap*(book)*, kitabı*(the book)*, kitaplar*(books)*, eseri *(the work)*, eserleri *(works)*, eser *(work)*, romanı *(the novel)*, romanları *(novels)*, roman*(novel)* … etc.

Similarly from the above sentences also the below words has the same semantic representation:

Isminde *(in the name)*, ismindeki *(the named)*, isimli *(with the name)*

So the morphological analysis should take care of the word varieties and return the same semantic representation for the words in the same meaning.

Finally the third word type in the morphological analysis is the database queried words. These are the free words for any possible variety. So the user can use almost any word in this part give the name of a writer or a book.

The pre-YACC phase was the querying phase from the database. So the user enters a word in the appropriate place of the sentence the word gets queried in the database.

The database returns the word result and morphological analysis gets sure that the word is a writer or book phrase.

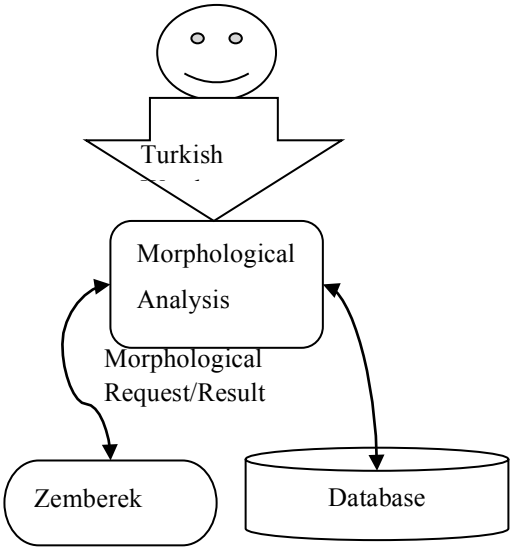

Fig. 2. Deployment of morphological analyzer

Above figure demonstrates the deployment of morphological analyzer between user, zemberek [5] and database in the pre yacc development phase.

After the failure of the yacc integration the above deployment has changed.

2.3 YACC Integration Phase

After integrating YACC [3] into the project, the queries have been carried to the YACC to generate an answer to the inserted question. The questions are parsed into the words and the LEX was only processing these words and producing the frozen results.

During this phase the zemberek integration and the database connection was completely replaced with the frozen words.

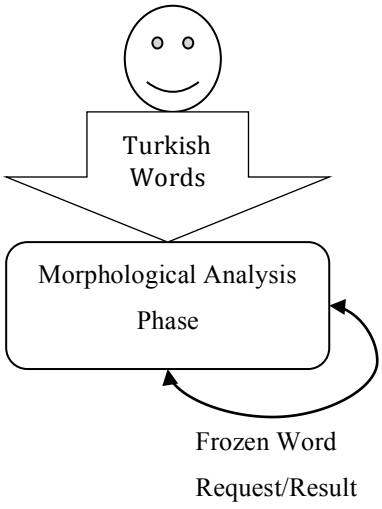

Fig. 3. YACC integration development phase of morphology

Above figure demonstrates the processing of the user input in the LEX[3]. Of course the result of the above process was carried out to the syntactic analysis which was YACC in this step.

The problem occurred in the possibilities of the words. In the previous section the list of all possible syntax entries with great variety requirement on the morphology has forced a solution to the morphological integration.

The problem was the number of frozen words entered in the LEX.

This problem has solved in the final version with morphological regular expressions phase.

2.4 Regular Expressions in Morphology

This final version of morphology solution can be understood better from the below example:

Let's consider the below frozen word entries in the LEX file from the previous section:

- Eseri (His art work)
- Eserini (The art work of him)
- Eseriyle (With the art work of him)
- Eserleri (his art works)
- Eserler (art works)

Before this development phase all the above words was accepted by separate entries in the LEX file. After the regular expression improvement the whole above sentences was accepted by a single rule:

```
"eser"[a-zğüıöçş]* {
      return KP;
}
```

Also this reduces the number of syntactic rules since each variety above is represented by only a single entry of KP return value.

