Intelligent Agent-based Multilingual Information Retrieval System

Sumam Mary Idicula David Peter S

Abstract

The goal of this work is to develop an Open Agent Architecture for Multilingual information retrieval from Relational Database. The query for information retrieval can be given in plain Hindi or Malayalam; two prominent regional languages of India. The system supports distributed processing of user requests through collaborating agents. Natural language processing techniques are used for meaning extraction from the plain query and information is given back to the user in his/ her native language. The system architecture is designed in a structured way so that it can be adapted to other regional languages of India.

Keywords : Information Retrieval, Natural Language Processing, Multilingual Computing, Indian Languages.

0. Introduction

Information is playing an increasingly important role in our lives. Information has become an instrument that can be used for solving problems. An intelligent information agent is one that acts on behalf of its user for information gathering [1]. It is capable of locating information sources, accessing information, resolving inconsistencies in the retrieved information and adapting to human information needs over time. An information agent is responsible for providing intelligent information services. Information market has now been transformed from supply-driven to demand-driven.

India being a multilingual country, information agents have to work in multilingual environment. To handle multilingual environment, specific requirements and good language knowledge is needed. Every language has a set of language resources like characters, lexicons, grammars and keyboard mappings. Agent should be open to updates in language resources. An information agent should be able to configure to the user preferred language and usage style. An agent should exhibit the capacity to learn new languages to increase the degree of multilingualism.

A multilingual information agent should have properties like

- Language autonomy- capability to control state and behavior according to the language.
- Reactivity : The ability to perceive and respond to multilingual percepts.
- Pro-activity : Ability to perform language oriented goal-directed behavior.
- Social–ability : Ability to exhibit interact in language sensitive environments.
- Learning : Ability to adapt to user's language preferences and new languages.

1. Objectives & Motivation

This work is aimed at developing an intelligent agent based system for information retrieval from database. Database hold huge quantities of data. There are several artificial languages for manipulating the data. But their usage needs knowledge about the database structure, language syntax etc. Several Natural language front-ends have been developed as a result of rigorous works done in this field of artificial intelligence. But majority of them use English as the natural language. India being a multi-lingual country

with only 5% of the population having education up to matriculation level, their use is limited. In this context, information retrieval in regional languages from database has tremendous impact. It is of very great use in application areas like e-governance, agriculture, rural health, education, national resource planning , disaster management , information kiosks etc. Even research organizations can make of use of such systems to bring their findings to common man.

The agent based information system developed can retrieve information in Hindi and Malayalam from the National Resource Information database. The user can give his/ her query to the system as if he/ she delegates a human being for information gathering. There is no rigid syntax for asking the query. The system using NLP techniques tries to extract meaning of the query and retrieve information from data stores and present the information to the user in his/ her own native language [2]. This system provides the following advantages to the user.

- the user can communicate with data stores in the regional languages
- flexibility in communication
- shielding the users from complexities of database model, DDL, DML etc
- handling anaphoric and elliptical queries
- capability of handling user misconceptions

2. System Design

The information retrieval system is implemented using Open Agent Architecture (OAA). The OAA is a framework in which a community of software agents running on distributed machines can work together on tasks assigned by human or non-human participants in the community [3]. The system is configured as a multi-agent system. The agents collaborating in the system are user-interface agent, morph-analyser agent, parser agent, SQL agent and facilitator agent. The communication language used among the agents are Interagent Communication Language (ICL). Fig.1 presents the basic architecture of the system.

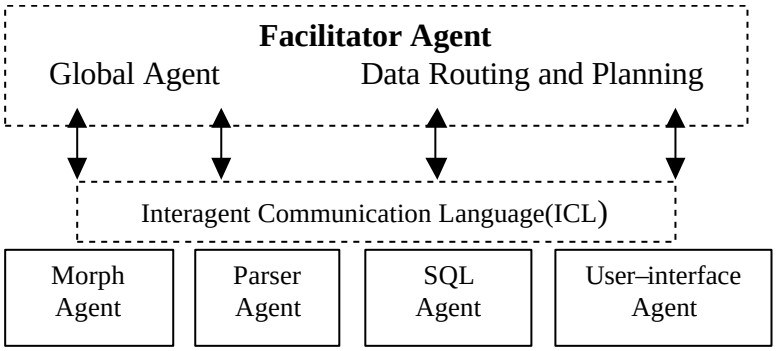


Fig 1 . System architecture

These agents are distributed over a network of machines. There is only one database server. Multiple users can use the system at the same time. For running the system all the machines should have J2DK 1.4.0 and OAA software package. All the client machines should have the User Agent loaded in them.

