Design and Development of Deterministic Finite Automata Parser for Querying Hardware and Software Configuration Information of Local Area Network

Mr. Santosh Patil
Research Student
Chh Shahu Institute of Business Education and Research,
Kolhapur, India
santoshgpatil111@gmail.com

Dr. Poornima G. Naik
Professor, Department of Computer Studies
Chh Shahu Institute of Business Education and Research, line
Kolhapur, India
luckysankalp@yahoo.co.in

Abstract— A typical Local Area Network (LAN) of an educational institution hosts different hardware devices and contains numerous softwares installed, serving the needs of persons from low technical expertise to high technical standards. Further as the academic curriculum is upgraded most of the existing softwares become obsolete and new softwares are installed or existing softwares are upgraded to new version to cater the needs which is a continuous process. To address such issues, a quick and reliable snapshot of the network configuration is desirable at any point of time. On many occasions the softwares with limited usage are installed only on few machines of a LAN. Thus, it proves to be a time consuming task to search the whole network for a single rarely used software. To tackle such issues the authors in the current paper have provided a first hand tool for automating the process of discovering LAN configuration, storing it persistently and querying the stored information in human language. The entire process is automated without any human intervention. The information pertaining to the hardware and software configuration is stored in a persistent Relational DataBase Management System (RDBMS) which can be manipulated by the tool automatically as the new hardware is connected to a LAN or a software configuration changes. The end user instead of querying the database directly will use the natural language, termed as Hardware Query Language (HQL) and Software Query Language (SOQL) designed by the authors, which is interfaced with RDBMS using DFA parser implemented by the authors. To implement HQL/SOQL, a finite set of symbols, words and language rules are defined which together constitute HQL/SOQL grammar. In this paper we present a deterministic finite automata (DFA) parser developed by us for parsing HQL/SOQL tokens. The state table and state diagrams are developed for different tokens of HQL/SOQL identified by us. State information is stored in a persistent database management system as a measure towards improving efficiency and extensibility. Currently, HQL/SOQL consists of only few commands and more commands will be added to HQL/SOQL command set in near future. The reports generated are exported to MS Word using Microsoft Word 12.0 object library.

Keywords*- Deterministic Finite Automata, Hardware Query Language, Object Library, Software Query Language, State Diagram, StateTable*

*****

I. INTRODUCTION

Every educational institution is equipped with a local area network that interconnects computers within a limited area which has made greater inroads into every educational premises. Neworks offer tremendous advantages among which data and resource sharing have gained a tremendous importance. The institutions face enormous challenges developing and maintaining infrastructures that keep pace with the demands of today's high-tech society. New softwares are continuously emerging and the existing softwares are continuously upgraded to newer versions. Besides supporting administrative and faculty requirements educational institutions must have the appropriate technology in place to prepare students to work and learn effectively. Further the Network security consists of the set of policies adopted for preventing and monitoring an unauthorized access, misuse, modification, or denial of a computer network and network-accessible resources. Network security involves the authorization of access to data in a network, which is controlled by the network administrator. Network security mechanism apart from securing the network protects and oversees operations being done.

The intent of our research is to design and develop an interface for LAN which accepts the queries pertaining to hardware and software installed in LAN in natural language (NL) which is parsed using DFA parser which is mapped to an SQL query and instantly provides a required information to an end user. The information pertaining to the machine, hardware and software information is stored in a persistent Relational DataBase Management System (RDBMS) which is dynamic and is instantly updated as the new hardware is connected to LAN or a new software is installed. The end user instead of querying the database directly will use the natural language, termed as Hardware Query Language (HQL) and Software Query Language (SOQL) designed by us, which is interfaced with RDBMS using VB. To implement HQL and SQL, we have defined a finite set of symbols, words and language rules, HQL and SOQL grammar. The state diagram and state tables are constructed based for the grammar specified. The human query is parsed using DFA parser designed by us and the queries which are successfully parsed will be evaluated by mapping them to the corresponding SQL query using Java interface to VB.

Significance of the Study

Our research mainly targets the following issues.

- Design and development of GUI interface which help to lab technicians for solving hardware and software queries related to LAN.
- To store and manage all hardware and software information of devices connected to LAN in a centralized database in MySQL.
- To detect and avoid IP conflicts common in networks
- .Bandwidth Management
- Monitoring speed of internet and pattern recognition to search anomalies.

- Authentication modules for different levels of network users.

Objectives of the Study

The main objective of the study is to Design and Develop NLP Interface for querying hardware and software configuration information in local area network for selected education institute.

- To dynamically discover the LAN architecture and list various computers in a workgroup and domain controller.
- To dynamically discover the various hardwares connected to LAN and softwares installed on various machines and store the same persistently in a centralized database.
- To design and develop DFA interface that displays queried information of all hardware and software in LAN.
- To design and develop NLP interface that displays queried information of all hardware and software in LAN.
- To design and develop NLP parser which helps in evaluating a query issued by an end user in a human language and mapping it to a SQL query.
- To detect IP conflicts in a network.
- To continuously monitor the speed of Internet and analyze the data for detecting patterns and solving network bottlenecks.

