Masters Thesis Partha Pakray

DATASET · JANUARY 2015

READS
33

1 AUTHOR:

Dr. Partha Pakray
National Institute of Technology, Mizoram
51 PUBLICATIONS **92** CITATIONS

SEE PROFILE

MULTILINGUAL RESTRICTED DOMAIN QA SYSTEM WITH DIALOGUE MANAGEMENT (BENGALI AND TELUGU AS A CASE STUDY)

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE AND ENGINEERING IN THE FACULTY OF ENGINEERING AND TECHNOLOGY, JADAVPUR UNIVERSITY

By

PARTHA PAKRAY

UNDER THE ESTEEMED GUIDANCE OF

Prof. SIVAJI BANDYOPADHYAY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
JADAVPUR UNIVERSITY
KOLKATA-700032
2007

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING AND TECHNOLOGY
JADAVPUR UNIVERSITY

TO WHOM IT MAY CONCERN

*I hereby recommend that the dissertation entitled "**Multilingual Restricted Domain QA System with Dialogue Management (Bengali and Telugu as a case study)**" has been carried out by **Partha Pakray**(Reg. No. 93509 of 2005-06, Class Roll No. 000510502002), under my guidance and supervision may be accepted in partial fulfillment for the degree of **Master of Computer Science and Engineering** in the Faculty of Engineering and Technology, Jadavpur University.*

…….

(Prof. Sivaji Bandyopadhyay)
Thesis Supervisor,
Department of Computer Science & Engineering,
Jadavpur University, Kolkata-700032.

Countersigned:

……
(Prof. Atal Chaudhuri)
Head of Department,
Department of Computer Science & Engineering,
Jadavpur University, Kolkata – 700032.

……
(Prof. M. K. Mitra)
Dean,
Faculty of Engineering and Technology,
Jadavpur University, Kolkata – 700032

JADAVPUR UNIVERSITY
FACULTY OF ENGINEERING AND TECHNOLOGY

CERTIFICATE OF APPROVAL

The foregoing thesis is hereby accepted as a credible study of an engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein, but approve the thesis only for the purpose for which it is submitted.

FINAL EXAMINATION FOR
EVALUATION OF THESIS

1. ——————————

2. ——————————

(Signature of Examiners)

DEDICATED

TO

MY PARENTS

Acknowledgement

I heartily wish to express my deepest gratitude to my respected teacher and guide, Prof. Sivaji Bandyopadhyay, for his active guidance and valuable suggestions through out the present work. I am very much indebted to him for the constant encouragement and inspiration that he has showed on me. I am very grateful to him for introducing me to this fruitful area of Natural Language Processing and for giving me the freedom to explore it. The above words are only a token of my deep respect towards him for all he has done to take my project to the present shape.

I am also highly grateful to Mr. Sudip Kumar Naskar for his valuable co-operation and motivation in the entire work.

I want to thank Prof. Atal Chaudhuri, Head of the Department, Computer Science & Engineering for providing me all the help as and when required by me.

I want to thank Kishore Ghosh Sir, Magnum Lab, Computer Science & Engineering Dept. for providing me all the help as and when required by me.

The Department of Computer Science & Engineering has provided good facilities for this work. There are lot of staff members who have provided the support needed to complete this work.

I am highly obliged to my friends Srinivas Rao, Susanta Bouri,Moloy Dhar who have co-operated with me both inside and outside the department in successfully completing this thesis work.

Last, but not least, I convey my deep sense of thankfulness to my family members and well wishers -- this would never have been possible with their support, both emotionally and mentally.

(PARTHA PAKRAY)

CONTENTS

Chapter 1: Introduction

Chapter 2: Literature Review

Chapter 3: Design of the Multilingual RDQA System

Chapter 4: Implementation of the Multilingual RDQA System

CHAPTER I

INTRODUCTION

1.1 About Question Answering system

A Question Answering (QA) system is an automatic system capable of answering natural language questions in a human-like manner: with a short, accurate answer.

The QA systems can be characterized with several qualities that fundamentally arise from the 1. Data content, 2. Format and 3. Language. A question answering system can be domain specific, which means that the topics of the questions are restricted. Often, this means simply that also the document collection, i.e., the corpus, in which the answer is searched, consists of texts discussing a specific field. This type of QA is easier, for the vocabulary is more predictable, and ontologies describing the domain are easier to construct. The other type of QA, open-domain question answering, deals with unrestricted topics. Hence, questions may concern any subject. The corpus may consist of unstructured or structured texts. Yet another way of classifying the field of QA deals with language. In **monolingual** QA both the questions and the corpus are in the same language. In **cross-language** QA the language of the questions (source language) is different from the language of the documents (target language). The question has to be translated in order to be able to perform the search. **Multilingual systems** deal with multiple target languages,i.e., the corpus contains documents written in different languages. In multilingual QA, translation issues are thus central as well.

Since the early days of artificial intelligence in the 60’s, researchers have been fascinated with answering natural language questions. However, the difficulty of natural language processing (NLP) had limited the scope of QA to domain-specific expert systems. QA has been studied in NLP since 1970s with the systems like BASEBALL [33], which provides answers to questions about the American Baseball League and LUNAR [11], which allowed geologists to ask questions about moon rocks. In recent years, the combination of web growth, improvements in information technology, and the explosive demand for better information access has reignited the interest in QA systems.

QA is regarded as more complex NLP application than other types of applications like information retrieval (IR) or information extraction (IE), and it is some time regarded as paramount of IR/IE. Typically QA is supported by Natural Language Processing and IE [16].

Every day people encounter situations when they need answers to questions in order to succeed in their personal or professional endeavors. For example, users may need to know quickly “When does Darjeeling Mail starts?” or “Where is Gate way of India?”, while the traditional approach has been to rely on structured knowledge sources (e.g. databases) or human expert help, a recently becoming popular approach is to provide automated question answering based on the information stored in large digital libraries of unstructured text documents or in the entire World Wide Web.

Current information retrieval systems like Google, Yahoo, and AltaVista and AOL etc. allow us to locate documents that might contain the pertinent information, but most of them leave it to the user to extract the useful information from a ranked list. This leaves the (often unwilling) user with a relatively large amount of text to consume. There is an urgent need for tools that would reduce the amount of text one might have to read in order to obtain the desired information. This track aims at doing exactly that for a special (and popular) class of information seeking behavior: **QUESTION ANSWERING**. People have questions and they need answers, not documents. Automatic question answering will definitely be a significant advance in the state-of-art information retrieval technology.

The goal of question answering (QA) is to identify and present to the user an actual answer to a question, rather than identifying documents that may be topically related to the question or may contain the answer. Complexity of natural language processing (NLP) and, as a result, the expenses associated with developing NLP based QA systems, resource consuming processing and low reliability may explain the reluctance of the developers of digital libraries or web search portals to incorporate QA technology. There could be different types of questions.

- Factual questions for which the answer is found verbatim or as a simple morphological variation. For the example question, “Who killed Mahatma Gandhi?”, the answer passage containing “Nathuram Godse killed Mahatma Gandhi”.
- Reasoning questions like “How did Socrates die?”, here ‘die’ may have to be linked with ‘drinking poisoned wine’ found in the retrieved document.
- Questions that require answer fusion as in “How can I assemble a computer?”. Here, the answer might have to be assembled from partial answers scattered across the same/various documents.

- Interactive questions are dialogue based, the answer should be found in the context of previous questions.

1.2 Types of Question Answering

The most popular classes of technique for QA are open-domain and restricted-domain. These two domains use thesauri and lexicons in classifying documents and categorizing the questions.

1.2.1 Open Domain Question Answering (ODQA)

Open domain question answering (ODQA) [35] deals with questions about nearly everything and can only rely on general ontology. ODQA has become a very active research area over the past few years, due in large measure to the stimulus of the TREC [30] Question Answering track. The TREC (Text Retrieval Conference) is a series of workshops designed to advance the state-of-the art in text retrieval by providing the infrastructure necessary for large scale evaluation of text retrieval methodologies.

Ask Jeeves ("http://www.ask.com") is the most well known ODQA system. According to the Nielsen/NetRatings MegaView Search report in 2005, Ask Jeeves is the fifth most popular search engine and the only natural language search engine on the list. Posing the question "*Who is the president of the United States*" to Ask Jeeves, elicits "*The Chief of State of the United States is President George W. Bush, who is also Head of State*" as the answer. To answer unrestricted questions, a general ontology or commonsense knowledge would be useful.

1.2.2 Restricted Domain Question Answering (RDQA)

Restricted-domain question answering (RDQA) [7] deals with questions under a specific domain like railways or medicine. The possible questions are limited by the domain, therefore it is possible to encode all the domain knowledge or ontology in the system to analyze questions or answer sources. In this RDQA, system gives exact answer instead of listing a set of documents related to answer. The answer sources can be fully structured data, which is easier to process. Green's BASEBALL [33] system is a restricted-domain QA system that only answers questions about one season's baseball data.

1.3 Question Answering Method

QA is very dependent on a good search corpus - for without documents containing the answer, there is little any QA system can do. It thus makes sense that larger collection sizes generally lend well to better QA performance, unless the question domain is orthogonal to the collection.

Parsing means taking an input and producing some sort of structure for it. The kind of structures that might be produced are morphological, syntactic, semantic and pragmatic

1.3.1 Shallow Parsing

Some methods of QA use keyword-based techniques to locate interesting passages and sentences from the retrieved documents and then filter based on the presence of the desired answer type within that candidate text. Ranking is then done based on syntactic features such as word order or location and similarity to query.

When using massive collections with good data redundancy, some systems use templates to find the final answer in the hope that the answer is just a reformulation of the question. If you posed the question "What is a dog?", the system would detect the substring "What is a X" and look for documents which start with "X is a Y". This often works well on simple "factoid" questions seeking factual tidbits of information such as names, dates, locations, and quantities

Shallow or Partial parsing is a task of recovering only a limited amount of semantic or syntactic information from the input natural language (NL) questions. It has proved to be a useful technology for the written and spoken dialogue domains. One of the main application area of shallow parsing is QA. It provides the structural basis for the NL questions. We are using shallow parser as a predefined module to analyze our input NL questions.

Typical modules within a shallow parser architecture include the following:

1. Part-of-Speech Tagging: Given a word and its context, to decide what the correct morphosyntactic class of that word is (noun, verb, etc.).POS tagging is a well-understood problem in NLP to which machine learning approaches are routinely applied.

2. Chunking: Given the words and their morphosyntactic class, to decide which words can be grouped as chunks (noun phrases, verb phrases, complete clauses, etc.).
3. Relation Finding: Given the chunks in a sentence, to decide which relations they have with the main verb (subject, object, location, etc.)

Application Domain of Shallow Parsing:

1. Speech-to-Speech translation system
    - Used to add robustness (Verb Mobil project [32])
2. Question Answering
    - Used to efficiently process large ill-formed documents
    - Parsed efficiently for accessing the information from NL input.
3. Text-mining Application
    - Used to biology text mining
    - Used to reduce the search space for full-blown ‘deep’ parsers

1.3.2 Deep Parsing

In full or deep parsing, a grammar and a search strategy are used to design a complete syntactic structure to sentences. The main problem here is to select the most plausible syntactic analysis given the often thousands of possible analyses a typical parser with a sophisticated grammar may return. Stochastic approaches can be used to order the analyses according to their probability or to generate the most probable parse(s) only.
However, not all NLP applications require a complete syntactic analysis. A full parse often provides more information than needed and sometimes less. E.g., in Information Retrieval, it may be enough to find simple NPs (Noun Phrases) and VPs (Verb Phrases). In Information Extraction, Summary Generation And Question Answering, we are interested especially in information about specific syntactico-semantic relations such as agent, object, location, time, etc (basically, who did what to whom, when, where and why) rather than elaborate configurational syntactic analysis

1.4 Language Paradigm

Since NL questions and answers are represented by language, we can characterize QA systems by the source language, which represents questions, and the target language, which represents answers.

1.4.1 Monolingual QA system

In Monolingual QA systems, both questions and answers are in the same language. Monolingual QA is good for people who speak one of the popular languages, and researchers have paid a great deal of attention to monolingual QA research. Monolingual system is built to rely on as few resources as possible.

1.4.2 Cross lingual /Translingual QA system

In Cross lingual QA (CLQA), the question is posed in a source language and the answer must be found in a target collection of a different language. Most attempts at CLQA have so far concentrated on translating the query into English and performing monolingual English QA on the translated query.

1.4.3 Multilingual QA System

In Multilingual QA (MLQA) system, user asks questions in one language and gets answers different from the source language or same as source language. To design such a system along with the technical design, there is a need to concentrate on the linguistic perspective.

Multilingual QA has emerged only in the last few years as a complementary research task, representing a promising direction for at least two reasons. First, it allows users to interact with machines in their native languages, contributing to easier, faster, and more equal information access. Second, cross lingual capabilities enable QA systems to access information stored only in language-specific text collections.