Furthermore this improvement gives another possibility of reducing syntax by removing the below word alternatives:

```
"roman"[a-zğüıöçş]* {
return KP;
}
"hikaye"[a-zğüıöçş]* {
return KP;
}
"kitab"[a-zğüıöçş]* {
return KP;
}
```

The syntax would have been covered all the possible entries of above word roots as a rule.

Besides the above improvement a major obstacle has faced during this implementation. The shorter roots were getting the priority which causes problem.

For example consider the word root "yaz" and "yazar" in the same LEX file entry. The "yaz" root has the priority since it is shorter than the "yazar" entry.

This problem causes a masking of "yazar" entry in the LEX file.

The solution is quite simple. The order of the LEX file entries defines a priority on the rules. For example the "yaz" root should be kept below the "yazar" root so the "yazar" root will get priority and only if the "yazar" root

has not matched than the LEX goes beyond and checks for whether the "yaz" root matches for the current word.

2.5 Syntactic Analysis

This chapter covers the syntactic analysis of the project. The syntactic analysis is responsible from the order of the words. The word order is parsed in this phase by depending on the morphological semantic and the meaning of the words.

For example the word meanings can vary from order to order, so the syntactic analysis goes through these word order alternatives and makes a comparison between the expected word order and the input word order. In the case of a match with the expected and input word order, the syntactic analysis generates a semantic representation for the answer generation phase. In fact this last phase can be integrated by the syntactic analysis since the possible outputs are already separated at this level.

The design of syntactic analysis can be separated into two major parts. The first part was mostly targeting the ontological queries, so the failure of the LEX integration.

- Ontological Queries phase
- Database Query phase

2.6 Ontological Queries Phase

The first phase gets queries the ontological questions. Unfortunately the implementation of ontological queries[4] was not possible because of the time limitations. The possible ontology queries were quoted below:

Below list holds the sample inputs targeted to be understood by the software. (Library Ontology)

Sample Sentences and their syntactic frames and semantic representations:

Before formulating the possible sentences in library dialog the formal definition of word phrases are given below:

Book Phrases (BP) holds the information about the name of the book. The below list holds the possible variations of the book names in Turkish sentences.

```
BP > BN
BP > BN Roman [ı|nı|lar|ları|larını]
BP > BN Kitab [ı|nı|lar|ları|larını]
BP > BN Eser [ı|nı|lar|ları|larını]
BP > BIR BP
BP > HERHANGI BIR BP
```

Writer Phrases (WP) holds the information about the name of the writer. The below list holds the possible variations of the writer names in Turkish sentences:

```
WP > WN
WP > WN Yaz [dığı]
WP > BIR WP
WP > HERHANGI BIR WP
```

Also library dialogs may contain the translation of the books. For example a book may be translated into many languages and the customer may request any copy of these languages. Below phrases are the samples of the translation phrases which may be contained in a possible library dialog in Turkish:

```
TP> TN [si]
```

Questions also holds several question words like who, where, when, etc. Below phrase is responsible to detect such phrases:

```
QP> [kaç [a|da|ı|ncı] |ne|hangi [si]
|var mı|yok mu
```

Other general purpose phrases can be listed as below:

```
VP -> Verb Phrases (consists of a verb
and possibly several adverbs)
NP -> Noun Phrases (consists of a noun
and possibly adjectives)
```

According to above word phrases the possible formulation of some target sentences are given below:

```
LP>BP TP NP QP VP?
```