2.1 User Interface Agent

It accepts queries from the user. Query can be input through the key board or through the window keypad with the help of touch pen. Both Malayalam and Hindi languages have been supported.

Seeing the difficulties of input/output in Indian languages, we decided to develop some convenient approaches to tackle this problem. They include

- Keyboard
- Window Keypad & Touch pen/Mouse

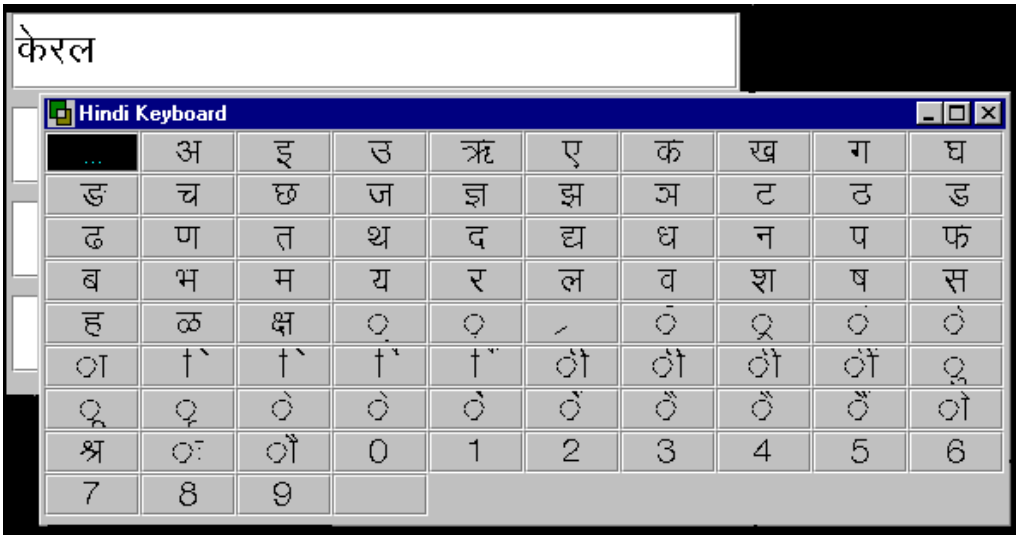
2.1.1 Keyboard Input

The commonly used method for entering text into the computer is with the help of a keyboard. Most of the computers are provided with QWERTY keyboard, which do not have enough keys to support all letters of Indian languages. So better utilization of available keys along with well-designed key assignments play an important role in the usability of keyboard in Indian context. As most of the users are well versed in using English keyboard, we decided to map Indian letters to English keys based on extent of similarity in pronunciation. For example Hindi letter 'Û ' is assigned to key 'k', letter '•Ö' to key 'j' etc. We are inputting joint letters with the help of key combination, for example the typing of the sequence › + Ë + œ will result in ù. The system will check the key sequences and will do automatic transformation. This technique will help to enter all letters in Indian languages within the current keyboard limitations.

Another problem in Indian language software creation is the rendering of characters. The standard used for information exchange is ISCII, which is an 8 bit-character encoding scheme where the Indian language characters are assigned to a unique ASCII value within the range of –127 to –1. A font that satisfies ISCII standard can render letters without any problem. But popular programming languages like Java use two byte Unicode characters. So if we want to render Indian letters in Java GUI controls, we have to map ISCII to Unicode. Now Unicode is emerging as an International standard for all languages. There is a Unicode standard for all Indian language scripts, Devanagari (U0900) ranges from 0900 to 097F and Malayalam (U0D00) ranges from 0D00 to 0D7F. As it's found as difficult to get fonts that satisfies both ISCII & Unicode we are doing a font specific mapping The Indian fonts used for Hindi and Malayalm are DVBW-TTYogeshEN and MLB-TTIndulekha respectively developed by C-DAC.