The research is under progress and in this paper, we are presenting a firsthand tool conforming to the first three objectives only.

Deterministic Finite Automata

Deterministic Finite Automata (DFA) can be seen as a special kind of finite state machine, which is in a sense an abstract model of a machine with a primitive internal memory. It is a finite state machine that accepts/rejects finite strings of symbols and only produces a unique computation (or run) of the automaton for each input string. 'Deterministic' refers to the uniqueness of the computation.

A deterministic finite automaton M is a 5-tuple, $(Q, \Sigma, \delta, q_0, F)$, consisting of A finite set I of input symbols.

i. A finite set of states (Q)
ii. A finite set of input symbols called the alphabet, Σ
iii. A transition or next state function δ, $\delta : Q \times \Sigma \rightarrow Q$
iv. A subset F of Q of accept or final states, $(F \subseteq Q)$
v. An initial or start state $(q_0 \in Q)$.

Let $w = a_1a_2 \ldots a_n$ be a string over the alphabet Σ. The automaton M accepts the string w if a sequence of states, $r_0, r_1, \ldots, r_n$, exists in Q with the following conditions:

$r_0 = q_0$

$r_{i+1} = \delta(r_i, a_{i+1})$, for $i = 0, \ldots, n-1$ where, $r_n \in F$.

In words, the first condition says that the machine starts in the start state q_0. The second condition says that given each character of string w, the machine will transition from state to state according to the transition function δ. The last condition says that the machine accepts w if the last input of w causes the machine to halt in one of the accepting states. Otherwise, it is said that the automaton rejects the string. The set of strings M accepts is the language recognized by M and this language is denoted by L(M).

II. LITERATURE REVIEW

In literature there exit numerous papers on natural language processing applied to various areas to reduce the gap between human and machine languages [1,9] One of the authors of this paper in involved in designing DFA and NLP parser for parsing manufacturing query language tokens [10,12]. In their work the authors have parsed the NLP query using NLP parser designed by them and the queries which are successfully parsed are evaluated by mapping them to the corresponding prolog query using Java interface to Prolog (JPL). Prolog rules are stored in three different prolog knowledge bases, mqlgrammar.pl, rules.pl, and methodrules.pl. NLP offers most flexible way to implement grammar which can be readily extended with least efforts and as such offers an efficient way of implementing rules in dynamically changing scenarios. The authors of paper [13] have presented the first unsupervised approach for semantic parsing that rivals the accuracy of supervised approaches in translating natural-language questions to database queries. Their system produces a semantic parse by annotating the dependency-tree nodes and edges with latent states, and learns a probabilistic grammar using EM. To compensate for the lack of example annotations or question-answer pairs, GUSP adopts a novel grounded-learning approach to leverage database for indirect supervision. On the challenging ATIS dataset, GUSP attained an accuracy of 84%, effectively tying with the best published results by supervised approaches semantic parsing for natural-language interface to database [14]. In this problem setting, a natural language question is first translated into a meaning representation by semantic parsing, and then converted into a structured query such as SQL to obtain answer from the database. Yukiko Sasaki Alam [15] describes a parser in progress which is directed to generating representations for text understanding. For the purpose of reducing the proliferation of unwanted parse trees, and collecting information necessary for generating the semantic representations, the parser uses rules based on phrasal and lexical subcategories. These designs alleviate parsing problems such as PP attachment and coordination attachment, while capable of displaying the dependency of various types of phrases and clauses, thus facilitating the writing of grammar.

III. CONCEPTUAL FRAMEWORK

Application Framework

- A framework is designed to cater the following needs.
- For storing all information of hardware and software present in LAN.
- For searching required software and hardware present in LAN.
- For generating various reports related to LAN.
- For developing multiple interfaces for desktop or android based mobile end users.

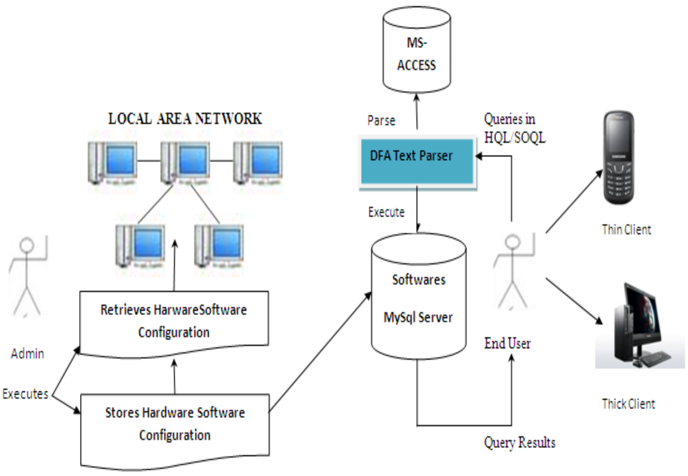


Figure 1. Application Framework

The corresponding layered architecture is depicted in Figure 2.