1.5 Overview of the work done

In this work, we have developed a keyword based multilingual restricted domain question answering system with dialogue management for **Railway information** in Bengali and Telugu. The system accepts typed text inputs and provides text output as well.

The architecture of a multilingual restricted domain question answering system with dialogue management has been proposed. The system maintains language specific domain and linguistic models. The user specifies the query language and a shallow parser for the language analyzes the input text query to obtain keyword and information chunks for identifying the query topic and subsequent generation of the SQL statements. The dialogue manager detects the presence of incomplete information in the input and predicts user intentions to make recommendations based on user model and domain ontology to get user response for completing the query. The dialogue model is a finite state machine in which each dialogue state is defined with the parameters: clarification requests, intended dialogue acts and discourse goals. The system maintains dialogue history for each query topic and for each language containing values of information chunks and the answers for the previous queries in the dialogue. The answer generation process retrieves data from the database and fills slots in the answer templates specific to query topic and language. Null answers are explained and recommendations are made suggesting alternative answers. The system has been developed in the Indian Railways information domain for accepting Telugu and Bengali text inputs and has been evaluated.

1.6 Introduction to later Chapters

Related areas like Multilingual Information Retrieval (MLIR), Multilingual Information Extraction (MLIE) and Multilingual Question Ansering System have been surveyed in chapter II. Chapter II also details about Multilingual QA dialogue systems, shallow parsing usage and related systems to our present work. Chapter III details about the design of our Multilingual RDQA system for railways in Bengali and Telugu. Chapter IV gives the implementation details of our system. The evaluation has been carried out in chapter V. Conclusion and future work roadmap has also been drawn in chapter V.

CHAPTER II

LITERATURE REVEIW

2.1 Related Areas

The areas that are related to our present work are Multilingual Information Retrieval (MLIR), Multilingual Information Extraction (MLIE) and Multilingual Question Answering System. These are detailed in next sections.

2.1.1 Multilingual Information Retrieval (MLIR)

The process of information retrieval (IR) consisting of locating relevant documents on the user input, such as keywords or example documents. A user of such system may want to retrieve a particular document or a particular class of documents. The intended documents are typically described by a set of keywords. IR systems typically allow query expressions formed using keywords and the logical connectives *and, or,* and *not*. Sophisticated IR systems estimate relevance of documents to a query so that the documents can be shown in order of estimated relevance.

In current information retrieval systems, the general method is to specify some keywords to obtain necessary information on the Internet. Given a query, an Information Retrieval (IR) system returns a list of potentially relevant documents which the user must then scan to search for pertinent information. This method could not satisfy the user's needs to extract the adequate information efficiently from a huge set of electronic documents, even though the construction of the retrieval service is easy. Question Answering (QA) is a technology that aims at retrieving the answer to a question written in natural language in large collections of documents. QA systems are presented with natural language questions and the expected output is either the exact answer identified in a text or small text fragments containing the answer.

The comparison between the conventional information retrieval technology and Question-Answering technology is shown in Table 2.1.1. In IR, the input query is expressed in the engine's query language, and the output consists of a ranked list of documents that are presumably relevant to the user's query. The user is then responsible for reading the documents to learn whatever it is that he or she wants to know. QA is different from IR in that the user is allowed to ask his or her question directly to the system in natural language, without having to translate it into some query syntax. The system then answers the question in the form of an exact answer extracted from a source document.

However different the two tasks are, the fate of QA is tied to IR. In QA systems that answer questions over a corpus of documents, some provision for a coarse, first-pass search over the entire set of available documents is necessary and for that many QA systems turn to an IR engine.

	The Conventional Information Retrival	**Question Answering Technology**
Input	Some Keywords	A Question by Natural Language
Output	Document List	The Words and Phrases including the answer

Table2.1.1: The comparison between the conventional information retrieval and Question-Answering

The most traditional approach to IR in general and to multilingual retrieval in particular, uses a controlled vocabulary for indexing and retrieval. In this approach, a document list (or a computer program) selects for each document a few descriptors taken from a closed list of authorized terms. Semantic relations (synonyms, related terms, narrower terms, broader terms) can be used to help choose the right descriptors, and solve the sense problems of synonyms and homographs. The list of authorized terms and semantic relations between them are contained in a thesaurus.

The Second approach multilingual interrogation is to use existing machine translation (MT) systems to automatically translate the queries, or even the entire textual database from one language to another. When only queries are translated from a source to target language, text can be searched in the target language and results can be dynamically translated back to the source language as they are displayed after the search.

This kind of method would be satisfactory if current MT systems did not make errors. A certain amount of syntactic error can be accepted without perturbing results of information retrieval systems, but MT errors in translating concepts can prevent relevant documents, indexed on the missing concepts, from being found.

The third approach to multilingual information retrieval is based on the Salton's vector space model This model represents documents in a **n**-dimensional space (**n** being the number of different words in the text database). If some documents

are translated into a second language, these documents can be observed both in the subspace related to the first language and the subspace related to the second one. Using a query expressed in the second language, the most relevant documents in the translated subset are extracted (usually using a cosine measure of proximity). These relevant documents are in turn used to extract close untranslated documents in the subspace of the first language

The web provides a convenient way to get to, and to interact with, information sources across the internet. However, persistent problem facing the web is the explosion of stored information, with little guidance to help the user to locate what is interesting. IR has played a critical role in making the web a productive and useful tool, especially for researchers. IR is a growing field that encompasses a wide range of topics related to storage and retrieval of all manner of media.

Traditional examples of IR systems are web portals like Google, Yahoo, AltaVista etc, online library catalogs and online document management systems such as those that store newspaper articles. The data in such systems are organized as a collection of documents; in IR, a document refers generically to the unit of text indexed in the system and available for retrieval. Depending on the application, a document can refer to anything from intuitive notions like newspaper articles, or encyclopedia entries, to smaller units such as paragraphs and sentences. In web based application, it can refer to a web page, a part of a page, or an entire website. A collection refers to a set of documents being used to satisfy user requests. A term refers to a lexical item that occurs in a collection, but it may also include phrases. Finally, a query represents a user's information need expressed as a set of items.

2.1.2 Multilingual Information Extraction (MLIE)

Information Extraction (IE) is a type of document processing which analyze unrestricted text in order to extract specific types of information. IE systems do not attempt to understand all of the text in all input documents, but they do analyze those portions of each document that contain relevant information. Relevance is determined by pre-defined domain guidelines which must specify, as accurately as possible, exactly what types of information the system is expected to find.

IR system identifies a subset of documents in a large text databases or in a library scenario, a subset of resources in a library, whereas IE system identifies a subset of information within document. This subset of information is not necessarily a

summary or gist of the contents of the document. Rather it corresponds to predefined generic types of information of interest and represents specific instances found in the text. Named Entity (NE) is the low-level information extraction, which is used in many NLP applications.

An information extraction system (cf. [1]) automatically analyzes texts as directed by the user's requirements. In particular, an IE system locates and extracts relevant information from texts in foreign languages, and structures the extracted information in user-defined templates in order to feed such downstream applications as visualization tools, database queries, and automated trend analysis tools. For example, an IE system extracts information about specific events and their associated properties such as organizations, people, time, locations, etc. In addition, an IE system can infer and add additional information. For most downstream applications, the users verify the output of an IE system. A multilingual information extraction system extracts from foreign language texts. While MIE systems access language-specific knowledge sources in processing foreign language texts, some of the MIE systems represent the extracted information internally in a language-neutral way using an interlingua. The MURASAKI system is an example of such a system. Other MIE systems can only output extracted information in the given language of the text.

2.1.3 Multilingual Question Answering System

Despite the great deal of attention that Question Answering (QA) has received in recent years due to the landmark Q&A Track at the Text REtrieval Conference (TREC), multilinguality has been outside the mainstream of QA research, which has focused mainly on the English language. Multilingual QA has emerged only in the last few years as a complementary research task, representing a promising direction for at least two reasons. First, it allows users to interact with machines in their native languages, contributing to easier, faster, and more equal information access. Second, cross-lingual capabilities enable QA systems to access information stored only in language-specific text collections.

Question Answering (QA) is a type of information retrieval/extraction. For a given collection of documents (such as World Wide Web)/ structured database the system should be able to retrieve answers to questions posed in natural language.

Ever since Question Answering (QA) emerged as an active research field, the community has slowly diversified question types, increased question complexity, and refined evaluation metrics, as reflected by the TREC (Text Retrieval Conference) QA track [Voorhees, 2003]. Several QA systems have responded to these changes in the nature of the QA task by incorporating various knowledge resources [Hovy et al., 2002], handling of additional types of questions tapping into external data sources such as web, encyclopedia, and databases in order to find the answer candidates, which may then be located in the specific corpus being searched [Xu et al., 2003].

The most popular classes of technique for QA are open-domain and restricted-domain [Diekema et al., 2004]. These two domains use thesauri and lexicons in classifying documents and categorizing the questions. Open domain question answering deals with questions about nearly everything and can only rely on general ontology. It has become a very active research area over the past few years. On the other hand, Restricted-domain question answering (RDQA) deals with questions under a specific domain. The possible questions are limited by the domain, therefore all the domain knowledge or ontology in the system are encoded to analyze questions or answer sources and the system attempts to provide exact answer instead of listing a set of related documents. If we create such a RDQA interface for structured database, we call it as Natural language interface to database system (NLIDB) [Androutsopoulos et al., 1995]. When an NLIDB system is extended in a cross-language scenario, it is referred to as a Cross Language Database Systems (CLDB) [Bandyopadhyay, 2000].

In practice, current QAs can only understand limited subsets of natural language. There are kinds of questions (e.g. questions involving negation, or quantification) that can be easily expressed in natural language, but that seem difficult (or at least tedious) to express using graphical or form based interfaces. Anaphoric and elliptical expressions are also handled by the QA systems. In recent years a large part of the research in QAs has been devoted to the design of QAs that can be used in different knowledge domains (Knowledge domain portability), with different underlying Database Management System (DBMS portability), or even with different natural languages (Natural language portability). There is a growing body of research on integrating speech recognition with the goal being to implement systems that engage users in spoken dialogue to help them perform certain tasks. We expect that this line of research will have a significant influence on future QA systems, giving rise to systems that will allow users to access databases by spoken dialogue, in situations for

which graphic and form-based interfaces are difficult to use. A present direction is to use dialogue instead of isolated questions in QA systems, thus bringing the areas of information extraction (IE) and dialogue system research together [Hirshman et al., 2001]. RDQA systems with dialogue management are typically designed to provide non-misleading and useful answers to a query by having negotiations. Due to the fact that the scope of the application defined by the back-end is restricted [Matthias et al., 2005], in presence of vague, ill-defined or misrecognized input from the user, dialogue management, can interactively request more information from the user until the user's intent has been determined. During the last five years or so, research in crosslingual / multilingual question answering systems has been vigorously pursued through the Cross-Language Evaluation Forum (CLEF), NTCIR Asian Language Retrieval, Multilingual Question Answering Workshops at EACL 2006, SIGIR 2006 Workshop on New Directions in Multilingual Information Access and several other workshops on Restricted Domain Question Answering, Multilingual Summarization and Cross Language Named Entity Extraction organized by Association for Computational Linguistics. A great number of restricted domain systems have been developed: Jupiter [Zue, 1997] that provides weather information; rail travel and ticket reservation services like RailTel [Bennacef, 1996] and ARISE European project [Lamel, 2000]. A practical question answering system in restricted domain [Hoojung et al., 2004] and the present system handles user questions similarly in extracting the keywords and information chunks from the input query, identification of the query topic and subsequent generation of the SQL statement to retrieve the answer from the database. However, the present system extracts the information from a relational database and keeps track of user dialogue and handles clarifications, elaborations and confirmations needed from the user with respect to the query. A similar monolingual system is described in [Reddy and Bandyopadhyay, 2006].

The present prototype multilingual question answering system works on the Indian railways information domain that accepts typed text inputs in Telugu and Bengali and provides text output as well. It provides railway information service such as train arrival and departure time information, fare calculations, trains between important stations etc. The user specifies the query language. The system generates SQL statement(s) from the input natural language (NL) question, executes the SQL query over a relational database and then provides user-friendly natural language answer in the query language. Depending on the user needs, the dialogue manager

generates dialogues with user for clarification requests or asking the missed information in case of incomplete data or elliptical queries.

Most existing Question Answering systems classify new questions according to static ontologies. These ontologies incorporate human knowledge about the expected answer (e.g. date, location, person), answer type granularity (e.g. date, year, century), and very often semantic information about the question type (e.g. birth date, discovery date, death date).

Open domain question answering deals with questions about nearly everything and can only rely on general ontologies. It has become a very active research area over the past few years, due in large measure to the stimulus of the TREC (Text REtrieval conference), and Question Answering track. This track addresses the task of finding answers to natural language (NL) questions (e.g. *How tall is the Eiffel Tower? Who is A.R.Rahaman?*) from large collections of database. This task stands in contrast to the more conventional IR task of retrieving documents relevant to a query, where the query may be simply a collection of keywords (e.g. *Eiffel Tower, Indian musician, born Chennai, India....*).