2.1.2 Entering Text Using Window Keypad & Touch pen / Mouse

The problems in key assignments and lack of sufficient keys to support Indian languages force as to provide a Keypad window with the text controls. The Keypad contains buttons for all letters for a particular language and users can activate the Keypad by right clicking on any text control. The users can enter text by selecting buttons on the Keypad with the help of Mouse or Touch pen. Even though this is not as flexible as typing from Keyboard, this will provide an additional feature mainly for the users who are not well versed with the key assignments. The end-users can either use the keyboard or the Keypad to enter text to a GUI Text component. The sample view of Hindi text control and Keypad is given below.



2.2 Morph Agent

Morphological analysis is the process of finding the root and the grammatical features of a word . This agent performs activities like splitting the query into individual words, spelling correction, domain dependent word grouping and attaching semantic properties to each word group. In this work the Morphological Analysis of words is conducted in a domain (database) specific way rather than a language specific way. For example consider the string “greater than”. In database specific sense it is a relational operator, but in language sense the words “greater” & “than” come under different grammatical categories and later we have to combine these to form a relational operator that requires one more processing step. So the main advantages of domain specific analysis are: -

- It will be easier to tokenize the incoming user query.
- It is the natural way in which human experts are performing.

Here the token represents a “meaningful unit” in database context and the morphological analysis is the process of identification of tokens present in the natural language query with the help of a lexicon. Knowledge content of Tokens are stored as frames. The output of morphological analysis will be a table that contains tokens.

Each token have following slots: -

- Tag
- Id
- Value
- Optional

Tag is a string that implies the database specific category of the token. A list of most significant tags & interpretations are given below in Table 1.

Tag	Interpretation
<CFLD>	Character attribute of a database table
<NFLD>	Numeric attribute of a database table
<CVAL>	Character value
<NVAL>	Numeric value
<FUNC>	An SQL function
<AND>	The logical operator AND
<OR>	The logical operator OR
<COND>	Relational operator

Table 1 Token Table

The tag labels are selected to imply database specific meaning. Id is an integer that along with tag will uniquely identify a token in a token table. The value is the actual content of the token and optional is a location that can be used to store additional information if necessary. The tag sequences that we can extract from the token table can be used for syntactic and semantic analysis. The Morph agent makes use of a Lexicon and Vibakthi Lexicon for its processing.

Lexicon Structures : The objective of the lexicon design is that it should contain sufficient information about words to fill the necessary slots of the token. So the lexicon should also contain almost similar attributes as that of tokens. Each record in a lexicon table contains following set of attributes: -

- ? Word
- ? Tag
- ? Meaning
- ? Optional

The word attribute is a string containing single or multiple words that form a domain specific “meaningful unit”. Tag is having similar purpose as in the case of token and meaning is the domain specific meaning of the word and optional is for additional information about word if required.

A sample filling of lexicon is given in Table 2.

Word	Tag	Meaning	Optional
•Ö®ÖÃÖÓÜµÖÖ	<NFLD>	Demography.totp	
†î¸	<AND>	AND	
ÃÖ²ÖÃÖê †×¬ÖÛú	<FUNC>	MAX	
ÃÖê ÃÖ×¬ÖÛú	<COND>	>	
Ûêú¸»Ö	<CVAL>	Kerala	State.sname

Table 2 Lexicon

2.2.1 Vibhakthi Lexicon

The lexicon structure given above is enough for Hindi morphological analysis, but in Malayalam and other Dravidian languages the Vibhakthi is always attached to the words and will play a significant role in understanding the relation between words in a string. The following example will demonstrate the difference between Hindi & Malayalam: -