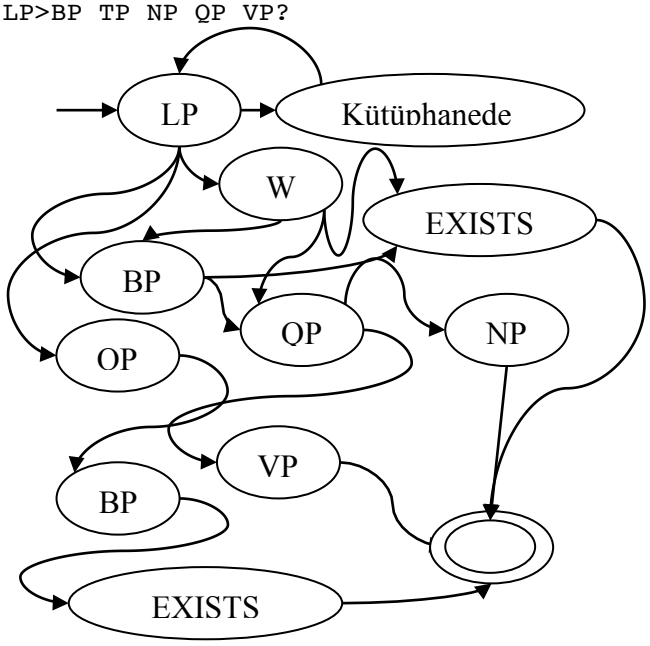

Fig. 4. FSM of library phrases

Figure 4 holds the finite state machine of the library phrases. Figure 4 is drawn according to the above samples.

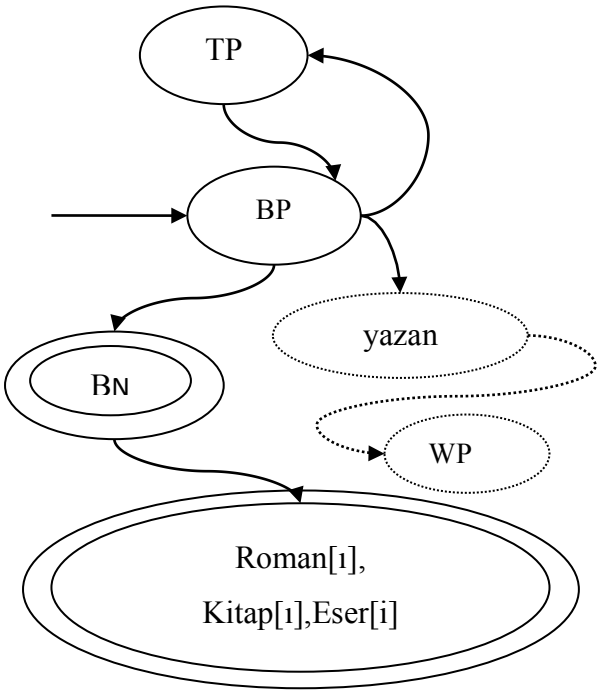

Fig. 5. FSM of Book Phrases

Figure 5 holds the book phrases and the finite state machine is drawn according to the above samples.

A book phrase can be translated to the writer phrase by the keyword "yazan"*(writer)*. In this case the database search is executed to translate the book phrase to the writer phrase.

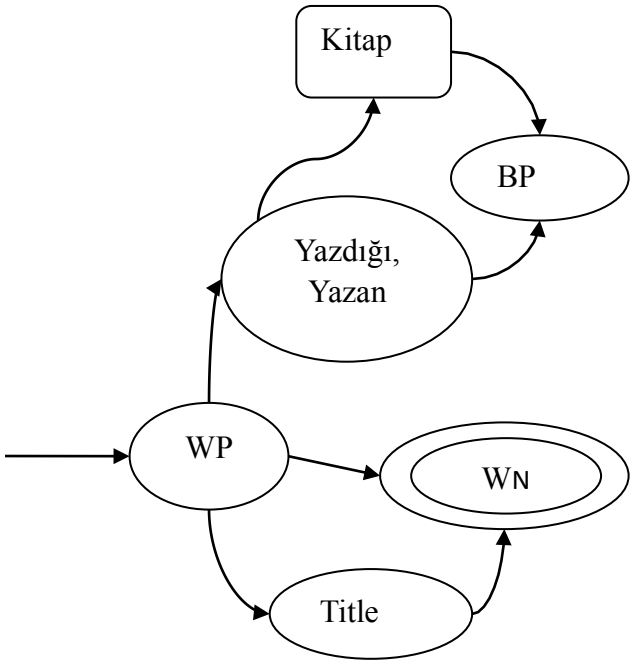

Fig. 6. FSM of Writer Phrases

Above figure holds the writer phrases and the possible variations of the writer phrases. A writer phrase can be translated into the Book Phrase by adding several words besides the writer phrase.