The difficulty of constructing open domain knowledge base is one reason for the difficulties of open domain question answering. Question answering requires much linguistic and common knowledge for answering correctly. Open domain QA is a hard task because no restriction is imposed either on the question type or on the user's vocabulary. The simplest approach to improve the accuracy of a question answering system might be restricting the domain it covers. By restricting the question domain, the size of knowledge base to build becomes smaller.

Restricted domain question answering deals with questions under a specific domain (for example, Railways, Employee database, medicine). But there are many document sets in restricted domains that are potentially valuable as a source for question answering systems. For example, the documentation pages of UNIX and Linux systems would make an ideal corpus for QA systems targeted at users that want to know how to use these operating systems. Users interested in these specific areas would benefit from QA systems targeted to their areas of interest.

Restricted domains typically have limited data available and therefore conventional techniques based on data redundancy can simply not be applied in an effective way. The scarcity of data available seems to prompt for a more targeted, NLP-intensive approach to QA. The use of additional corpora such as the WWW

raises a number of interesting questions. For instance, will these corpora help or obstruct the proper functioning of an NLP-intensive approach to QA? And, how do we find good pockets of information that are appropriate to the chosen domains?

On the other hand, restricted domains (e.g. Railways, law, and medicine) have specific stylistic conventions. Often these domains use terminology that is not stored in conventional lexica. Consequently NLP approaches devised for open domain systems may under-perform on these specific domains, thus raising the question of how portable these systems can be.

Characteristics of Restricted domain system:

- Usually, for RDQA, the answers are searched in relatively small domain specific collections, so methods based on exploiting the redundancy of answers in several documents are not useful. Furthermore, a highly accurate passage retrieval module is required, because frequently the answer occurs in a very small set of passages.
- RDQA systems are frequently task-based. So, the repertory of question patterns is limited by allowing a good accuracy in question processing with limited effort.
- User requirements regarding the quality of the answer tend to be higher in RDQA. As pointed out in [14], no answer is preferred to a wrong answer.
- In RDQA, not only Named Entities (NEs), but also domain specific terminology plays a central role. This fact usually implies that domain specific lexicons and gazetteers have to be used.

A natural language interface to database (NLIDB) [3] is a system that allows the user to access information stored in a database by typing request in some natural language (e.g. Bengali, Telugu).

The very first attempts at NLP database interfaces are just as old as any other NLP research. In fact database NLP may be one of the most important successes in NLP since it began. Asking questions to databases in natural language is a very convenient and easy method of data access, especially for casual users who do not understand complicated database query languages such as SQL. The success in this area is partly because of the real-world benefits that can come from database NLP systems, and partly because NLP works very well in a single-database domain.

Databases usually provide small enough domains such that ambiguity problems in natural language can be resolved successfully.

Prototype NLIDBs had already appeared in the late sixties and early seventies. The best-known NLIDB of that period is LUNAR [11], a natural language interface to a database containing chemical analyses of moon rocks. LUNAR and other early natural language interfaces were built having a particular database in mind and thus could not be easily modified to be used with different databases. (Although the internal representation methods used in LUNAR were argued to facilitate independence between the database and other modules).

Advantages of NLIDB

- User is not required to learn an artificial communication language like formal query languages (like SQL) to access information from interface.
- There is no hassle in invoking forms, linking frames, selecting restrictions from menus, etc (like Graphical interfaces and Form-based interfaces) to access information.
- There are kinds of questions (e.g. questions involving negation, or quantification) that can be easily expressed in natural language, but that seem difficult (or at least tedious) to express using graphical or form-based interfaces.
- Support anaphoric and elliptical expressions.

Disadvantages of NLIDB

- Users find difficult to understand (and remember) what kinds of questions the NLIDB can or cannot cope with.
- User is in confusion whether the rejected question is outside the systems linguistic coverage or whether it is outside the system's conceptual coverage.
- Users assume NLIDBs are intelligent. But such systems are basically not intelligent [13].
- Inappropriate medium for communicating with a computer system. Because natural language is claimed to be too verbose or too ambiguous for human-computer interaction.
- NLIDBs usually require tedious and lengthy configuration phases before they can be used.

The **different architectures** adopted for NLIDB systems are pattern matching based, syntax based, semantic grammar based and intermediate representation language based.

- Pattern matching systems answer the user NL questions based on the matched patterns from the database. These are often managed to come up with some reasonable answer, even if the input is out of the range of sentences the patterns were designed to handle.
- In syntax based systems, the user question is parsed (i.e. analyzed syntactically), and the resulting parse tree is directly mapped to an expression in some database query language.
- In semantic grammar systems, the question answering is still done by parsing the input and mapping the parse tree to a database query. The difference, in this case, is that the grammar's categories (i.e. the nonleaf nodes that will appear in the parse tree) do not necessarily correspond to syntactic concepts.
- Intermediate representation language systems first transform the natural language question into an intermediate logical query, which expresses the meaning of the user question in terms of high level world concepts that are independent of the database structure. The logical query is then translated to an expression in the databases query language to evaluate against the database.

It is true that, intrinsically, IR engines and QA systems differ in design, objectives and process. An IR engine is geared to deliver snippets or documents from a query, a QA system strive to deliver the exact answer to a question.

If one is to differentiate three features of both systems, one of the first difference concerns the query mode: natural language for the QA systems and "Boolean like" for the IR engines. We define "Boolean like" extensively as the use of Boolean operators associated to underlying constraints induced by the word matching techniques.

The second difference concerns what is delivered to the user. Question Answering systems deliver one or more exact answers to a question and their context whereas classical engines return snippets with links to the texts those snippets were extracted from.

The third difference relates to the dynamic and openness status of the corpora. Usually QA systems use confined or close corpora with low update rate, while classical IR engines are tuned to the Web queries and their reference file are continuously updated.

We believe that QA is an ideal test bed for demonstrating the power of IE. There is a natural co-operation between IE and IR [25]; we regard QA as one major intelligence which IE can offer IR.

(i) IE can provide solid support for QA.

(ii) Low-level IE like Named Entity tagging is often a necessary component in handling most types of questions.

(iii) High-level domain independent IE, i.e. extraction of multiple relationships and general events is expected to bring about a breakthrough in QA.

2.2 QA Dialogue Systems

Language itself has always been the mark of humanity and sentience, and *conversation* or *dialogue* is the most fundamental and specially privileged arena of language. Dialogues exhibit anaphora and discourse structure and coherence, although with some slight changes from monologue.

2.2.1 Properties of Dialogue

The properties of dialogue are *turn taking, grounding* and *implicature*. One difference between monologue and dialogue is that, dialogue is characterized by *turn taking*, user asking something, system responding to the user question, then user and so on. That means a dialogue is broken up into turns. Unlike in monologue, most important character of dialogue is *grounding*, where user and system need to establish a common ground on their interaction. The need to achieve ground means that the system must ground or acknowledge the user's questions, or else make it clear that there was a problem on reaching ground.

The final property of dialogue is the way of interpreting a typed sentence, which relies more than the literal meaning of the sentences. Here *implicature* means a particular class of licensed inferences. Grice [12] proposed that what enables the users to draw these inferences is that conversation is guided by a set of *maxims*, general

heuristics which play a guiding role in the interpretation of dialogues. The proposed four maxims are: Maxim of Quantity (Be exactly as informative as is required), Maxim of Quality (Try to make our contribution one that is true), Maxim of Relevance (Be relevant), Maxim of Manner (Be perspicuous).

2.2.2 Dialogue System

A *dialogue system* has minimal knowledge of the world, the objects in the world, and the task. It observes actions and requests performed by the user. Furthermore, the dialogue system knows the abstract state of the task (whether the task is in an initial, valid, finished, or error state) and the number of task constraints that are not satisfied.

A dialogue system must be able to access, gather and integrate knowledge from various domain knowledge sources and application systems in order to determine the precise meaning of a request and produce an appropriate response. In general, dialogue systems often have a modular architecture with processing modules for interpretation, dialogue management, background system access, and generation.

Today much research is carried out within the area of natural language interfaces and dialogue systems, and more and more systems are becoming publicly available. Most of the systems provide information retrieval services or assistance in solving a specific task. The systems differ in complexity due to the domain and the approach taken in the design. Some systems are very knowledgeable and contain several interacting knowledge sources and models, others relay on much simpler models and procedures. The variety of dialogue system architectures that incorporate various models, has led to confusion when it comes to the purpose and contribution of specific model. The relations between various knowledge sources and models also diffuse.

Dialogue systems are developed for various reasons, e.g. to be used commercially or for research purposes. They can be classified along a number of dimensions such as grammar-based, plan-based, general purpose, domain specific, multi- modal, information retrieval and task planning.

Many *simple dialogue systems* are constructed in a more or less holistic fashion, not making clear differentiations between linguistic, dialogue, and domain components or reasoning, treating everything other than speech input and output as the “dialogue component”. Such architectures allow shortcuts in the design process

and fine-tuning to the particular anticipated task and dialogue interaction, which can speed up both system implementation time and run-time. However, the resulting systems are not particularly portable to other domains, tasks within the same domain, or even very robust in the face of different styles of interaction in accomplishing the task. Often, where the dialogue component is concerned, all that can be carried over into the next system is the experience gained by building such a system. Toolkits for constructing scripted dialogues, such as [27] make the construction process faster, but do not address the underlying problem of partitioning dialogue knowledge from linguistic and domain knowledge in order to reuse the same dialogue strategies.

2.2.3 Dialogue System Architectures

Dialogue systems often have a modular architecture with processing modules for interpretation, dialogue management, background system access, and generation. The processing modules utilize a number of knowledge sources, such as, grammar, lexicon, dialogue model, domain model, and task model [9].

Interpretation module used to interpret the user requests correctly and generator module used to generate user understandable response by accessing the information from the background system.

2.2.3.1 Dialogue Management

The role of the Dialogue Manager (DM) differs slightly between different dialogue system architectures, but its primary responsibility is to control the flow of the dialogue by deciding how the system should respond to a user request. This is done by inspecting and contextually specifying the information structure produced by an interpretation module. If some information is missing or a request is ambiguous, clarification questions are specified by the Dialogue Manager and posed to the user. Should a request be fully specified and unambiguous the background system can be accessed and an answer be produced. As a basis for these tasks the Dialogue Manager can utilize a dialogue model, a task model, and a dialogue history.

A *Dialogue model* used to hold the generic description of how the dialogue is to be constructed, i.e. to decide what action to take in a certain situation. It is used to control the interaction, which involves determining:

1) What the system should do next (and what module is responsible for carrying out the task)

2) Deciding what communicative action is appropriate at a given dialogue state.

There are various proposals on dialogue models that can be divided in two groups: intention-based and structurally based. They differ in how they model the dialogue, especially if the user's goals and intentions behind the requests need to be captured or not. Structurally based models are often controlled using a dialogue grammar whereas intention-based utilize plan operators. Furthermore, plan-based systems use plan operators to model not only dialogue knowledge but also task, domain and meta knowledge. This allows for plan recognition to be the only processing mechanism needed.

A *System Task model* is used to represent, how the system's tasks are performed. However, the terms, task and task model can refer to very different phenomena. It is important to make a clear distinction between the system's task(s) and the user's task(s). A user task is non-linguistic and takes place in the real world. Models of such tasks involve the user's goals and how they can be achieved. Models of system tasks describe how the system's communicative and other tasks, e.g. database access, are carried out.

A typical example of the difference between the two types of task models can be found in a time-table system where the user states that (s)he needs to be at the train station to catch a certain train and requests information on buses going there. The information that the user is going to the train station is user task model information, indicating that buses arriving after the departure time of the train are not relevant. The system task model on the other hand, models the information required for complex requests, such as date and departure place in a timetable system. It is used by the Dialogue Manager when collecting user information in order to perform a background system access. In plan-based systems the domain models takes a similar role, but wider as they often also involves advanced problem solving. In this work, we are not considering user task models, only system task models, as a part of dialogue manger and not a separated one.

The *Dialogue history* used to record the focus of attention and contains information about objects, properties, and relations as well as other dialogue information such as speech act information and system task information.

2.2.3.2 Domain Knowledge Management

If a request is fully specified it can be used to retrieve the desired information from a background system. This task is seldom discussed in literature on dialogue systems, perhaps because it is considered a rather straightforward task. There are, however, several problems related to this. For example, in cases where the background system is distributed and consists of several domains and application system knowledge sources the dialogue system must know which of them to access, in what order, and how the results should be integrated into one answer. This type of knowledge can be represented in a domain task model.

Although fully specified, requests can contain vague or ambiguous information or even some errors that cannot be detected and handled without extensive domain knowledge. This type of domain knowledge is stored in domain knowledge sources, called knowledge modules. They contain knowledge of the world that is talked about and can vary much in the form and content. Information from a domain knowledge source is primarily used to find the relevant items and relations that are discussed, to supply default values, etc. The knowledge represented in a domain knowledge source is often coupled to the application system, e.g. a database system. In such cases it is often used to map information from a Dialogue Manager to concepts suitable for database search. It is for example common that users give vague temporal descriptions that has to be mapped to more precise time intervals before the information can be used to access an application system.