For a Hindi string: Ûêú¸»Ö ´ÖêÓ

Malayalam equivalent is: ¶‰¥¨Ì°Þ

Which is a combination of: ¶‰¥¨¹ and ƒÞ

From the above example it is clear that for doing morphological analysis in Malayalam there is a requirement for additional information, which will help to extract word & vibhakthi from a combined word. The conventional way of doing this is either by storing all forms of a word or by storing the type of transformations occuring while adding each vibhakthi. But in our problem the NRI database contains thousands of words that represent proper nouns such as state names, district names, village names etc. We have to address the problem of expansion (for example addition of a new state) and modification (for example the re-naming of an existing place). So following the conventional way will lead to a situation that, once a new word is added to database we have to add that word and vibhakthies manually to lexicon. Another factor of consideration is the size & complexity of the lexicon. To address the above-mentioned problems we have designed an effective strategy by building an additional lexicon for Malayalam morphological analysis. The lexicon structure and sample records are given in Table 3.

Word1-Word2	Word2-Word1	Vibhakthi Tag
¹	µÌ¾¯à	<KAL>
¹	Ì°µú	<INTE>
¹	Ì°Þ	<IL>
¸	°µú	<INTE>

Table 3. Vibakthi Lexicon

The Word1 is the root word that is present in the lexicon and Word2 is the word that is present in the user query. The first attribute is the result of Word1 *minus* Word2 and second attribute is the result of reverse process. The vibhakthi tag is the identifier for a particular vibhakthi. The *minus* operator performs a character level comparison from left to right and if there is a miss match between letters, it will add the letter of the first word to difference. The following examples will give the clear picture: -

Word1	¶	‰	¥	¨	¹		
Word2	¶	‰	¥	¨	Ì	°	Þ
Word1 - Word2	-	-	-	-	¹	-	-
Word1	¶	‰	¥	¨	¹		
Word2	¶	‰	¥	¨	Ì	°	Þ
Word2 - Word1	-	-	-	-	Ì	°	Þ

Only the root word is stored in the lexicon and all valid transformation rules are stored in the vibakthi lexicon in the form as narrated above. So the word in the user query is compared with the words in lexicon and if it satisfies the above equations then we can find the root word (word in lexicon) and the vibhakthi attached to it. The completeness of the vibhakthi lexicon determines the accuracy of the whole process. The noted advantages of this method are: -

- ? As the same vibhakthi transformation rule can be applied to a class of words, the size & complexity of the lexicon is reduced considerably.
- ? In the event of a database record modification, the system can automatically check for new words by querying the database to get values of all attributes that are having string data type.

Automatic Spell Checking & Correction : In an effort to give more flexibility to the end-user, the system will perform a word level automatic spell checking and will correct the spelling if there is a reasonable level of matching. The spell checking is performed with the help of lexicon. The words in the incoming query are compared with the words that are present in lexicon. Note that the lexicon contains string of words in the *Word* field, but the spell checking is performing at individual word level. So there is a necessity to build a linear list of words from lexicon. The process of spell checking will proceed by performing the following steps: -

- ? Check whether the word represents numeric value.
- ? Check for an exact word match
- ? Check for Word + Vibhakthi (For Malayalam only)
- ? Check for a reasonable match
- ? Mark word as *JUNK*

The process will systematically perform each step and if all the initial four steps fail then the word will be considered as a junk word. The incoming word that is having more that 2/3 match to lexicon word(s) is considered as a candidate for spell correction and is replaced with maximum matched lexicon word. A bi-directional checking along with size comparison is required to find the extent of matching between two words.

Word Grouping : This is the final phase of Morphological Analysis. After spell checking & correction the refined words are grouped together to form database specific tokens. Token may contain a single word or group of words. As said earlier tokens are the 'meaningful units' that help in domain specific processing. Word grouping identifies all the tokens that are present in the user query. The algorithm will handle possible ambiguities that might arise during grouping. The algorithm initially check for a valid token by grouping maximum number of words and if no match is found then will proceed the checking by eliminating last word from the group. For example after word grouping the sentence "Ûêú¸»Ö Ûêú ÃÖ²ÖÃÖê †×¬ÖÛú •Ö®ÖÃÖÓÜµÖÖ" will produce following set of tokens: -

Token	Ûêú¸»Ö	Ûêú	ÃÖ²ÖÃÖê †×¬ÖÛú	•Ö®ÖÃÖÓÜµÖÖ
Tag	<CVAL>	<KE>	<FUNC>	<CFLD>

The word grouping will produce a token table that contains all necessary information about tokens, which itself is the final output of the entire Morphological Analysis task. The tag string <CVAL><KE><FUNC><CFLD> is used for doing syntactic and semantic analysis of the user query. The additional information about the tokens are used for SQL generation.