End User
HQL/SOQL in Natural Language
Prolog Query
SQL Query
Physical Database

Figure 2. Layered Application Architecture

Proposed Algorithm

```
/* Algorithm in C-Style */

/*
    Every high-level language has built-in string manipulation
functions present in a string library. The following functions
assume the existence of the following string manipulation
functions.
instr() – Accepts two string arguments and returns the position
of the second string within a first string, if the string is not
found returns -1.
   Right() – Accepts two arguments of type string and int,
respectively and returns a substring of a string passed as the
first argument containing  rightmost n characters passed as the
second argument to a function.
*/
char words[10][10];
int cntWords;
char syntax[10];
char query[50];
/*Any high level language interfacing with back end database
management system provides high level API for primitive
database functions such as creating a connection object,
checking the current position of the resultset pointer and
selecting a set of rows based on the given criterion. Hence this
algorithm assumes some standard functions as shown below:
Standard Functions used in the Algorithm
          getConnection - is a built-in function returning
Connection object for a given connection string
          getResultSet()  -   is a built-in function returning a
resultset containing query results;
.
          isEOF(ResultSet) -  is a function which returns a
boolean value, indicating whether the resultset
                              pointer is at the beginning of or at
the end of  resultset. */

 /*    conString : Connection String for connecting to a back
end. */
function parse()
{
          read sentence;
          cntWords=count_words(sentence);
         split_words(sentence);

          syntax="correct";
         con=getConnection(conString);
          for (i=0;i<=cntWords;i++)
          {
            query="SELECT * FROM  tokens WHERE level =
" + (i+1);
           resultSet=getResultSet(con,query);
           if ((isEOF(resultSet) == false)
            {
               syntax="Incorrect";
               break;
            }
          }
          if (syntax="correct")
                    print "Parsed Successfully...";
          else
                    print "Syntax Error!";
}

function int count_words(char sentence[10])
{
          int count=0;
          int pos;
          pos=instr(sentence," ");
while (pos != -1)
          {
                    count++;
                    sentence=right(sentence,pos+1);
                    pos=instr(sentence," ");
          }
return count;
}

function split_words(char sentence[10])
words=sentence.split(" ");
}
```

Control Flow Diagram

Figure 3. represents a control flow diagram depicting the brief working of DFA parser.

 ISSN: 2321-8169
5896 - 5903

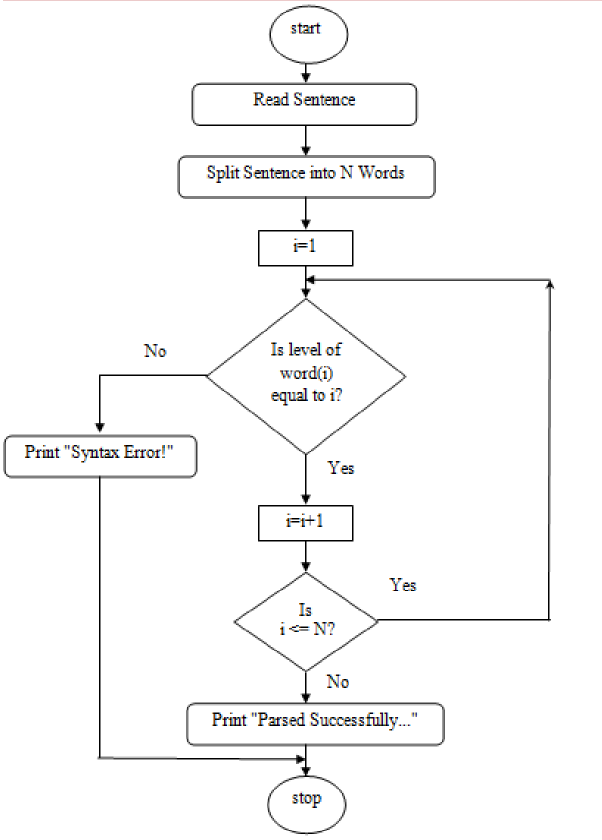


Figure 3. Control Flow Diagram for the working of DFA Parser

Grammar for HQL and SOQL.

To implement HQL and SOQL, we have constructed a language by defining the rules which specify how to test a string of alphabet letters to verify. A finite set of symbols used in the language is given by

Σ = {a, b, c, d, e, f, h, i, k, m, n, o, p, r, s, t, u, v, w, x, y}

and a set of words over an alphabet is given

L={ are, brands, capacity, different, disk, hard, has, is, machine, maximum, of, os, processor, ram, speed, version, what, where, which}

General syntax of HQL/SOQL Commands

Some sample HQL/SOQL commands are given below:

1. Where is *?
2. Which machine has maximum ram?
3. Which machine has maximum harddisk capacity?
4. Which machine has maximum processor speed?
5. What are different OS?
6. What is a version of *?
7. What are different machine brands?
8. What are different processor types?

where * substitutes for any software name.