To develop a Dialogue Manager that easily can be customized to new domains and in which different dialogue strategies can be explored, the Dialogue Manager should only be concerned with phenomena related to the dialogue with the user. It should not be involved in the process of accessing the background system or performing domain reasoning. These tasks should instead be carried out by a separate module, a Domain Knowledge Manager.

This Domain Knowledge Manager is responsible for retrieving and coordinating knowledge from the different domain knowledge sources and application systems that constitutes the background system. The Dialogue Manager can deliver a request to the Domain Knowledge Manager and in return expects an answer retrieved from the background system. If a request is under-specified or contains inconsistencies from the Domain Knowledge Manager's point of view, a specification

of what clarifying information is needed will instead be returned to the Dialogue Manager.

2.2.4 Example Dialogue Systems

Knowledge and reasoning that is used in dialogue systems like GALAXY, LINLIN, RAIL TEL, SUNDIAL, TRAINS, and VERBMOBIL have been described.

Monolingual Dialogue system:

The GALAXY [26] system is a distributed multi-modal multi-domain simple-service system that provides users with information about travel and weather. The LINLIN [21] system is also a dialogue system used for information retrieval, which has been customized for various domains, e.g. information about second-hand cars, charter trips to the Greek archipelago, and more recently timetable information for bus traffic. The RAILTEL [4] system and the SUNDIAL [5] system, both give information over the telephone about a specific domain, railway transportation and air traffic information respectively.

The TRAINS [18] system is a spoken-language conversational planning agent, whose task is to assist the user in managing a railway transportation system in a micro world. The TRAINS dialogue manager maintains the flow of conversation and addresses the conversational goals such as coming up with an operational plan for achieving the domain goal of successfully moving oranges. To do this, the dialogue manager must model the state of the dialogue, its own intensions, and the user's requests, goals and beliefs. The manager uses a conversation act interpreter to semantically analyze the user's utterances, a domain planner and executer to solve the actual transportation domain problems, and a generator to generate sentences to the user.

Obligations are used in the TRAINS system to enable the system to correctly produce responses. The dialogue manager reasons about the user goals. The dialogue manger's goal is to find the user's goal and creating an appropriate plan.

Multilingual Dialogue system:

ARISE (Automatic Railway Information System for Europe) [20] is a spoken dialogue system to provide train timetable information over the phone. Prototypes have been developed in four languages: Dutch, French, English, and Italian. ARISE

uses a mixed initiative Dialogue Manager (DM). A mix of implicit and explicit confirmation is used, based on how confident the system is in deciding whether an item has been correctly understood.

ARISE was aimed at improving and adjusting speech recognition, speech understanding and user-friendly dialogue strategies, with the purpose to improve the technology up to the point that it can be commercially deployed .The main goal was to develop an automatic system that handles bulk of routine, telephone based enquiries regarding train schedule. The system will work as part of expanded, human operated service so that there is seamless operator fallback. The users comprised two groups: railways in the Netherlands, France and Italy who are installing and managing the ARISE systems, and a mass market of callers who will obtain schedule information from them. The partners of ARISE have built prototypes in four different languages (English, French, Italian and Dutch).

Verbmobil [32] is a speaker-independent and bidirectional speech-to-speech translation system for spontaneous dialogues in mobile situations. It recognizes spoken input, analyses and translates it, and finally utters the translation. The multilingual system handles dialogues in three business-oriented domains, with context-sensitive translation between three languages (German, English, and Japanese). Since, Verbmobil emphasizes the robust processing of spontaneous dialogues, it poses difficult challenges to human language technology. Verbmobil is a hybrid system incorporating both deep and shallow processing schemes.

Verbmobil uses a multi-blackboard architecture that is based on packed representations at all processing stages. These packed representations together with formalisms for under specification capture the non-determinism in each processing phase, so that the remaining uncertainties can be reduced by linguistic, discourse and domain constraints as soon as they become applicable. Verbmobil's multi-engine approach, e.g. its use of five concurrent translation engines: statistical translation, case-based translation, sub string-based translation, dialog-act based translation, and semantic transfer. Distinguishing features like the multilingual prosody module and the generation of dialog summaries are highlighted. Verbmobil has successfully met the project goals with more than 80% of approximately correct translations and a 90% success rate for dialog tasks.

2.2.5 Complexity of Dialogue Systems

As information services and domains grow more complex, the complexity of dialogue systems increases. They tend to need more and more domain knowledge and the domain reasoning mechanisms also have to become more sophisticated. Utilizing domain knowledge reasoning is in many cases necessary for a dialogue system to interpret and respond to a request in an intelligent manner, especially as requests can be vague and sometimes ambiguous. This involves not only requests for information from application specific knowledge sources, but also requests related to the properties and structures of the application and requests that are outside the scope of the application. Thus, dialogue systems must be able to access, gather and integrate knowledge from various domain knowledge sources and application systems in order to determine the precise meaning of a request and produce an appropriate response.

2.2.6 Evaluation of Dialogue Systems

Interactive Dialogue systems are based on many components technologies: natural language understanding, natural language generation, and database query language. While, there has been a great deal of progress in developing well-understood evaluation metrics for many of these components, there has been less progress in developing metrics and frameworks for evaluating dialogue system that integrate all these components.

One problem is that dialogue evaluation [31] is not reducible to transcript evaluation, or to comparison with a wizard's reference answers, because the set of potentially acceptable dialogues can be very large. Another problem is that there are many potential metrics that can be used to evaluate a dialogue system. For example, a dialogue system can be evaluated by measuring the system's ability to help users achieve their goals, the system robustness in detecting and recovering from the errors of understanding, and the overall quality of the system's interaction with the users. It is not clear how different metrics overlap with one another, or what the tradeoffs between metrics might be.

Dialogue metrics can be classified as *objective* or *subjective*. *Objective metrics* can be calculated without recourse to human judgement, and in many cases, can be logged by the dialogue system so that they can be calculated automatically. Objective metrics that have been used to evaluate a dialogue as a whole include:

- Percentage of correct answers with respect to a set of reference answers
- Percentage of successful transactions or completed tasks
- Number of turns
- Dialogue time or task completion time
- Mean user response time
- Mean system response time
- Percentage of diagnostic error messages
- Percentage of “non-trivial” (more than one word) requests
- Mean length of “non-trivial” requests

Subjective metrics require subjects using the system and/or human evaluators to categorize the dialogue or turns within the dialogue along various qualitative dimensions. Because these metrics are based on human judgement, such judgements need to be reliable across judges in order to compete with the reproducibility of metrics based on objective criteria. Subjective metrics that have been used to evaluate dialogue systems include:

- Percentage of implicit recovery requests (where the system uses dialogue context to recover from errors of partial recognition or understanding)
- Percentage of explicit recovery requests
- Percentage of contextually appropriate system requests
- Cooperativity (the adherence of the system’s behavior to Grice’s [12] conversational maxims)
- Percentage of correct and partially correct answers
- Percentage of appropriate and inappropriate system directive and diagnostic utterances
- User satisfaction (user’s perception about the usability of a system, usually assessed with multiple choice questionnaires that ask users to rank the system’s performance on a range of usability features according to a scale of potential assessments)

CHAPTER III

DESIGN OF THE MULTILINGUAL RDQA SYSTEM

3.1 System Architecture

The architecture of the multilingual restricted domain QA system is shown in Figure 3.1. The language-specific components have been shown as double line boxes while language neutral components have been shown as single line boxes. The user specifies the query language while starting a dialogue with the system. The language specific shallow parser receives each question in the dialogue for analysis. It tags the input query morpho semantically using the domain ontology present in the domain and linguistic models. These tagged words are assembled into chunks that represent station name chunks, train name chunks, keyword chunks etc. The keyword chunks and sometimes information chunks are used to identify the topic of the query, i.e., the query frame. The domain model includes the words (or chunks) that occur in the input query in each language describing train name, station name etc. together with the type of the chunk. It also includes the default value rules and the interpretive rules in each language. The linguistic model for each language includes the inflections, post-positions or route words and the keywords for handling the inflected words and to identify the query topic unambiguously. The user model includes the user details like name, service, designation, office and home address and the journey class frequently traveled by the user among others that are considered by the system while predicting the user intentions or making recommendations to the user. The Railway information database stores numeric attribute values like time, fare, train number etc. in English and non-numeric attribute values like train name, station name in English and other languages. Dialogue Manager (DM) is used to obtain missing information in the user query from the user model or the dialogue history or to generate a sub dialogue with the user or to present a clarification request with the user in case of ambiguity in the user input. Once we get all the required information directly or through sub dialogue from the user, SQL generation procedure corresponding to the query frame is called to generate all necessary SQL statement(s) using the information chunks.In this system, two aspects of dialogue are modeled by the DM, a generic description of how the dialogue can be interpreted by the dialogue model [Flycht-Erikson et al., 1999] and the representation of resulting dialogue in the dialogue history [Flycht-Erikson et al., 2000] that contains previous information. DM also keeps track of the anaphoric / elliptical queries from the user that constitutes the dialogue

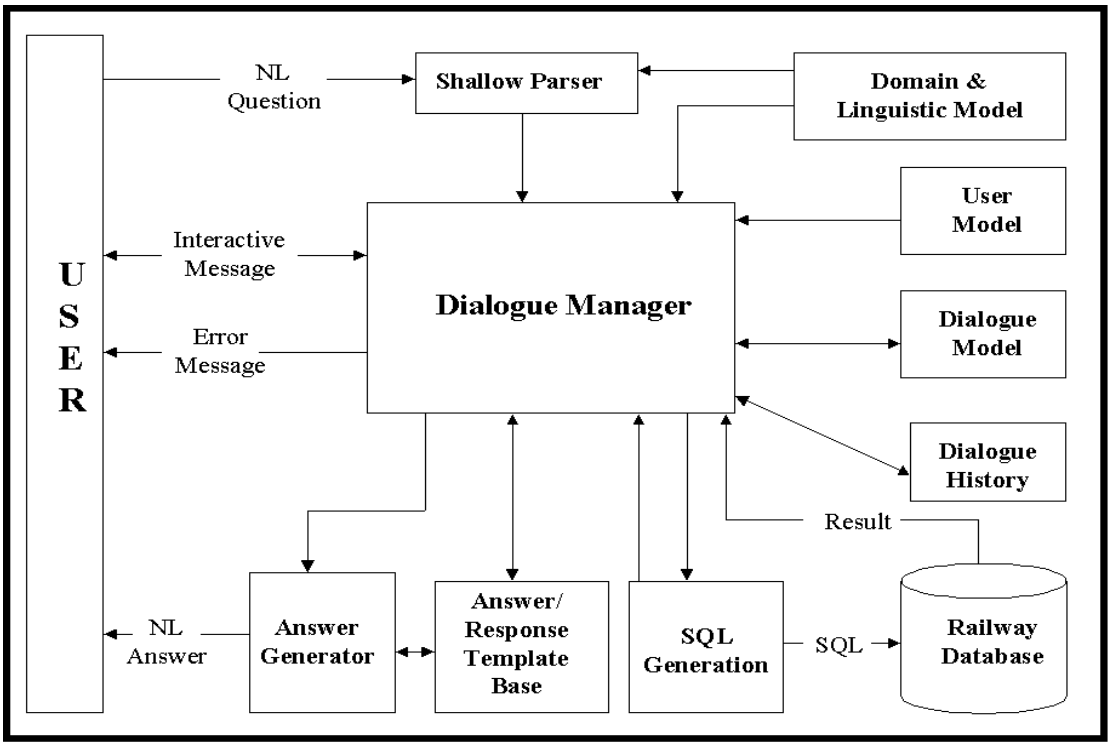


Figure 3.1. System Architecture

using dialogue history information. Unlike in monologue, most important character of dialogue is grounding, where user and system need to establish a common ground on their interaction. If there is a lack in grounding, like system identifying a wrong query frame or user provides wrong information (like missing train name, source/ destination station names etc.), system sends an error message indicating the error in the user query through DM. The SQL statements are used to retrieve the correct answer from the database. This result is forwarded to answer generator via DM to generate a natural language answer. In case, SQL statements generate a null response from the database, the DM will send a cooperative message depending on the user query. The language-specific Answer/Response Template base is consulted by the Answer Generator to retrieve the appropriate answer templates and also by the DM to retrieve the appropriate response templates.

3.2 Domain and Linguistic Model

Domain model holds the knowledge of the world that is talked about. Information from the domain model is primarily used to guide the semantic interpretation of user's

requests; to find the relevant items and relations that are discussed, to supply default values, etc. The knowledge presented in this domain model is coupled to the background database system.

The amount of domain knowledge needed in a dialogue system differ depending on the domain and the system's task. This means that the domain model can range from a rather simple conceptual model to full-fledged world model capable of complex reasoning.