2.3 Parser Agent

The syntactic analysis is the process of checking the validity of a sentence. It checks whether the sentence is grammatically correct . Here pattern driven parsing is used. Each pattern has got specific database mapping meaning. These patterns are then mapped to columns & conditions in a database.

The parsing is with the help of a set of production rules. The production rules contain natural language pattern as antecedent and category as consequence. The incoming query is split to form a tree, which contain patterns found in the natural language query. Since our main aim is to find the conditions and columns for generating SQL, we are having only two category items. They are <CONDITION> and <COLUMN>. If any pattern is not following any of the antecedent of a production rule then it will be treated as a <FILLER>. The Table 4 gives some of the production rules used: -

Pattern (Antecedent)	**Category (Consequence)**
<NFLD><NVAL><COND>	<CONDITION>
<FUNC><NFLD>	<COLUMN>
<CVAL><CFLD>	<CONDITION>
<CFLD>	<COLUMN>
<FUNC><CFLD>	<COLUMN>
<CVAL>	<CONDITION>
<NFLD><NVAL><NVAL><COND>	<CONDITION>
<FUNC><NFLD>	<COLUMN>

Table 4 Antecedents & Consequents of production rules

After parsing the tokens that match with the antecedent of a production rule, they are grouped together to form a node. The node contains information about the pattern, category and participating tokens, which is the final output of the overall understanding process. These nodes are passed to the next module for SQL generation.

The natural language understanding is done by the collaborative work of all the above-mentioned agents. The natural language query will go through each of these agents and undergoes certain type of transformation/processing. The final output is used for SQL generation.

Example

The overall understanding process can be explained with the help of an example as shown below: -

Consider the following natural language query after spell correction: -

×ÛúŸÖ®Öê ÝÖÖÑ¾Ö ´ÖêÓ •Ö®ÖÃÖÓÜµÖÖ 2500 ÃÖê †×¬ÖÛú Æî

The word grouping will result in following set of tokens: -

[×ÛúŸÖ®Öê] [ÝÖÖÑ¾Ö] [´ÖêÓ] [•Ö®ÖÃÖÓÜµÖÖ] [2500] [ÃÖê †×¬ÖÛú] [Æî]

The tagging is done as follows: -

Token	×ÛúŸÖ®Öê	ÝÖÖÑ¾Ö	´ÖêÓ	•Ö®ÖÃÖÓÜµÖÖ	2500	ÃÖê †×¬ÖÛú	Æî
Tag	<FUNC>	<CFLD>	<MEM>	<NFLD>	<NVAL>	<COND>	<HY>

The token table produced as a result of Morphological Analysis is given below. The id is used for identifying tokens of similar category. For example if one more function is available then it will be having an id '2'.

Tag	Id	Meaning	Optional
1	<FUNC>	1	COUNT
2	<CFLD>	1	Village.vname
3	<MEM>	1	mem
4	<NFLD>	1	Demography.totp
5	<NVAL>	1	2500
6	<COND>	1	>
7	<HY>	1	hy

The parsing is done on the tag sequence: -

<FUNC><CFLD><MEM><NFLD><NVAL><COND><HY>

Parsing results in identification of following set of patterns, which is used by the SQL generator: -

Pattern	Category	Tokens
<FUNC><CFLD>	<COLUMN>	Tokens 1 & 2
<MEM>	<FILLER>	Token 3
<NFLD><NVAL><COND>	<CONDITION>	Token 4,5 & 6
<HY>	<FILLER>	Token 7

2.4 SQL Agent

This agent generates SQL equivalent of the user inputted natural language query. After the understanding task, we will get some clear indications of required columns & conditions in the final SQL. The SQL is generated based on the underlying database structure and set of expert rules for query building. So there should be provisions for: -

- Getting information about the underlying database structure, which includes meta-data of individual tables
- Building SQL by interpreting the meaning of particular natural language pattern and with the help of database specific information.