State Table and State Graph for Manufacturing Query Language.

DFA is a set S of states that are connected by function f. A transition is an event of going from one state to another. DFAs are represented in two formats. Table and Graph. Table representation of DFA for HQL/SOQL queries is shown in Table I.

TABLE I. STATE TABLE FOR HQL/SOQL QUERIES

Input States	WHERE	IS	*	WHICH	MACHINE	HAS	MAXIMUM	RAM	HARDDISK	CAPCITY	PROCESSOR	SPEED
S_0	S_1			S_4								
S_1		S_2										
S_2			$\{S_3\}$									
S_4					S_5							
S_5						S_6						
S_6							S_7					
S_7								$\{S_8\}$	S_{11}		S_9	
S_9												$\{S_{10}\}$
S_{11}										$\{S_{12}\}$		
S_{16}			$\{S_{17}\}$									
S_{19}					S_{20}						S_{22}	

Input States	WHAT	IS	VERSION	OF	ARE	DIFFERENT	BRANDS	TYPES
S_0	S_{13}							
S_1								
S_2								
S_{16}								
S_4								
S_5								
S_6								
S_7								
S_9								
S_{11}								
S_{13}		S_{14}			S_{18}			
S_{14}			S_{15}					
S_{15}				S_{16}				
S_{18}						S_{19}		
S_{20}							$\{S_{21}\}$	
S_{22}								$\{S_{23}\}$

An equivalent state graph is shown in Figure 4.

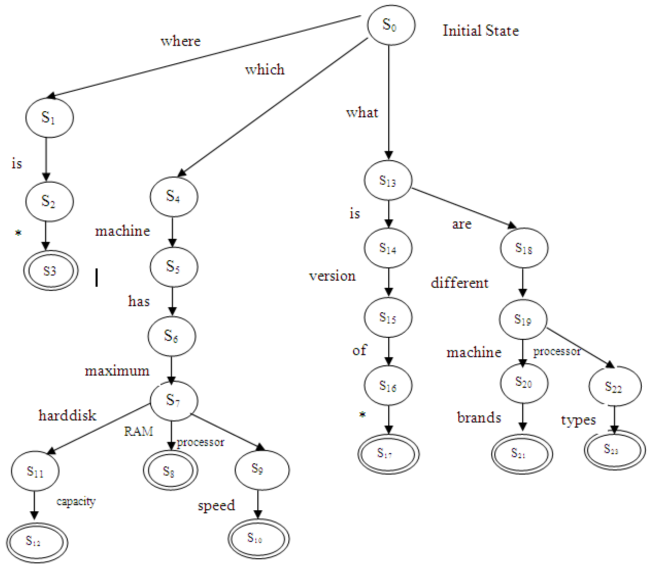


Figure 4 State Graph for HQL/SOQL Query

Figure 4 illustrates a deterministic finite automaton using a state diagram for the List query. In the automaton, there are twenty two states: S0, S1… S22 (denoted graphically by circles).S0 is an initial state of the state diagram. The automaton takes a finite sequence of 0s and 1s as input. The automaton takes a finite sequence of strings as input. For each state, there is a transition arrow leading out to a next state on accepting the input. Upon reading a string, a DFA jumps deterministically from the current state to another by following the transition arrow. For example, if the automaton is currently in state S0 and if current input string is "where" , then it deterministically jumps to state S1 . A DFA has a start state (denoted graphically by an arrow coming in from nowhere) where computations begin, and a set of accept states (denoted graphically by a double circle) which help define when a computation is successful and the syntax is correct.

The state graph shown in Figure 5. depicts the transition from initial or start state to final state on consuming tokens.

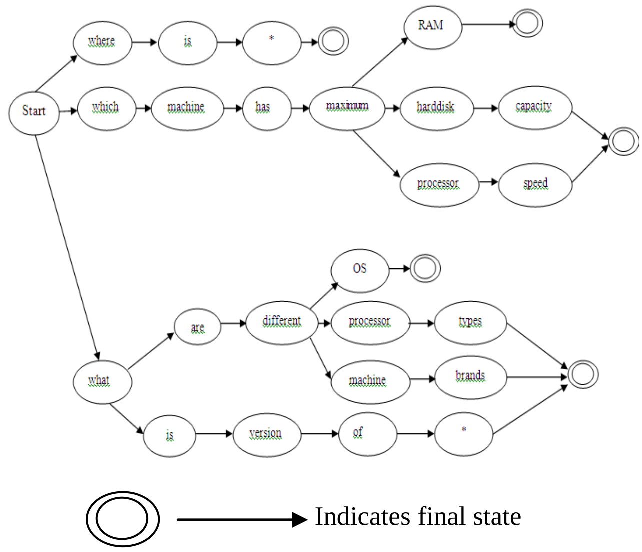


Figure 5. State Graph for HQL/SOQL

Table II classifies different HQL/SOQL tokens into different levels.