Also a title can be appended before writer noun as a defining statement.

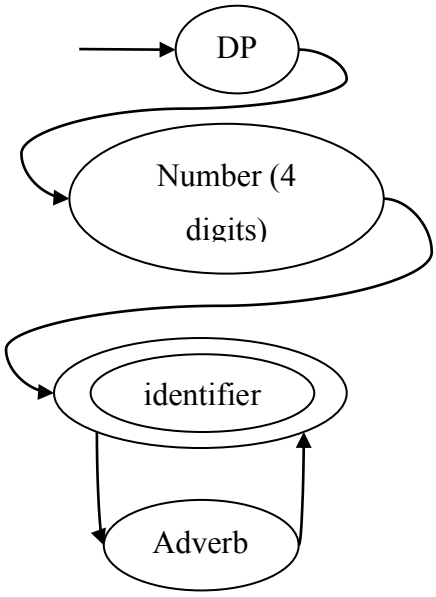

Fig. 7. FSM of Date Phrases

Above figure holds the date phrases of the possible alternatives. A date phrase should contain a number with 4 digits. Also an identifier or an alternative adverb can be added after the date phrases. For example in a date phrase like "1976 yılında yayınlanan" *(in the year 1976)* the number 1976 is considered as a number and "yılında" is considered as an identifier. The word "yayınlanan" is an adverb identifying the date.

A question phrase is a frozen list of words. The list consists of the above grammar, which is also quoted below:

```
QP>  [kaç  [a|da|ı|ncı]  |nere  [de|ye]
|nasıl|ne zaman|ne kadar|ne|hangi [si]]
```

Also a translation phrase is again a list of frozen words. Such as "ingilizcesi, almancası, rusçası".

Verb phrases and the noun phrases are consist of the morphological analysis.

2.7 Database querying phase

In this phase of maturity the database queries are executed in the YACC file.

The query details and the implementation of functions will be explained in the implementation chapter. This sub section covers the details of the Rules of the YACC file with respect to the database querying phase.

Above rules are designed for the database querying phase. Each of the above finite state machines represents a word group in the syntactic analysis.

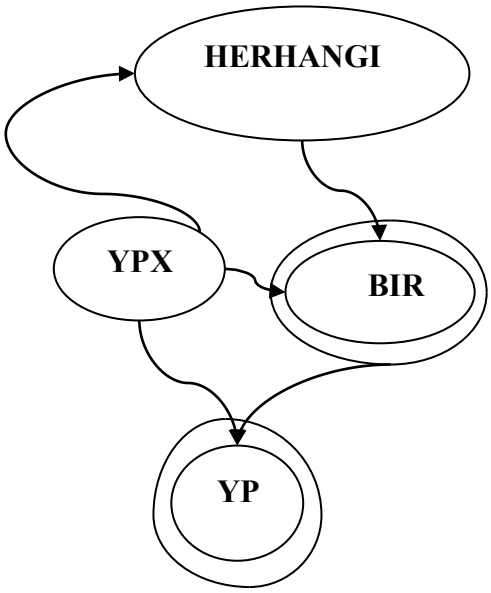

Fig. 8. Finite State machine of writer phrases

Above figure holds the possibilities of the writer phrase. The writer can be a writer name (YP in above figure) or can take any of the words “Herhangi” or “Bir”.