2.4.1 Understanding Database Structure

As many tables are there in the database, the type of relation between tables and the conditions for joining tables have significant effect on the final SQL. Also there should be sufficient data about the individual tables such as fields, data type, constraints etc. The table structures shown in Table 5 and Table 6 are used for SQL generation. The Table 5 is used for storing information about individual tables and Table 6 is used for storing information about the relation between tables in the database.

Database Table Field	Database Table	Data Type	Constrain
State.scode	State	CHAR	PKEY
State.sname	State	CHAR	
Demograhy.totp	Demography	NUMBER	

Table 5 Metadata of database tables

Database Table 1	Database Table 2	Table Join Condition	Relation Type
State	Village	State.scode=Village.scode	Master-Detail
Village	Transport	Village.vcode=Transport.vcode	Detail-Detail
Transport	Landuse	Transport.vcode=Landuse.vcode	Detail-Detail
State	Landuse	State.scode=Landuse.scode	Master-Detail

Table 6 Table Join Conditions

The Tables 5 & 6 will provide database specific information to the SQL generator as and when required.

2.4.2 Building SQL

Interpretation of the natural language patterns that we received as a result of parsing is required for generating SQL. The system will have to interpret the expert rules that are having a format similar to the one that is given below: -

```
IF (pattern = “Some natural language pattern”)
   {
   ____________________________________
   Actions for building SQL
   ____________________________________
   }
```

For example: -

```
IF (pattern = “<FUNC><NFLD>”)
   {
   ADD ‘<FUNC>(<NFLD>)’ TO COLUMNS;
   ADD TABLE(<NFLD>) TO TABLES;
   }
```

There are two methods for storing and interpreting the expert rules. These are: -

- Hard-code the rules in a programming language
- Create rule-interpreter for interpreting rules stored in a rule-base.

The first method is having the advantage that it is easy to implement. But if we want to add new rules then the only way we could do is by making changes to the program itself. If we are using the second method, we can add/modify rules in the rule-base without making any modification to the program. The rule-base can be created as an XML document, so that the processing can be done easily with the help of already available XML parsers.

The rule-base will look something similar to the one given below: -

```
<Rule-Base>
   <Rule>
      <Pattern> FUNC NFLD </Pattern>
      <Action>
         <Exec> ADD 'FUNC(NFLD)' TO COLUMNS</Exec>
         <Exec> ADD TABLE(NFLD) TO TABLES</Exec>
      </Action>
   </Rule>
</Rule-Base>
```

The rule-base given above is the simplified form, the original one will embed programming language codes such as function calls inside the <Exec> </Exec> tags. The rule interpreter will parse this XML document and will execute the statements in the <Exec> </Exec> tags. The great aspect of this approach is that in future we will be in a position to formulate some meta-rules for producing rule-base for a new database. The meta-rules contain instructions to build rule-bases for different database structures. So it will be possible to produce configurable natural language interface (NLI), which will reduce the time & effort required for producing interfaces separately.

2.5 Facilitator Agent

The facilitator agent is a server agent that is responsible for coordinating agent communication and control and for providing a global data store to its client agents. It maintains a registry of agent service and data declarations [4]. The facilitator agent breaks tasks into subtasks, matches subtasks to service providers, routes and collects information among distributed participants.

2.6 Interagent Communication Language (ICL)

ICL is the interface language shared by all agents, no matter what machine they are running on or what computer language they are programmed in. Every agent participating in the system defines and publishes a set of capabilities specifications expressed in the ICL, describing the services that it provides [5,6]. These establish a high-level interface to the agent, which is used by a facilitator in communicating with the agent and delegating service requests to agents. The capabilities are referred as "solvables". In this work parsing a token string is a solvable for the parser agent. Displaying the output on the screen is a solvable for the user-interface agent. While performing a task, an agent can request the service provided by other agents through the procedure "Solve( )"

3. Results

Some sample screen shots of the multilingual information retrieval system are given below. Since we have only developed a prototype of the system, we haven't populated the database considerably. Hence only few records are seen in the output. The User Agent displays the results at the client machines.