TABLE II. CLASSIFICATION OF HQL/SOQL TOKENS INTO DIFFERENT LEVELS QUERIES

LEVEL	1	2	3	4	5	6
	where	Is	*?			
	which	machine	has	maximum	ram?	
	which	machine	has	maximum	harddisk	capacity?
	which	machine	has	maximum	processor	speed?
	what	are	different	os?		
	what	is	version	of	*?	
	what	are	different	machine	brands?	
	what	are	different	processor	types?	

Table III summarizes the grouping of different tokens according to their level.

TABLE III. GROUPING OF TOKENS ACCORDING TO LEVEL

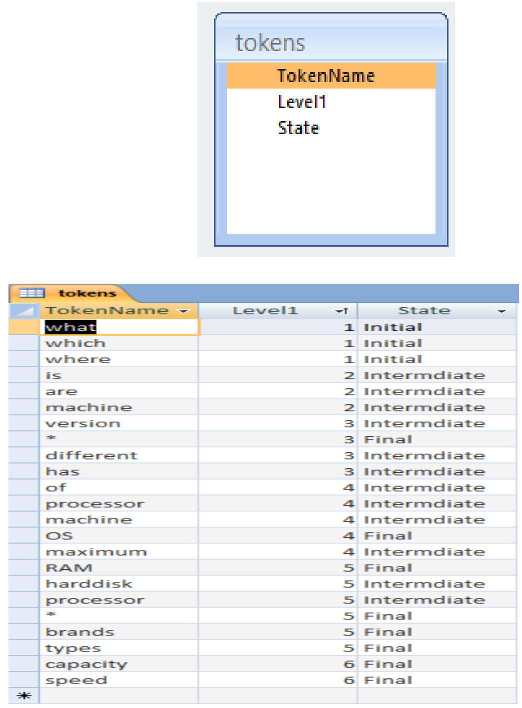
LEVEL	TOKENS
1	what, where, which
2	are, is, machine
3	*?, different, has, version
4	machine, maximum, of, os, processor
5	*?, brands, harddisk, processor, ram?, types?
6	capacity?, speed?

IV. RESULTS AND ANALYSIS

The model developed above is implemented in VB, Java and MySQL. The tokens are stored in MS-Access database along with the level and state information. Level corresponds to the position of the token in a state graph while state can contain one of the following values.

- Initial
- Intermediate
- Final

Figure 6. depicts MS-Access database containing a single table for parsing HQL/SOQL query along with some sample data.



TokenName	Level1	State
what	1	Initial
which	1	Initial
where	1	Initial
is	2	Intermdiate
are	2	Intermdiate
machine	2	Intermdiate
version	3	Intermdiate
*	3	Final
different	3	Intermdiate
has	3	Intermdiate
of	4	Intermdiate
processor	4	Intermdiate
machine	4	Intermdiate
OS	4	Final
maximum	4	Intermdiate
RAM	5	Final
harddisk	5	Intermdiate
processor	5	Intermdiate
*	5	Final
brands	5	Final
types	5	Final
capacity	6	Final
speed	6	Final

Figure 6. Structure of Parser Database along with Sample Data

The structure of the database for storing hardware and software configuration information in MySQL database is depicted in Figure 7.

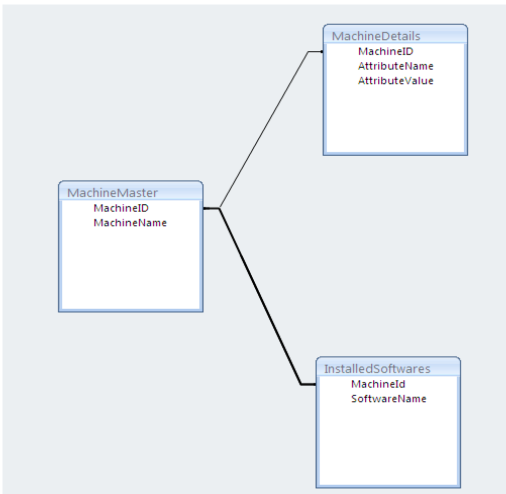
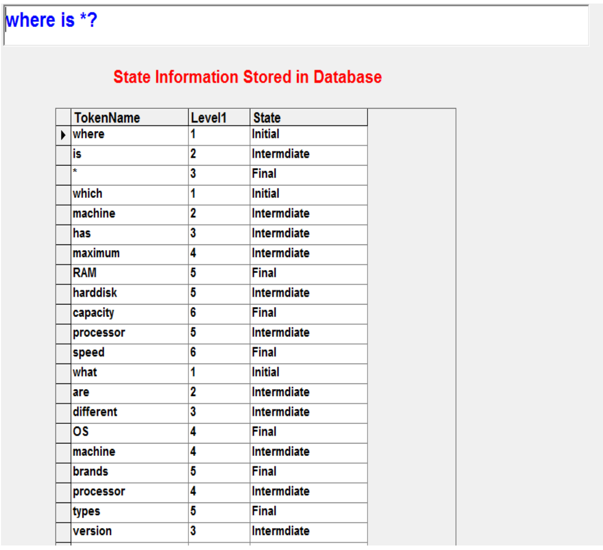