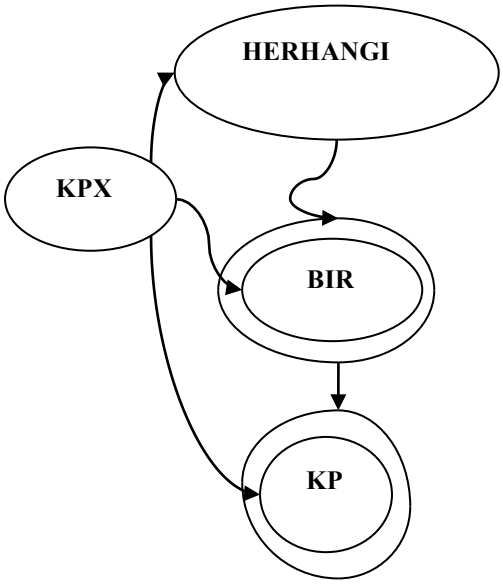

Fig. 9. Finite State machine of book phrases

Above figure is similar to the writer phrase the only difference is the replacement of the writer with the book.

The book name can again get any of the words “Herhangi” or “Bir”.

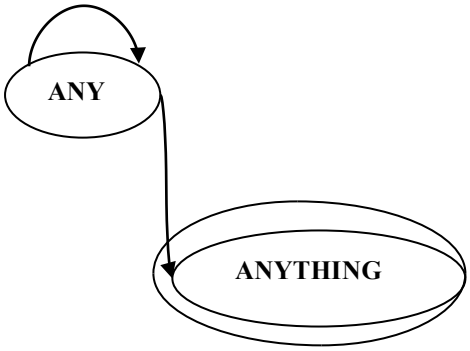

Fig. 10. FSM of Writer and book names

Writer and book names should end by ANYTHING which is a lex entry in the LEX file. And also the lex entry can get any number of pre words.

```
Orhan
Orhan Veli
Orhan Veli Kanık
```

For example all the above writer entries should be fetched by the above rule. So the above rule is implemented flexible to the number of words.

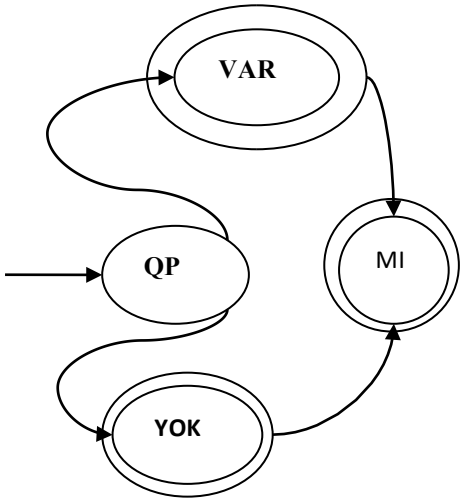

Fig. 11. FSM of Question phrase

Above finite state machine holds the accepted question phrases of existence. The user can query whether something exists or not by the above FSM.

The final view of the accepted phrases is given below:

2.8 Database Design

This chapter covers the details of the database design and implementation. MySQL [6] is selected for the database management system because of its flexible and easy to maintanance behaviour. Also this project is database technology independent, so any other technology can be easily substituted with the MySQL technology.

Below is, the database scheme of the required tables and the columns of each table.

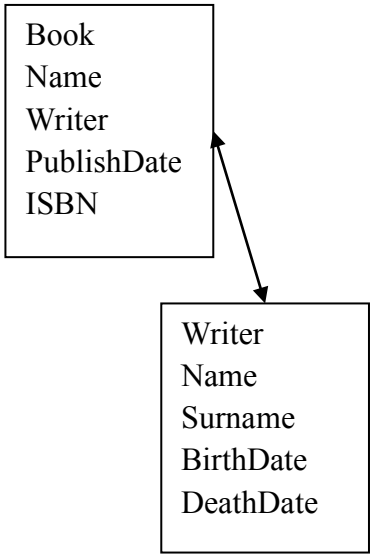

Fig. 12. Database scheme of the book automation.