We use a fairly conceptual domain model to make the question answering system natural and intuitive to use. For a system operating on a restricted domain, domain model is quite obvious since it will greatly improve the disambiguation and parsing. To be able to parse the user dialogue and decide how to continue a dialogue, the system must need to have knowledge about the domain. So, we integrate such a kind of domain knowledge in the domain model.

The Railway database is stored in a relational model that contains the information about Indian railways. The main tables used here are schedule table for each train, fare tables for special trains that have a different fare structure, Route tables for each route and tables that include train running frequency details etc. Some temporal tables are maintained in order to check the status of a purchased railway ticket (which is known as checking the Passenger Name Record or PNR status of the ticket) and reservation availability information of a specific journey of a particular train. The words that occur in the input query for Railway information system includes words describing train name, city name / station name, reservation class, and date and/or period of journey or keywords etc. Hence the domain model contains look up tables in each language with these words and the corresponding train name, station name, alias train name and station name etc. The domain knowledge is coupled with the background database system. The domain knowledge also consists of two kinds of rules for each language: Default value rules supply default values for information not specified by the user, e.g. "I would like to go on the 6th", the current month is taken as the default (or the next month if the 6th has already passed), Interpretive rules transform vague qualitative values given by the user into more precise quantitative values used by the system, for example "I want to go this morning" is transformed into "I want to go today between 6 am and 12 noon" [Bennacef et al., 1996]. The system maintains profiles for each registered user in the User model that includes name, service, designation, office and home address and the journey class frequently

traveled by the user among others in order to suggest source or destination station names, class of journey etc. in case of incomplete data from that user. The linguistic model for each language consists of an inflections table which contains both noun and verb inflections related to the domain to handle the inflected words in the query, and post-positions or route words table to identify station name under correct category. A list of keywords is kept in the keywords table to identify the query frame without any ambiguity. This model also maintains some set of orthographic rules (spelling rules) to identify the root word(s) from the surface level word(s).

The words that occur in the database query for Railway information system includes words describing train name, station name, reservation class, and date and/or period of journey or keywords that specify the topic of the query. Hence the domain model contains look up tables for train name, station name, alias tables for train name and station name (i.e. city name) [for example, Howrah-Barabil Janashatabdi is popularly known as Tata Janashatabdi. Similarly, Kolkata city has 3 Stations, namely Howrah, Sealdah and Kolkata] and Reservation table, which consists about reservation class names.

The *Train Name* table stores the name of the trains in Bengali and their numbers. The table has the following fields.

Train_Name -> Name of the train in Bengali.

Train_No -> Train Number.

For example, for Darjeeling Mail, it stores the number as 2343+2344.

In the *Station Name* table we have stored the station names in Bengali. The table has the following fields.

Station_Name - Name of the station in Bengali.

EStation_Name- Transliterated name of Station name in English. (Since we are storing our database in English).

In the *Reservation Class* table we have stored the name of the reservation classes in Bengali and their aliases (i.e., the short name of the reservation class in English).

Reservation_class - Name of the reservation class in Bengali..

Alias - Short name of the reservation class in English.

In the *City Name* table we have stored the station names in Bengali. The table has the following fields.

City_Name - Name of the city in Bengali.

Stations- Names of all the possible stations.

ECity_Name-> Transliterated name of city name in English.

The elaborate view of above tables and other tables used in this model are detailed in chapter 4. In our question answering system, there is no need of separate explicit system model; it is part of domain model itself. The domain knowledge also consists of two kinds of rules:

Default value rules supply default values not specified by the user. For example, if the departure date has not been specified, *"I would like to go on next Monday"*, the date of next Monday is calculated based on the current date.

Interpretative rules transform imprecise values given by the user into appropriate ones used by the system. For example, the utterance *"I want to go this morning"* is transformed into *"I want to go today between 6 am and 12 noon"*. [4].

Similarly, linguistic model (part of Domain and Linguistic model) consists of domain related linguistic knowledge used in the processing of our natural language (NL) query and in the generation of NL answer as well. It consists of Bengali inflections table, which contain both noun and verb inflections like ĉ-ý (te), ĉ¤ (se), *etc.*, which are related to the domain to handle the inflected words (for ex: œý¢ˆƤýšĉŸĉ ... ĉƟ›åĉ¤ (expresse) in the query, and post positions table, that consists of post positional words like ĉ—ĉˆý (*theke* [from]) to identify station/city name as source station/city (for ex: êšŦýêˆÈ¢í ĉ—ĉˆý (New Delhi theke [from New Delhi])).

It also consists of Bengali weekday names like ĉ¤ĉŸ›ƥĉƒ›ƥ (*somavaar* [Monday]), Ÿiŝ¢›ƥĉƒ›ƥ (*mangalavaar* [Tuesday]) etc. A Bengali Month table is maintained to identify the month name correctly in the user query and a period table used to identify the period of journey properly. Bengali numbers table is kept to identify the date of journey correctly. (For ex: ›ƒ/¢ĉ ĉŸ(*porla may* [May First]). We kept a list of keywords in the keywords table to identify the query frame without any ambiguity.

This model also maintains some set of *orthographic rules* (spelling rules) to identify the root word(s) from the surface level word(s) (example spelling rule is shown in later section). To generate a user understandable sub dialogue, some type of

linguistic knowledge is also maintained in linguistic model to reason about the domain and to make the system smarter.

3.3 Railway Database Management

The system as a whole is engaged in data access, and is a hybrid system with subsystems to analyze the natural language query and formal query language SQL, and a data retrieval and database management system. The *database* is structured and contains the information to provide the railway information service. For example in a Railway information system, database contains information about the arrival/departure time of trains, their fares and their running information etc. The aim of database management is to describe the information, in order to offer the service.

We maintained our database in a relational model. Because today relational model is the primary data model for commercial data-processing applications. It has attained its primary position because of its simplicity, which ease job of the programmer as compared to earlier data models such as the network model or the hierarchical model. The relational model stresses on data independence. This means that the user and front-end programs are effectively isolated from the actual database organization.

The main tables used here are schedule table for each train, fare tables for special trains like Rajdhani, Shatabdi etc. that have a different fare structure, Route tables for each route and tables that include train running frequency details etc. Some temporal tables are maintained in order to check the status of the railway ticket (which is known as checking the Passenger Name Record or PNR status of the ticket) and reservation availability information of a particular train. Implementation details are discussed in chapter IV.

3.4 Multilingual Shallow Parser

This question answering system generally does not need all the information from the user input query, so a partial or shallow parsing of the input sentence is more accurate and more robust than deep or full parsing [22]. Shallow parsing provides the structural basis for NL questions. Since there is no restriction in the user's NL input query and the QA system is concerned about the tagging of input words using the domain knowledge present in the domain model and the linguistic knowledge present in linguistic model, we are doing shallow parsing at the semantic level not at syntactic

level. It is roughly used to extract the semantic frames from the input NL query. In our system, shallow parsing [25] focuses on local semantic structure to reduce search space by analyzing the input NL questions. We chose shallow parsing as our benchmark task to analyze the input NL query. Shallow parsing generates semantically tagged output for the input question.

Our shallow parsing task is a combination of morphosemantic tagging (identifying the morphosemantic tags like train name, station name, reservation class etc. as well as the associated inflections and / or post positions, instead of specifying the internal structure of the input query), chunking (finding the fragments in the input query that relate to source/destination station names, train names etc. from the morphosemantic tags) and query frame decision (keywords to identify the query frame).

3.4.1 Morphosemantic Tagging

Morphology is the study of the way words are built up from smaller meaning-bearing units, *morphemes* .A morpheme is often defined as the minimal meaning-bearing unit in a language. The root words are extracted from the inflected words using the inflections tables and orthographic rules during morphological analysis of the input. The tagger uses morphosemantic tags like train name, station name, reservation class name etc. as well as the associated inflections and / or post-positions to tag the input query using the domain and linguistic models. The tagger also identifies keywords in the input query that are used to identify the query topic. Some words may not be found in the domain model knowledge. Those words do not contain any semantic information and are simply discarded. For example, In the Bengali query ... [When the Rajdhani Express will reach Delhi], the word Delhi is not inflected as it matches with the city / station name.

Since the QA system is for railway information, NL query have morphosemantic tags like <Train_Name>/<Train_No> for train name/number, <Station_Name>/<City_Name> for station name/city name, <Reservation_Class> for reservation class, and <Date> for date or <Period> for period of journey etc. and linguistic tags like <Keyword> for keywords, <Post_Position> for post positional words, <Pre-Position> for the pre positional words,<Route_Word> for the route words and<Inflection> tag to indicate the inflections. It may happen that some words

may not be found in the domain model knowledge. Those words do not contain any semantic information and are simply discarded. Shallow parsing uses the extensive domain and linguistic model knowledge to tag the words present in the NL query.

3.4.2 Chunking

Chunking is a process of finding semantically related non-overlapping groups of words from the morphosemantically tagged output. The direct database related information is present in the form of chunks. Each chunk containing a head and (nearly) all of its pre modifiers, exuding arguments and post modifiers. Here chunking is based only on the morphosemantically tagged information. The proper chunking of input words helps in improving the efficiency of the RDQA system by finding out the chunks.

For example, <Station_Name> chunk is morphosemantically tagged like <Station_Name>+<Inflection> or <Station_Name>+ <Post_Position> or <Station_ Name>+<Route_Word>. We are doing chunking, in order to identify the category of station name (i.e. Source station/Destination station/Route station) correctly. According to the above example, it is tagged as <Station_Name>+<Inflection>, so we assumed the station [illegible] (Howrah) is under destination station category (Because there are no postpositional /route words; consists only inflection).

3.4.3 Query Frame Decision

Restricting the query domain and information resource, the scope of the user request can be focused. That is, there are a finite number of expected question topics. Each question topic is defined under a single query frame. Some query frame examples for this Railway information system are fare of a journey [Fare], arrival [Arr_Time]/departure time [Dep_Time] of a train, trains running between important stations [Trains_Imp_Stations], scheduled time [Sched_Time], weekly frequency of a train [Arr_Freq / Dep_Freq], availability of reservation class in a particular train on a particular date [Reser_Availability] and PNR Status [PNR_Enquiry]. This [Sched_Time] query frame deals with the queries related to period or time of journey like "*What are the trains starting on morning from Howrah to New Delhi*" and about starting/ reaching time of journey like "*Which train reaches early*". [Arr_Freq]/[Dep_Freq] query frames deals with questions of frequency related like

"*What are the trains running between Howrah and New Delhi that leave/reach on Monday/all the days*".

Once the question is parsed, using the domain and linguistic model information, shallow parser needs to identify the query frame without any ambiguity. Identification of query frame plays vital role to get the correct NL answer. In some cases, ambiguity will occur i.e. a single natural language query statement may belong to one or more query frames because same keywords are used to identify more than one query frame. We had designed finite state automata (FSA) shown in figure 3.2 to identify the query frame without any ambiguity.

The working of FSA is as follows: The keywords present in the user input NL query are gathered and put in the form of a list. Initially we check whether the words in the list are of *Fare* type (questions having word like ìýô"ýô (b*hara* [fare])), then FSA moves from state q_0 to q_4, concluding that the query is under [Fare] query frame. If the list contains, words are of *How many* type (questions having word ˆý-ýŠÚòŧô (kotogulo [how many])), then we conclude the query is under [Reser _Availability] query frame moving FSA from state q_0 to q_1. If it contains words like 10 digit PNR number, then we conclude the query is under [PNR_Enquiry] query frame moving from state q_0 to q_5. Otherwise it considers the other cases.