Figure 7. Structure of MySQL Database for Storing Hardware/Software Configuration

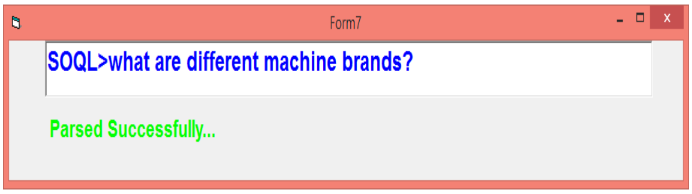
Figure 8. shows splitting of given sentence into different tokens, assigning the level and state information to each of them and storing them in a database persistently for future use.

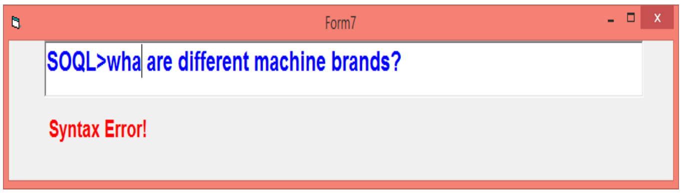


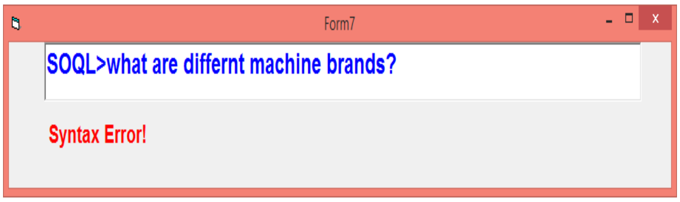
TokenName	Level1	State
where	1	Initial
is	2	Intermdiate
*	3	Final
which	1	Initial
machine	2	Intermdiate
has	3	Intermdiate
maximum	4	Intermdiate
RAM	5	Final
harddisk	5	Intermdiate
capacity	6	Final
processor	5	Intermdiate
speed	6	Final
what	1	Initial
are	2	Intermdiate
different	3	Intermdiate
OS	4	Final
machine	4	Intermdiate
brands	5	Final
processor	4	Intermdiate
types	5	Final
version	3	Intermdiate

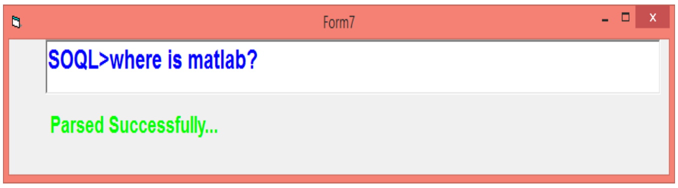
Figure 8. Storing State Information in a Database

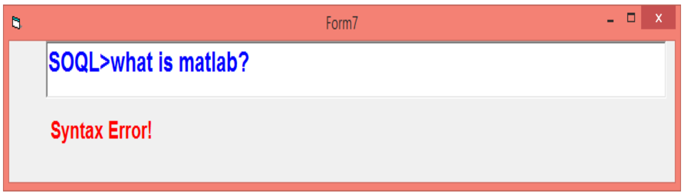
Figures 9 (a) - 9 (h) show the results of parsing of HQL/SOQL queries by DFA parser