It is obvious that the above scheme is a simple databas structure with two tables. The database structure keeps the relation between writer and the books. Also the cardinality of the relation between books and the writers does not provide a many-to-many relation. The relation is a simple one-to-many relation. For example a book can be written by only 1 writer. In the case of a book with multiple writers, the system does not handle such cases.

3 Conclusion

During this project, ZEMBEREK has implemented using JAVA technology on both LINUX and Windows environments and because of the problems on connection between YACC, LEX and Zemberek this technology is solved by implementing regular expressions over LEX file.

The LEX file input parser and regular expressions are implemented and modified for the Turkish character set and the connection between LEX and YACC is solved by carrying out these Turkish input characters.

Another technological achievement during this project is implementation of MySQL over the LINUX with Turkish input. Also creating connection to the MySQL via C programming was another problem and solved by installing the MySQL source packages on the LINUX.

Finally the project is capable of running the LEX, YACC, MySQL and C codes compiled together with Turkish input support.

After these technological installation problems the implementation problems has solved by writing several query functions. Also another achievement is reached by adding several input sentence patterns to the YACC file. YACC file keeps the project flexible because of its input pattern support. The LEX file keeps the input flexible by its regular expression support and by adding the flexibility of the LEX to the YACC the project can almost cover any input sentence in Turkish. The only problem of such flexible environment is preventing one of the rules already inserted by adding another rule to the LEX or YACC files.

Another problem related to the flexibility was the database records. The book names on database can keep almost any of the inputs including the frozen words in a YACC input pattern. The problem is solved by adding a flexible recursive YACC rule.

Besides the above technological and designing achievements this project was also an opportunity for me to implement my theoretical studies during undergraduate to a real life project.

For example management of this project is done parallel to the linear model in Software engineering project. The programming skills from introduction to programming and data structure courses helped me to implement the string manipulations' and the data structures implemented during the functions of YACC.

Also the installation and management of modules over LINUX operating system has been achieved in operating systems courses. Rss3@nz

ppdif

Besides the technological and design achievements and the opportunity to practice the above courses I have implemented a relatively large scale project first time. I hope this experience will help me in the future engineering cases.

ACKNOWLEDGMENT

This project has been supported by the research department of Istanbul University, project number YADOP-16728.

4 References

[1] On the Theoretical Foundation of Meta-Modelling in Graphically Extended BNF and First Order Logic Hong Zhu Theoretical Aspects of Software Engineering (TASE), 2010 4th IEEE International Symposium on Publication Year: 2010 , Page(s): 95 - 104

[2] Arabic Morphological Analysis: a New ApproachSonbol, R.; Ghneim, N.; Desouki, M.S. Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on Digital Object Identifier: 10.1109/ICTTA.2008.4530014 Publicat ion Year: 2008 , Page(s): 1 - 6

[3] Simple calculator compiler using Lex and YACC Upadhyaya, M. Electronics Computer Technology (ICECT), 2011 3rd International Conference on Volume: 6 Digital Object Identifier: 10.1109/ICECTECH.2011.5942077 Pu blication Year: 2011 , Page(s): 182 - 187

[4] Ontological queries: Rewriting and optimization Gottlob, G.; Orsi, G.; Pieris, A. Data Engineering (ICDE), 2011 IEEE 27th International Conference on Digital Object Identifier: 10.1109/ICDE.2011.5767965 Publicati on Year: 2011 , Page(s): 2 - 13

[5] Information retrieval from turkish radiology reports without medical knowledge Kerem Hadımlı, Meltem Turhan Yöndem FQAS'11: Proceedings of the 9th international conference on Flexible Query Answering Systems, October 2011

[6] MySQL: lessons learned on a digital library Di Giacomo, M. Software, IEEE Volume: 22 , Issue: 3 Digital Object Identifier: 10.1109/MS.2005.71 Publication Year: 2005 , Page(s): 10 - 13