The description of other cases is as follows. For example, keywords like ßìô≠ßìô (*Jaoa* [*go*]), Èôxô (asa [come]), òõô"ëþôòšô (pouchano [reach]), and ëþôßÞþô (*chara* [start]) etc. are used to identify both the time related query frames like [Arr_Time], [Dep_Time], and what related query frames like [Trains_Imp_Stations], [Arr_Freq]/ [Dep_Freq], [Sched_Time] respectively. To resolve this ambiguity, we consider *what/which* type of questions (question having words êˆý (ki [what]), òˆýôš (kon [which]) etc.), like êš"Fýê˜Ëŧí òòˆý ¦þô≠ßÞþô ßìôÞþôßÞþ êˆý òßÞþŧ Èôòô Ëëþ (*newdelhi theke howrah jabar kon rail ache* [What are the trains start from New Delhi to Howrah]), are under what

related query frames (above example query is under [Trains_Imp_Stations] query frame), in which FSA moves from state q_0 to q_3, where

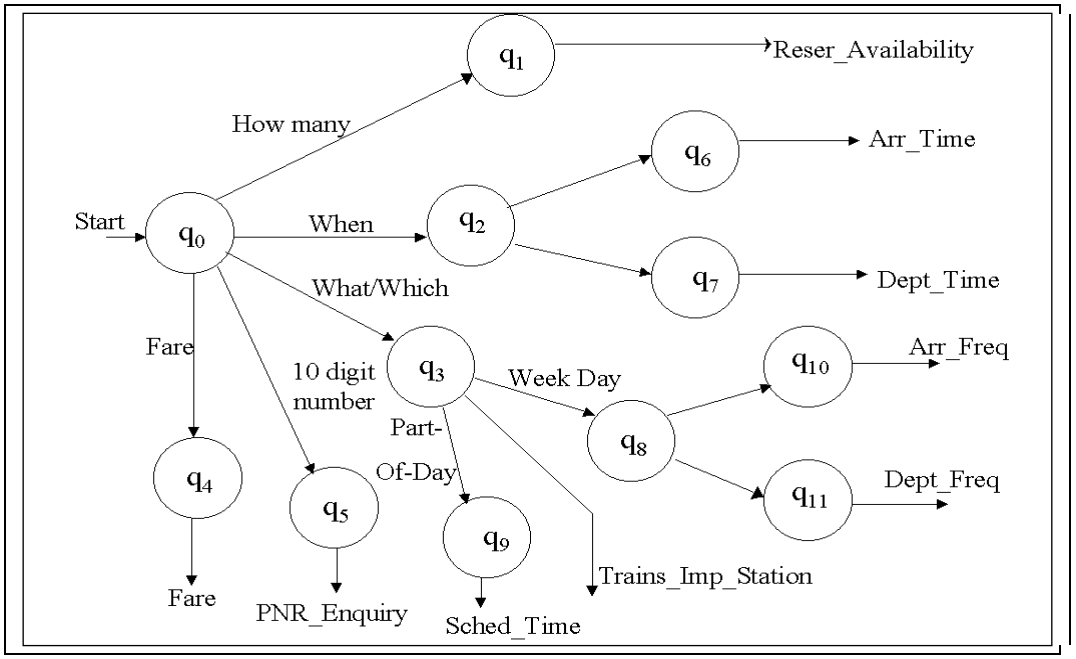


Figure 3.2: FSA for identification of query frame

as, *when* type of questions (questions having words ^ý‰š (kokhan [when]), è^ýêš ¤ÄÿßÍ (*kon somay*[at what time]) etc.), like ^ý‰š è^ýêt^ýê–ýê ßpê™pêší . . . - èÐ›å¤Ð ëpê"ýèßp (*kokhan kolkata rajadhani express charbe* [When Kolkata Rajdhani Express starts]), are under time related query frames (above example query is under [Dep_Time] query frame) i.e. FSA moves from state q0 to q2.

Similarly, along with keyword *what*, if the query contains weekday names like è¤êÄÿJpêßJp (*somavaar* [Monday]), ÄÿJ¿Š¢Jpêß Jp (*mangalavaar* [Tuesday]) etc, then we consider, the query is of *frequency* type and moves to state q_8 from q_3. From state q_8 if it contains the keywords used in [Arr_Time] query frame, then the query is under [Arr_Frequency] otherwise, it is under [Dep_Frequency] query frame.

Along with the *what* keyword, if the query contains words like ¤^ýêt (sokal [morning]), ¤Ç™pê (*sondha* [Evening]) etc, FSA moves from state q_3 to q_9, thus

concluding the query is under [Sched_Time] query frame. Otherwise, the query is under [Trains_Imp_Stations] query frame.

The selection of query frame has a great influence on precision of the system, while there is not much likelihood of errors in other processes, such as getting the information from the dialogue history or generating SQL statement(s) from the selected query frame and/or retrieving the answer from the database and generating natural language answer from the retrieved result.

Consider the **example query,** ^ý‰š ~ôêÁ¬¢ê¨ ò¨Ÿ¢ êš"ƒý¬¢›ô¦ƒp Š Úê"àý ò›õ"ëpôò¬pp (kakhan Darjeeling Mail New Jalpaiguri pouchabe [When the Darjeeling Mail goes to New Jalpaiguri])shown in figure 3.3, at first shallow parser morphosemantically tags the input NL query. The chunks, which contain the exact database query related information, are to be identified from the morphosemantic tags followed by query frame decision.

Question:
^ý‰š ~ôêÁ¬¢ê¨ ò¨Ÿ¢ êš"ƒý¬¢›ô¦ƒp Š Úê"àý ò›õ"ëpôò¬pp (kakhan Darjeeling Mail New Jalpaiguri pouchabe [When the Darjeeling Mail goes to New Jalpaiguri])?

⇩

Morphosemantically tagged output:
<Keyword> ^ý‰š(Kokhan[When])</Keyword>
<Train_Name> ~ôêÁ¬¢ê¨ ò¨Ÿ¢(Darjeeling Mail[Darjeeling Mail)</Train_Name>
<Station_Name>êš"ƒý¬¢›ô¦ƒp Š Úê"àý(New Jalpaiguri[New Jalpaiguri])</Station_Name>
<Keyword>ò›õ"ëpôò¬p(Pouchabe)[arrive])</Keyword>

⇩

Chunking:
(Keyword Chunk)[<Keyword> ^ý‰š</Keyword>](Train Name Chunk)[<Train_Name> ~ôêÁ¬¢ê¨ ò¨Ÿ¢</Train_Name>] (Station Name Chunk)[<Station_Name>êš"ƒý¬¢›ô¦ƒp Š Úê"àý</Station_Name>](Keyword Chunk)[<Keyword>ò›õ"ëpôò¬p</Keyword>]

Chunks identified: ~ôêÁ¬¢ê¨ ò¨Ÿ¢ (Darjeeling Mail) is Train name
êš"ƒý¬¢›ô¦ƒp Š Úê"àý (New Jalpaiguri) is Destination Station
Query frame decision:

Figure 3.3. Shallow parsing of the input query

In the example question shown in figure 3.3, the word ˆý‰š (Kokhan [When]) is tagged as <Keyword>, ˜ĉêÆ¢ê¢¨ ĉŸ¢(Darjeeling Mail) as <Train_ Name>êš˝ý¢›ĉ¦þšÜê˝àý(New Jalpaiguri) as <Station_Name> and ĉ›ĉ˝ëþĉĉþ(Pouchabe) [will go]) as <Keyword>.

In the chunking process, we get the direct database related information like station name with proper category, train name etc. For the above example, chunks identified are under ˜ĉêÆ¢ê¢¨ ĉŸ¢(Darjeeling Mail) as train name (there is no inflection at all).

3.5 Dialogue Manager

A dialogue manager (DM) is that part of a dialogue system that connects the I/O devices (Shallow Parser, Answer Generator) and translators (SQL Generation) (whether they are spoken or typed language, a command language, menu selection, graphical presentation, etc.) to the parts that do the domain task reasoning and performance. Figure 3.1 illustrates the connection of a dialogue manager to the rest of the system. The primary responsibility of the DM is to control the flow of dialogue by deciding how the system should respond to a user request and the coordination of the other components in the system. The NL interpretation module, shallow parser, will need to consult with the dialogue manager and the domain model in order to get a full contextual interpretation of the user's request, while the dialogue manager will be responsible for the manner in which information produced by the other modules get reported back to the user, following the conventions of natural language conversation.

The main task of the dialogue manager (DM) component is to decide on the appropriate way to react to the user input. The reasoning includes recognition of communicative intentions behind the user's requests as well as planning of the system's next action, whether this is database retrieval or a question to clarify an insufficiently specified request. The information retrieval dialogue is divided into three phases: main information exchanges, preceded and closed by formalities. Each dialogue is structured into a number of sub dialogues. Sub dialogues concerning the task are application dependent and include request, response, precision and explanation. If the query frame is incomplete with respect to information needed for database access, a precision sub dialogue generates questions requesting the user to supply specific information. When the query frame is complete, a DBMS query in SQL is generated using specific request generation rules for the query frame. Similarly a response is constructed using the natural language response generation rules and shown to the user. If the user does not respond to a system request for information, but instead asks for an explanation, an explanation sub dialogue is initiated. Similar explanation sub dialogues are generated by the system to explain a null answer and suggest alternatives. For example, if the temporal constraints given by the user are too restricted, the system suggests the closest train to the specified departure or arrival time. The DM uses the Answer/Response Template base to generate the appropriate response in the output. Each query frame expects certain type of information from the user input query. This information is in the form of chunks present in the morphosemantically tagged output.

The dialogue manager is responsible for maintaining the flow of conversation and making sure that the conversational goals are met. For this system, the main goal is to provide the information about Indian railway service. Dialogue Manager controls the flow of dialogue by deciding at high level how the system's interaction should proceed and what questions to ask or clarification requests to make and when to ask or make them and to make the coordination between the other components of the system.

Figure 3.4 shows an outline of our dialogue manager algorithm. This algorithm generates sub dialogues based on grounding, clarification requests or user's discourse goals. Dialogue manager also handles anaphoric references and elliptical expressions. Setting up the dialogue manager strategy we have to take into account

several factors including the dialogue flow, the clarification requests, and the discourse goals of user, the helping features and the response generation.

DIALOGUE MANAGER

while conversation is not finished
if user has completed a turn
then parse the user's request
if system has clarifications
then address the clarification requests
else if system has turn
then if system has intended dialogue acts
    then generate a sub dialogue with the user
    else if discourse goals are unsatisfied
    then address goals
end

Figure 3.4: Dialogue Manager Algorithm.

Response Generation

The types of responses can be generated during the dialogue are requests for specific information (precision sub-dialog), responses, sub dialogue generation. At each user dialog act, the response generator, which is part of the dialogue manager, builds a sentence where gaps are filled in from the content of the current semantic frame, the dialog history, domain and linguistic model information and the DBMS response. Careful attention has been paid to construct natural sounding sentences that contain the appropriate contextual information, when possible, summarized in a single sentence.

Each query frame expects certain type of information from the user input query. This information is in the form of chunks presented in the morphosemantically tagged output. If the required information was not present in the user query, Dialogue Manager (DM) sends a request to user asking the required information. Table 2 details

about the required chunks for each query frame. Until, we get all the chunks from the user, we cannot proceed to the other procedures like SQL generation, Answer generation etc. This information is kept under dialogue manager.

Query Frame	Needed Chunks
[Arr_Time]	Destination station, Train name/number
[Dep_Time]	Source station, Train name/number
[Fare]	Source and Destination station/ Distance between stations, Train name/ number and Class of journey
[Trains_Imp_Stations]	Source and Destination station
[Arr_Freq]	Weekday of journey, Source and Destination station
[Dep_Freq]	Weekday of journey, Source and Destination station
[Sched_Time]	Period/ time user wants to travel, Source and Destination station
[Reser_Availability]	Date of journey, Train name/number, Reservation class name
[PNR_Enquiry]	10 digit PNR number

Table 2. Chunks needed for each query frame.

For example, [Arr_Time] query frame expects, user will specify train name, destination station/city name. If user didn't mention that information, DM asks questions about the information by generating a sub dialogue which sounds natural language. DM utilizes a variety of knowledge sources and models in order to handle these situations.

Let us consider an example dialogue shown in figure 3.5. At first, user asks a NL question. After analyzing the user query at U_1, system identifies that the query is

under [Fare] query frame, because of the keyword ìýô "ýô (b*hara* [fare]). For [Fare] query frame, system expects user must provide information like source station/city name, destination station/city name, train name and the reservation class, in which user needs fare details. But in the user request U_1, he mentioned only source [ê£ƒ/ô¢˜ô (Sealdah)] and destination station [êš"ƒýৎ¢›ô¦ƒþŠ˜Ûê"àý (New Jalpaiguri)], there is no train name and reservation class for which he needs fare details. So, system generates a sub dialogue like ê£ƒ/ô¢˜ô ò—ò^ý êš"ƒýৎ¢›ô¦ƒþŠ˜Ûê"àý ›/¢Jý òƒ'Jþ¢ ÒôòÐëþ (*sealdah theke newjalpaiguri porjonto rail ache* [For which train going from S*ealdah* to N*ew Jalpaiguri*, you want information]) with the user, asking for the train name, in which he needs fare details with natural sounded response, using response generator.

For the system request, user provides train name as ˜ôêòÃৎ¢ê¢¨ òÝ¢ (Darjeeling Mail) at U_2. Once train name is obtained from the user, at S_2, system asks for the reservation class he needs fare details, user asks for the sleeper class at U_3. Once all this information is obtained, SQL generation procedure is called to generate the necessary SQL statement(s) for retrieving the information the database.
At S_3, using the retrieved result from the database (210 in this case), system generates NL answer like ˜ôêòÃৎ¢ê¢¨ òÝò¢ ê£ƒ/ô¢˜ô ò—ò^ý êš"ƒýৎ¢›ô¦ƒþŠ˜Ûê"àý›/¢Jý ê'ƒýê^ýòÐ'ƒýƒ'Jþ ìýô"ýô ¦þ¢ J¢ìòƒ'JþP ^Ôýô¤ò-ý 210 'ƒýô^ýô (darjeeling maile *sealdah theke new jalpaiguri porjonto tikiter bhara holo sleeper classe 210 taka* [Fare from S*ealdah* to N*ew Jalpaiguri* by Darjeeling Mail in Sleeper Class is Rs.210]) using template based answer generator.

After then, user asks an elliptical question at U_4, this elliptical expression is handled by using dialogue history information, similarly, at U_5, user question is an anaphoric expression.