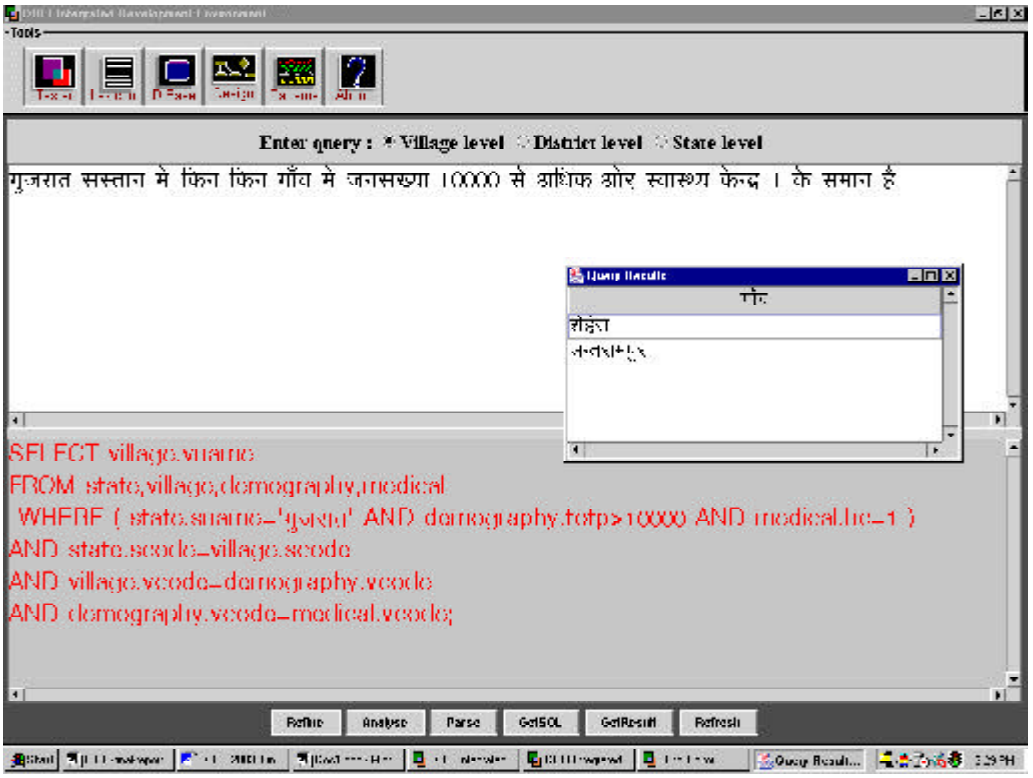
The screen shots display the results of queries given in Hindi and Malayalam. The user typed query and the corresponding SQL statement generated by the SQL Agent and the result obtained can be seen in the screen shots.

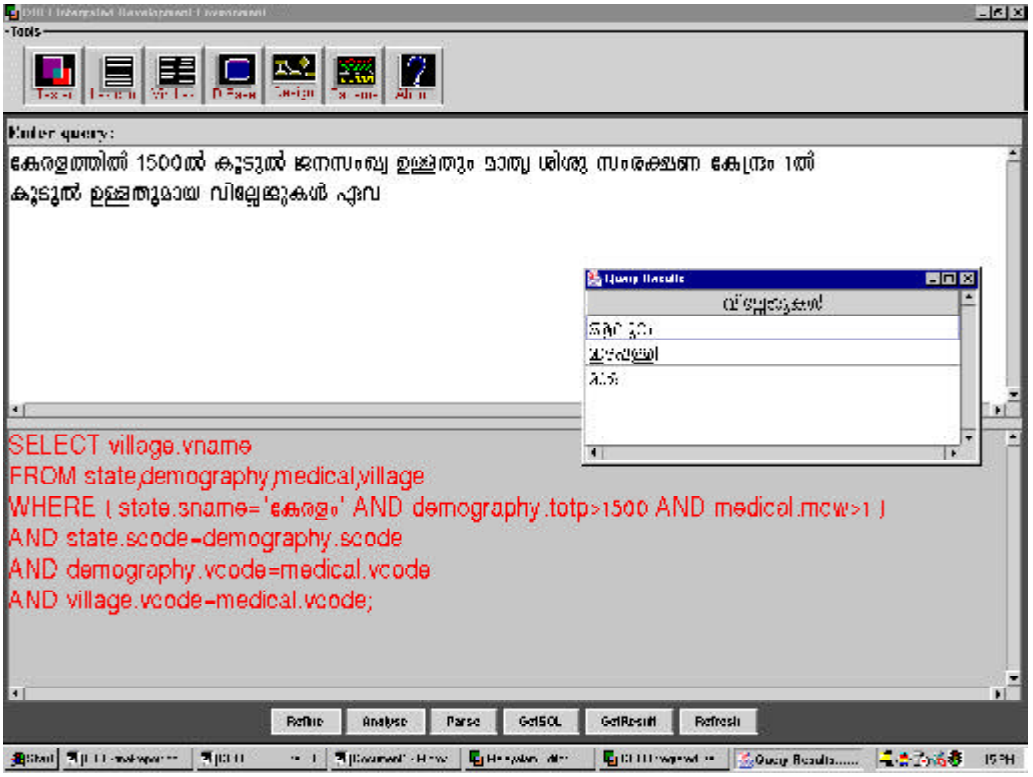
The Hindi and Malayalam queries are given below.