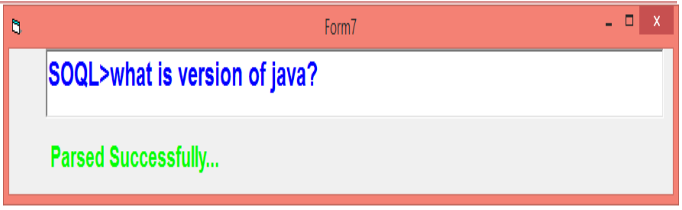












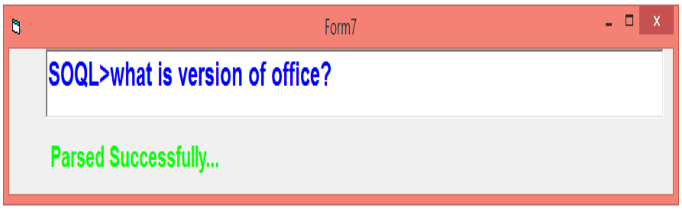
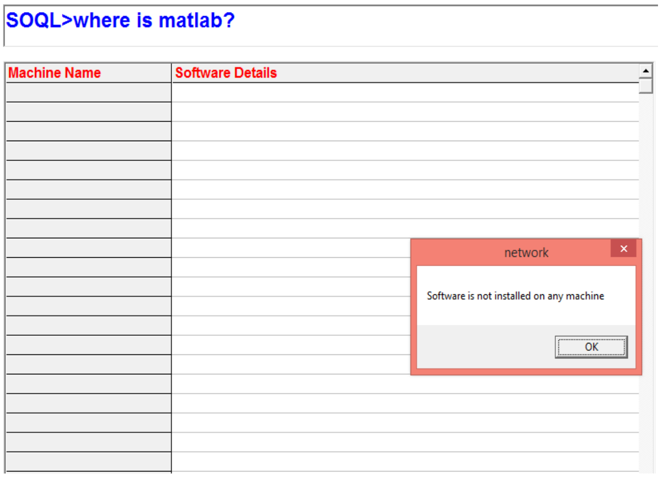
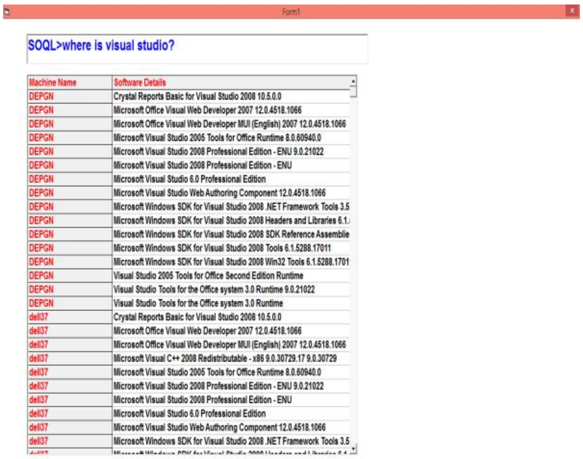


Figure 9(a) – 9 (h) Parsing of HQL/SOQL queries by DFA parser.

Figures 10 (a) – 10 (h) show the results of execution of HQL/SOQL queries by mapping them to the corresponding SQL queries.





 ISSN: 2321-8169
5896 - 5903

SOQL>what are different OS?

OS Name
Microsoft Windows XP Multiprocessor Free

SOQL>which machine has maximum ram?

Machine Name	Physical Memory
DEPGN	4024 MB

SOQL>which machine has maximum processor speed?

Machine Name	Processor Speed
DEPGN	2.9GHz

SOQL>what is version of java?

Machine Name	Software Details
DEPGN	Java (TM) 7 1.7.0.0
DEPGN	Java Auto Updater 2.0.2.4
DEPGN	Java DB 10.5.3.0 10.5.3.0
DEPGN	Java(TM) 6 Update 21 6.0.210
DEPGN	Java(TM) SE Development Kit 6 Update 21 1.6.0.210
DEPGN	Java(TM) SE Development Kit 7 1.7.0.0
dell37	Java (TM) 7 1.7.0.0
dell37	Java Auto Updater 2.0.2.4
dell37	Java DB 10.5.3.0 10.5.3.0
dell37	Java(TM) 6 Update 21 6.0.210
dell37	Java(TM) SE Development Kit 6 Update 21 1.6.0.210
dell37	Java(TM) SE Development Kit 7 1.7.0.0

SOQL>what are different machine brands?

Machine Brand	Count
DellInc.	4
HP	1

Figure 10 (a) – 10 (h). Execution of HQL/SOQL queries.

The reports are dynamically generated in VB using Microsoft Word Object Library 12.0 which is shown in Figure 11. The partial code for achieving this is listed in Appendix A.

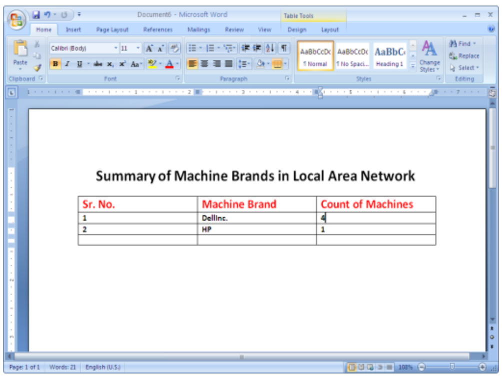


Figure 11. HQL/SOQL Query Result Exported to MS Word.

V. Conclusion and Scope for Future Work

In the current work authors have designed and implemented a DFA parser for querying the hardware and software configuration information stored in a centralized MySQL database. The end user instead of querying the database directly will use the natural language, termed as Hardware Query Language (HQL) and Software Query Language (SOQL) designed by the authors, which is interfaced with RDBMS using DFA parser implemented by the authors. To implement HQL/SOQL, a finite set of symbols, words and language rules are defined which together constitute HQL/SOQL grammar. In this paper we present a deterministic finite automata (DFA) parser developed by us for parsing HQL/SOQL tokens. The state table and state diagrams are developed for different tokens of HQL/SOQL identified by us. State information is stored in a persistent database management system as a measure towards improving efficiency and extensibility. The parser is tested for few HQL/SOQL queries and the language is easily extensible for incorporating more queries in the knowledge database.