U_1: ^ý-ý ìýô "ýô êÆß/ô¢ ˜ô ò—ò^ý êš"ÿý৭¢›ô¦ ÿþŠƯê "àý ›/¢Ĵý (koto bhara *sealdah theke newjalpaiguri porjonto* [What is the fare from *Sealdah* to New Jalpaiguri]).

S_1: êÆß/ô¢ ˜ô ò—ò^ý êš"ÿý৭¢›ô¦ ÿþŠƯê "àý ›/¢Ĵý ôß̀Ɉþ¢ ÒôòȾëþ (*sealdah theke newjalpaiguri porjonto rail ache* [For which train going from S*ealdah* to N*ew Jalpaiguri*, you want information]).

U_2: ˜ôêÆ৭¢ê¢¨ òŸ¢ (Darjeeling Mail).

S_2: ˜ôêÆ৭¢ê¢¨ òŸò¢ß/þ ò^ýôš ^Ôýô¤ (darjeeling mail*er kon class* [For which class, you need fare details in Darjeeling Mail])

U_3: Ĵ¢í›ôß/þƤ ^Ôýô¤ (*sleeper class* [For Sleeper Class)

S_3: ˜ôêÆ৭¢ê¢¨ òŸò¢ êÆß/ô¢ ˜ô ò—ò^ý êš"ÿý৭¢›ô¦ ÿþŠƯê "àý›/¢Ĵý ê"ÿýê^ýòȾ"ÿýßɈþ ìýô "ýô ¦þ¢ Ĵ¢í›ôßɈþƤ ^Ôýô¤ò-ý 210 "ÿýô^ýô

(darjeeling maile *sealdah theke new jalpaiguri porjonto tikiter bhara holo sleeper classe 210 taka* [Fare from S*ealdah* to N*ew Jalpaiguri* by Darjeeling Mail in Sleeper Class is Rs.210])

U_4: –áýí"ÿýôß/ôß/þ . . . ê¤Ÿò-ý (*threetier acte* [and, in Three Tier AC]).

S_4: ˜ôêÆ৭¢ê¢¨ òŸò¢ êÆß/ô¢ ˜ô ò—ò^ý êš"ÿý৭¢›ô¦ ÿþŠƯê "àý›/¢Ĵý ê"ÿýê^ýòȾ"ÿýß/þ ìýô "ýô ¦þ¢ –áýí"ÿýôß/ôß/þ . . . ê¤ò-ý 591 "ÿýô^ýô

(darjeeling maile *sealdah theke new jalpaiguri porjonto tikiter bhara holo three tier acte 591 taka* [Fare from S*ealdah* to N*ew Jalpaiguri* by Darjeeling Mail in Three Tier AC is Rs.591]).

3.6 User Model

The Figure 3.5: A simple example dialogue

the User model to reason about the incomplete data. The system generates precision sub dialogues to authenticate these guesses with the user. If the user responds positively, the system generates the answer. In case of negative response, the system displays possible alternatives and the user selects the appropriate information.

In this model we maintain information about travelling details for each and every registered user. These details are stored in one table is called booking history details. In this table, take into account all the information like station_from, station_to, date, train name, train no and reservation class etc. The user already travelled from sealdah to new jalpaiguri for sleeper class in darjeeling mail, So we maintian one record in booking history table, that details are considered by the system while predicting the user intentions or making recommendations to the user.

3.7 Dialogue Model

The dialogue is made up of turns in which typically a question, as clear as possible, is made to guide the user for the kind of answer expected by the system in each turn. Once the user has answered, the system processes the input and initiates the next turn that can contain another question or a message to the user. The general information about the construction of a dialogue, held by the dialogue model, is often based on the representation of relations between the constituents of dialogue. This knowledge is used to control the dialogue, i.e. to decide what action to take in a certain situation.

In this dissertation, the dialogue model refers to the ways in which the dialogue is implemented from the point of view of the system. This may correspond to the overall structure of the dialogue flow, as perceived by the user. Similarly, the implementation of dialogue model may differ from the mental model what the user has. As one of the main focuses of this dissertation is on system architecture, it is

meaningful to define the dialogue model to cover all those ways that are used to represent the organization of dialogue components and interaction between those components.

Finite-state machines are logical choice for dialogue management in many dialogue system applications. There are many reasons for the popularity of state machines. First of all, they are well known and widely used model and has a strong theoretical background, good properties, from the computational viewpoint, it is easy to understand and straightforward to use. In its most basic form, a finite state machine consists of a set of nodes representing dialogue states and set of arcs between the nodes. Arcs represent the transitions between states. The resulting network represents the whole dialogue structure, and paths through the network represent all the possible dialogues, which the system is able to produce.

Our system supports finite-state dialogue model. In our model, each dialogue state is defined with three parameters: clarification requests, intended dialogue acts, and discourse goals. Each action results in a new dialogue state. The clarification requests (CR) [29] are generated by the system in case of ambiguity, in order to achieve grounding with the user. The intended dialogue acts (IDACT) [19] are generated by the system to get the missing information for a particular query frame from the user. These CR and IDACT are generated through sub dialogue by the DM to user. Discourse goals (DG) [28] keep track of the underlying activity related tasks still necessary to fulfill, which are identified during user interaction with system. We put the unfulfilled discourse goals in a stack structure.

The dialogue model is used to classify the current request into one of the dialogue act categories, and predict the next dialogue act. Here each state represents the system response and the arcs represent user inputs, which move the dialogue from one state to another.

Dialogue model description for the above example dialogue shown in figure 3.5, is as follows. At state 1 (after U_1, but before S_1), system has no CR. System has intended to ask about train name and reservation class name, for which user needs fare details. So, those are put under IDACT. The discourse goals are fare calculation and finding the trains between stations Sealdah and New Jalpaiguri. We put finding the trains between stations as one discourse goal, because, even user didn’t mention the train name, we need to find the trains between those stations and asks the user to select one, in which he needs fare details. These goals are put in the stack form. Once

the user provides the train name (by selecting one in a list of trains), it enters into state 2.This description is shown in figure 3.6.At state 2 (after U_2, but before S_2), system still has one more IDACT, i.e. asking the user for reservation class name. After getting the reservation class name from user, system enters into state 3.At state 3 (after U_3, but before S_3), system finds fare of the Darjeeling Mail in Sleeper class, making a state move form 3 to 4. At state 4 (at S_3), system generates a NL answer. During generation of NL answer, there won't be any CR, IDACT, and DG, because system doesn't have any obligations.

State 1 (After U_1, but before S_1):

CR: None.

IDACT: Train name and Reservation class [User didn't mention the train name and class name in which, he needs fare details].

DG: Fare calculation; Find trains between Sealdah and New Jalpaiguri stations.

State 2 (After U_2, but before S_2):

CR: None.

IDACT: Reservation class.

DG: Fare calculation; [DM generates a sub dialogue, giving list of trains running from Sealdah to New Jalpaiguri, user asks for Darjeenilg Mail].

State 3 (After U_3, but before S_3):

CR: None

IDACT: None

DG: Fare calculation [System asks for the reservation class in the Darjeeling Mail for which he needs fare details, user asks fare details for Sleeper Class].

State 4 (At S_3): The Answer generator generates NL answer.

State 5 (After U4, but before S_4):

CR: None

IDACT: None

DG: Fare calculation (Here user dialogue is an elliptical expression; asking the fare of Three Tier AC, system gets proper information like query frame, source and destination names and train name from dialogue history).

State 6 (At S_4): The Answer generator generates NL answer

State 7 (After U_5, but before S_5):

CR: None

IDACT: None

DG: Departure time (Here, user dialogue is an anaphoric expression; user asks for departure time of Darjeeling Mail, system gets proper information like train name and source station from dialogue history).

Figure 3.6.Dialogue state description for theexample dialogue shown in figure 3.5.

At state 5 (after U_4, but before S_4), system's discourse goals are again filled up by the fare calculation. But at this state, system doesn't have any CRs and IDACTs even the user question didn't contain the necessary information, because system gets the proper information from the dialogue history. Using the dialogue history information and the current dialogue information, system calculates the fare of Darjeeling Mail in Three Tier AC making a move from state 5 to 6, generating NL answer.

Similarly, at state 7(after U_5, but before S_5), system's goals are filled by finding the departure time of Darjeeling Mail. Since dialogue history is providing the train name and source station details, there won't be any CRs and IDCTs. System finds the departure time by making a move from state 7 to 8. At state 8, system generates a NL answer using template based answer generator.

3.8 Dialogue History

As a basis for the above tasks, dialogue history represents the state of dialogue, which records focal information, i.e. what has been talked in the past and what is being talked in the present. It is used for dialogue flow control and disambiguation of context dependent requests and context sensitive interpretation like anaphoric and elliptical expression resolution.

Our system focuses only on the objects that have been mentioned in the attention level of discourse structure. We maintain a history of semantic frames, which contains information about previous chunks and their types as well as other dialogue information like answers retrieved by the current SQL statement(s) and the answers for the previous queries in the dialogue.

In case of incomplete information in the user query, semantic frames corresponding to the current request are completed from the *dialog history*, in which, it take into account all the information like source, destination stations/cities, departure day, class and train name of the journey etc given by the user previously.

In every-day discourse, people use incomplete (*elliptical*) sentences, the meaning of which is complemented by the discourse context. This phenomenon is illustrated in the following dialogue between the user and system from the example dialogue shown in figure 3.5 at U_4.

U_4: *-áýí"ƒýôƒs/ôƒs\/þ . . . êx¥ò-ý* (*threetier acte* [and, in Three Tier AC]). Here user asks about fare details for the Three Tier Ac in the Darjeeling Mail running from Sealdah to New Jalpaiguri. System gets the information like query frame ([Fare]), Train Name (Darjeeling Mail), Source Station (Sealdah) and Destination Station (New Jalpaiguri) from the dialogue history and get the reservation class information (Three Tier AC) from the current request. Using this information, system finds the fare between those stations in Three Tier AC.

Consider the dialogue between the user and system in example 3.5 at the user's request U_5.

U_5 *^ý‰š ëþôƒs\/þò\/þ* (*kokhan charbe* [When it will start]). This is obtained from the dialogue history. The goal of the user is to get departure time of that particular train.

The DM gets semantic frames from the other system components. These frames are filled by interpreting the request in the context of the ongoing dialogue, domain knowledge, and dialogue history. The DM prompts for missing information or sends a SQL query. Before the query is sent off, DM checks whether new information is contained in the query or the information is contradictory to information given before. If this is the case then the DM can either keep the original information or replace it with the new one in the dialogue history or engage in a confirmation sub-dialogue.

The DM looks at the query after language processing has been completed (but before the formal query is issued), as well as after the result has been obtained from the formal query. An important issue is to correctly manage the dialogue history. To do this, it is necessary to be able to add and remove information from the history

because the accuracy of the system mainly depends on the representation of the dialogue history and how the DM responds to the user's dialogue.

We are representing dialogue history in the form of a table, containing information about train name, source and destination station/city name, query frame, answers generated from the present or previous questions etc.

3.9 SQL Generation

For each query frame there is a separate procedure for SQL statement(s) generation. This module needs the chunks generated by the shallow parser, which are stored in dialogue history.

If the chunks are present in the current query, these are used. Otherwise, the chunk information is obtained from the dialogue history. Even if the dialogue history doesn't contain the necessary information, DM generates a sub dialogue with the user asking for the missing information. After getting all the required information from the user, this procedure generates all necessary SQL statement(s), which act(s) on the railway database stored in a relational model to retrieve the appropriate result.

From the example query shown in figure 3.3, according to the keywords in the query, [Arr_Time] query frame has been selected. Next, the SQL generation procedure for retrieving the arrival time is called. The procedure considers that, the user will provide the train name and station name for the query related to arrival time. During the generation of SQL statement(s), SQL generation procedure must need to confirm the train running details like whether user needs details for up/down train, by sending a clarification request (CR) modeled by the dialogue model to user via dialogue manager as shown in figure 3.7. Here user asks about the upward journey i.e. Darjeeling Mail running from Sealdah to New Jalpaiguri (Train No is 2343). Then the SQL statement generated as shown in figure 3.7 is

SELECT Arr_Time FROM Schedule2343 WHERE Station_Name='NEW JALPAIGURI'.

Similarly, for the fare related query, SQL generation procedure would be called depending on the type of train. The procedure considers that the user will provide the train name and reservation class. If the train is of Express type, it considers that the user may provide either the source and destination stations of the journey or the distance between those stations. If it is of Rajdhani type, it considers that the user may provide source and destination station of journey. Similarly for the

other query frames, SQL generation procedure considers that the user may provide the necessary information.

3.10 Answer Generator

Once the SQL statement for an input query statement is generated, it is triggered on the database and the retrieved information is used to represent the answer. The retrieved information is updated in the dialogue history for further reference.

Each query frame has its corresponding answer generator. We have used template based answer generation method. Each template consists of several slots. Those slots are filled by the retrieved answer and the updated semantic information in the dialogue history.