[illegible]

കേരളത്തിൽ 1500ൽ കൂടുൽ ന ംഖ്യ ഉള്ളതും മാതൃ ശിശു ംരക്ഷണ കേന്ദ്രം 1 ൽ

കൂടുൽ ഉള്ളതുമാമ വില്ലേ ുകൾ ഏവ





4. Conclusion & Future work

The advantage of this system is that the user can query the data store in his/ her own native language without knowing the complexity of the database structure and location of the database. The agents involved in the system are adaptive to the user language and the usage style. It is the facilitator agent who is planning and coordinating the sequence of tasks involved in query processing. The Open Agent Architecture is useful for building complex systems in which there are many heterogeneous components and in which flexibility and extensibility is important. The user interface can be made more friendly by adding agents capable of processing multimodal inputs like speech and gesture. Now the output is only in text form. It can be extended to include spatial information by integrating this system with spatial database. It adds more value to the result and can be effectively used by government bodies for resource planning. The common language framework used, makes this system adaptable to other regional languages of India .

5. Acknowledgement

This research work was sponsored by Indian Space Research Organization. We thank I.C Matieda of Space Applications Center, Ahmedabad for providing many analyses and suggestions that helped an efficient implementation.

6. References

1. Gerhard Weiss, *Multiagent Approach to Distributed Artificial Intelligence*, MIT Press 1999.
2. Bahrathi A, Chitanya V and Sangal R, *Natural Language Processing – A Paninian Perspective,* Printice-Hall of India, 1996
3. Michael N Hunhs, Munindar P Singh , *Readings in Agents*, Morgan Kaufmann 1997
4. Jeffrey M Bradshaw, Software Agents, AAAI Press, 1997
5. Nicholas R Jennings, Michael J Woodridge*, Agent Technology - Foundation, Application and Markets*, Springer , 1999
6. M. Sugumaran and P. Narayanasamy, *"An Intelligent Multiagent Meeting Schedule"r* in the proceedings of the International Conference on Artificial Intelligence in Engineering and Technology, Malaysia, 2002

About Authors

Dr. Sumam Mary Idicula is Reader in Computer Science, Cochin University of Science & Technology. She holds Ph.D in Computer Science from Cochin University of Science & Technology. She has contributed 18 Technical papers in the proceedings of International Conferences & Journals. Her area of Interest are Multilingual Computing, Intelligent Agent-based Systems, Software Engineering.
E-mail : sumam@cusat.ac.in

Sh. David Peter S is Reader in Computer Science Cochin University of Science & Technology. He holds M.Tech in Computer Science & Engineering from IIT, Chennai. He has to his credit 17 Technical papers in the proceedings of International Conferences & Journals. His fields of Interest is Intelligent Agent-based Systems, Software Engineering, Neural Networks.
E-mail : davidpeter@cusat.ac.in