Our future work focuses on designing a supervised neural network for parsing the HQL/SOQL queries into two different classes "correct" and "incorrect" based on the input pattern.

References

[1] Saif Mohammad, Bonnie Dorr, and Graeme Hirst, 2008. Computing word-pair antonymy. In Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing, pages 982–991, Honolulu, HI. ACL.

[2] Klein and C. D. Manning. Natural language grammar induction using a constituent-context model. In Advances in Neural Information Processing Systems (NIPS 14), pages 35–42. 2002.

[3] Kudo and Y. Matsumoto. Chunking with support vector machines. In Conference of the North American Chapter of the Association for Computational Linguistics & Human Language Technologies (NAACL-HLT), pages 1–8, 2001.

[4] Mnih and G. E. Hinton. Three new graphical models for statistical language modeling, International Conference on Machine Learning (ICML), pages 641–648, 2007.

[5] Natural Language Processing (Almost) from Scratch, Journal of Machine Learning Research 12 (2011) 2493-2537, Ronan Collobert, JasonWeston, L´eon Bottou, Michael Karlen , Koray Kavukcuoglu, Pavel Kuksa

[6] Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Struc-tured classification with probabilistic categorial grammers. In Proceedings of the Twenty FirstConference on Uncertainty in Artificial Intelligence pages 658–666, Edinburgh, Scotland. AUAI Press.

[7] Luke S. Zettlemoyer and Michael Collins. 2007. On-line learning of relaxed ccg grammars for parsing to logical form. In Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning

[8] Booth, Taylor L. & Richard A. Thompson (1973) Applying probability measures to

[9] abstract languages. IEEE Transactions on Computers C-22.5: 442–450.

[10] Two Strategies for Text Parsing, Joakim Nivre, CSLI Publications, University of Chicago Press

[11] Mr. Girish R. Naik, Dr. V.A.Raikar, Dr. Poornima G. Naik, Implementation of DFA Parser for Manufacturing Query Language Tokens, International Journal Of Engineering Sciences & Research Technology, Vol. 4 Issue 1, ISSN: 2277-9655, pages 371-382.

[12] Mr. Girish R. Naik, Dr. V.A.Raikar, Dr. Poornima G. Naik, Multi Objective Criteria for Selection of Manufacturing Method, International Journal of Advanced Research in Computer

Science and Software Engineering, Volume 4, Issue 7, July 2014, ISSN: 2277 128X, pages 989-1002.

[13] Mr. Girish R. Naik, Dr. V.A.Raikar, Dr. Poornima G. Naik, Multi Objective Criteria for Selection of Manufacturing Method using NLP Parser, International Journal on Recent and Innovation Trends in Computing and Communication, Volume: 2 Issue: 11, ISSN: 2321-8169, pages 3484-3493.

[14] Hoifung Poon Pedro Domingos, Unsupervised Semantic Parsing.

[15] B.J. Grosz, D. Appelt, P. Martin, and F. Pereira. 1987. Team: An experiment in the design of transportable natural language interfaces. Artificial Intelligence,32:173–243.

[16] Yukiko Sasaki Alam, A Subcategory-based Parser Directed to Generating Representations for Text Understanding, Procedia - Social and Behavioral Sciences, Volume 27, 2011, Pages 194–201.

Appendix A

Accessing Word Object Libray in VB

```
Set oWord = CreateObject("Word.Application")
    oWord.Visible = True
    Set oDoc = oWord.Documents.Add

    'Insert a paragraph at the beginning of the document.
    Set oPara1 = oDoc.Content.Paragraphs.Add
    oPara1.Range.Font.Size = 20
    oPara1.Range.Text = "Summary of Machine Brands in
Local Area Network"
    oPara1.Range.Font.Bold = True
    oPara1.Format.SpaceAfter = 24   '24 pt spacing after
paragraph.
    oPara1.Range.InsertParagraphAfter


Set oTable =
oDoc.Tables.Add(oDoc.Bookmarks("\endofdoc").Range,
4, 3)
oTable.Range.ParagraphFormat.SpaceAfter = 6

' set table border
oTable.Borders.Enable = True

oTable.Rows(1).Range.Font.Size = 16
oTable.Rows(1).Range.Font.Color = vbRed
oTable.Rows(1).Alignment = wdAlignRowCenter

'  set table heading

oTable.Cell(1, 1).Range.Text = "Sr. No."
oTable.Cell(1, 2).Range.Text = "Machine Brand"
oTable.Cell(1, 3).Range.Text = "Count of Machines"
Open "vendors.txt" For Input As #1
cnt = 0
While (EOF(1) = False)
 Input #1, count1, mname
 cnt = cnt + 1
 oTable.Cell(cnt + 1, 1).Range.Text = CStr(cnt)
 oTable.Cell(cnt + 1, 2).Range.Text = mname
 oTable.Cell(cnt + 1, 3).Range.Text = count1
Wend
Close #1
```