Let us consider the templates for the [Arr_Time] and [Dep_Time] query frames, in which slots are filled by the answer and semantic frames stored in the dialogue history.The slots like [Arrival_Time] and [Departure_Time] are filled by the retrieved result, whereas, slots [Train_Name], [Destination_Station] and [Source_Station] are filled by semantic information updated in the dialogue history.

1. The [Arr_Time] Query frame is selected.

2. The system confirms about up/down journey of the train by sending a clarification request (CR).

3. The user requested for upward train, running from Sealdah to New Jalpaiguri via DM.

4. *SELECT Arr_Time FROM Schedule2343 WHERE Station_Name='NEW JALPAIGURI'*.

5. DBMS returns "08:00 hrs".

6. ০৮:০০ [illegible] (*08:00 darjeeling mail new jalpaiguri pouchabe* [At 08:00hrs D*arjeeling Mail* goes to New Jalpaiguri])

Figure3.7. SQL and Answer Generation for the example query shown in figure 3.3

For the example query shown in figure 3.3, the retrieved result is 08:00 hrs. It is forwarded to the answer generator procedure that generates NL answer shown in figure 3.7. Since the query frame is [Arr_Time], the generated answer is ০৮:০০

˜ĉêĉÆЧ¢ê¢¨ ĉŸ¢ êš˝ቸýЧ¢›ĉ¦ቸþЅ˝Ùê˝àý ĉ›ĉ˝ëþĉĉンþ (*08:00 darjeeling mail new jalpaiguri pouchabe* [At 08:00hrs D*arjeeling Mail* goes to New Jalpaiguri]), which will be sent to the user.

CHAPTER IV

IMPLEMENTATION OF THE MULTILINGUAL RDQA SYSTEM

4.1 Implementation Scheme

The implementation of this QA system was done by using Visual C++ version 6.0 and GistSDK version 3.0 packages as the programming tools and MS-Access 2000 has been used as the database package under windows XP operating system.

Visual C++ is an extremely powerful tool for Windows programming product today. Since we had designed an interactive system, in which user and system communicates through a set of dialogues (here dialogue means a small window tool), we chose VC++ to design dialogues, and make interaction properly.

GistSDK kit a software development kit for Indian languages. The GistSDK consists of keyboard drivers, Indian language fonts, insertable components, Bilingual support for 10 Scripts - covering 11 Indian Languages (viz. Assamese, Bengali, Devanagari (covering Hindi and Marathi), Gujarati, Kannada, Malayalam, Manipuri, Oriya, Punjabi, Tamil, and Telugu) and English, Monolingual support for editing in Sanskrit, Vedic and Grantha, Support for Roman, Three Indian script keyboard layouts, Support to Indian standards (National, Local as well as International standards), API – Application Programming Interface, Utilities, Tools and all that you need to create desktop Indian Language Applications for the PC-User.

Since Ms-Access is a relational model, we created the database in the form of tables. The relational model gives us much flexibility in designing a database schema to model the railway database. Since we are extracting information from database tables, we concentrated more in the design of railway database.

4.2 Design of Domain and Linguistic Model

Domain and linguistic model hold the knowledge of the world that is being talked about. Information from the domain and linguistic model is primarily used to guide the semantic interpretation of user's requests; to find the relevant items and relations that are discussed, to supply default values, etc. The knowledge presented in this domain and linguistic model is coupled to the background database system.

Below are the tables with their design view that have been used in the system's Domain Model.

4.2.1 Table Information

<u>Table:</u> Train_Names

The **Train Name** table stores the name of the trains in Bengali and their numbers. This table has Train_Name and Number as attributes. For example, for Darjeeling Mail, it stores the number as 2343+2344.

<u>Table:</u> Station_Names

In the **Station Name** table, we have stored the name of the stations in Bengali. The table has the fields Station name and transliterated name of station in English (Because we are storing our database in English). For example, for [illegible] (sealdah), it is stored as SEALDAH.

<u>Table:</u> ReservationClass

In the **Reservation Class** table, we have stored the name of the reservation classes in Bengali and their aliases in English. For example, for [illegible] (two tire ac), it is stored as Two Tire AC

<u>Table:</u> Cities

In the **City Name** table, for each city name in Bengali, we have stored all the station names in English separated by a comma and the city name in English. For example, for [illegible] (newdelhi), that corresponding station names are (NEW DELHI, DELHI SARAI ROHILLA, OLD DELHI, HAZRAT NIZAMUDDIN, DELH CANTT)

<u>Table:</u> Routes

In the **Route Name** table, we store the name of the route table for the corresponding route number. For example route number 3 having route table name ROUTE_NJP_HWH.

<u>Table:</u> Bengali_Inflection

Along with the domain knowledge, we are storing some other tables consisting of linguistic knowledge. In the **Bengali Inflection** table, we store both noun and verb inflections in order to identify the root word(s) from the inflected word(s).

Table: Postposition

In order to identify source station/city correctly from the user query, we store all the possible postpositions in the **Postposition** table as. For example, ò—ò^ý (theke)

Table: Route_Words

To identify route station/city correctly from the user query, we store all the possible route words in the **Route Word** table.

Similarly, to identify the weekday names, month names, dates, period of journey correctly, we are maintaining separate tables for each category.

Table: *Dialogue_History*

We represent the ***dialogue history*** also in the form of a table, consists the semantic frames as attributes. Dialogue history stores the present and previous dialogue information. To differentiate present and previous information we kept dialogue id as one attribute. In order to get the pervious query frame or previous query answer, we kept query frame and answer information as attributes.

Table: Keywords

To identify the keywords properly, we kept the **keywords** in the form of a table. For example the keyword ò^ýôš (*kon* [which]) is of *which* type, the keyword /ôfß/ (jay [go]) is of arrival type.

4.3 Railway Database Design

The main tables used in the railway database are schedule table for each train, fare tables for special trains like Rajdhani, Mail/Express, Shatabdi etc. that have a different fare structure, Route tables for each route and tables that include train running frequency details etc. Some temporal tables are maintained in order to check the status of the railway ticket (which is known as checking the Passenger Name Record or PNR status of the ticket) and reservation availability information of a particular train.

We are designing a prototype model for this railway information system, so we consider a very few tables in this railway database. For example, in the Indian railways, there are around 90 routes are there, but we are designing tables for 3 routes only due to time and space constraint. Even though, these tables are more than enough to access information for all types of questions. Some of the possible tables with their design view are shown in below.

4.3.1 Table Information

Table: Schedule

We maintain a separate schedule table for each train running in each direction (upward and downward). For example, table name Schedule2343 indicates the schedule table for the Darjeeling Mail running from Sealdah to New Jalpaiguri, where as, schedule table Schedule2344 used for the Darjeeling Mail running from New Jalpaiguri to Sealdah. To calculate Arrival and Departure time information, these tables's information is useful.

We stored time in the form of a string, because MS-Access is not well supported for time and date functions unlike other relational models like Oracle, SQL server.

Table: Fare

Separate fare tables are maintained for Rajdhani, Mail/Express, Shatabdi and Jan Shatabdi trains; because fare between particular source and destination station differ by the type of train.

Table: for Rajdhani Express trains

In some cases, many Rajdhani trains are running between same source and destination stations. In that case, fare between those stations differed by the route station in which the train is running. For example, Rajdhani trains running from Howrah to New Delhi will go either by Gaya or Patna station. Fare between those stations differs by the route station. So we used an attribute *Via* to indicate the route station. In generally,

Rajdhani Express trains have First AC, Two Tier AC, Three Tier AC reservation classes. That why, we are considering fare details for those classes only.

In another case, fare between two stations is different by the train running between those stations, even many Rajdhani Express are running. So, we used another attribute *By* to indicate the train number. Similarly we maintained fare tables for Shatabdi and Jan Shatabdi Express trains.

Table: for Mail/Express trains

Fare between two particular stations in Mail/Express trains is calculated based on the distance between those stations. We are calculating the fare based on the distance slabs. Fares will be same for all the distances with in *Dist_From* and *Dist_To* slab. So we are keeping the table as shown above.

Table: Train_Index

In some cases, like calculating fare in a particular train, we need source and destination station names, if the user forgets to give that information, system need to get that information by looking at the train name. In order fulfill this type of requirement, we are using source and destination station names as attributes for a particular train and the type of train (Rajdhani, Mail/Express, Shatabdi or Jan Shatabdi) as another attribute, used in fare calculation.

Table: Reservation_Class

In enquiring the reservation availability of a particular train, users ask questions mentioning reservation class names knowingly or unknowingly. If the required reservation class is not presented in that train, system has to respond properly by giving a cooperative message like "*That train doesn't have your required class, but has these classes*". In order to handle these types of situations we need to maintain a reservation class presence table. Here we are keeping Train_No in text format, because we are storing both up and down ward train numbers together in string format. We used the Boolean values (either 1 or 0) to represent the reservation class presence in that particular train for the table shown as above.

Table: GetRoute

In order to find the trains between two important stations, first we need to find out route under which the stations belong. To get this route number information we are keeping this table with attributes source, destination stations and route numbers. Some times, source and destination stations may belong to many routes. So, we keep the route number data separated by comma.

Table: Route

In order to get the actual trains running between important stations, we are maintaining a separate route table. In this route table we kept train number information along with the station names belonging to that route as attributes. The station names are stored in Boolean format. For example look at the route table for NEW JALPAIGURI-HOWRAH route (ROUTE_NJP_HWH) as shown above. Similarly we are maintaining route tables for other routes.

Table: Reser_Avble

In order to get the reservation availability details of a train for a particular class on particular date, we are maintaining static (not distributed) Reser_Avble table that consists details about the number of seats available for each train in the corresponding reservation classes.

For a particular train, only the available reservation classes are filled with appropriate values. For example, Darjeeling Mail has AC2, AC3, and SL classes only. Those classes are filled with the corresponding values, remaining fields are empty.

Table: Frequency

Some times, user asks question about running details of trains, i.e. whether the train starting or reaching on Monday or Tuesday etc. In order to handle these situations, we kept both arrival and departure frequency details in table format, in which each row corresponding to a particular train and its running details in Boolean format (‘1’ indicates the train running on that particular day otherwise not) along with value of number of days running, kept under Arr_Freq attribute.

Similarly like Arrival frequency table, a Departure frequency table is maintained having same number and type of attributes, but instead of Arr in

Arr_Mon, we use Dep as Dep_Mon, which gives departure details of particular train. Similarly for the other attributes also.

4.4 Implementation of User Interface

We are creating the MFC AppWizard (exe) Visual C++ program for the user interface. An MFC program is an executable application for Windows that is based on the Microsoft Foundation Class (MFC) Library. When you use the MFC AppWizard to create an MFC program, you get a working starter program. This program has built-in functionality that when compiled, will implement the basic features of a Windows executable (.EXE) program. The MFC starter program will include C++ source (.CPP) files, resource (.RC) files, header (.H) files, and a project (.DSP) file. The code generated in these starter files is based on MFC.

The MFC AppWizard series is a branching path of either 4 or 6 steps depending on the architecture you select for your program. You can move forward and backward through the steps and make changes to the options you have selected. In step 1, out of three architectures [**Single Document (SDI)**, **Multiple Document (MDI)**, or **Dialog Based**], we choose SDI type program. For this SDI, step 2 is the selecting Database view support for our program. Out of four database support options [**None**, **Header file** support, **Database View (with file support)** or **Database View (without file support)**], we select the **Database View (without file support)** option as database support (because we do not need such an extensive file support). This is followed by selection of Data Source. Out of three [external ODBC (**Open Database Connectivity**) database, DAO (**Data Access Object)** database or **OLE DB** database] possible database connections, we chose for the ODBC data source, because we can use any of the database packages as the backend without having to change or alter the main program for retrieving and manipulating them. This is possible because with each of such package, a driver will be associated which will map the same functions onto the different packages in different ways as required. After then we need to select the appropriate data source.

In database terms, a data source is a specific set of data, the information required to access that data, and the location of the data source, which can be described using a data source name (DSN). Since, we are working with the CDatabase

class, the data source must be configured through Open Database Connectivity (ODBC) Administrator. The data source we are using is a Microsoft Access file in a local directory.

4.4.1 Configuring Data Source

ODBC Administrator is used to configure the data sources available to us locally. To open ODBC Administrator, click **Start**, and then click **Control Panel**. Double-click **32-bit ODBC.** After creating all the tables under a single Ms Access database, we are adding that database in the User Data sources under Microsoft Access Driver by the ODBC (Open Database Connectivity) Data Source Administrator. Once User data source name (DSN) is configured, we need to connect the DSN with the VC++ environment by selecting it under ODBC option mentioned in step 2. The recordset type we are selecting is *dynaset.* A "dynaset" is a recordset with dynamic properties. During its lifetime, a recordset object in dynaset mode (usually called simply a "dynaset") stays synchronized with the data source. Records in our application that are added to or deleted from the recordset are reflected in the dynaset. After establishing the database connectivity, we are completing the remaining steps (step 3 to step 6) of MFC AppWizard with default settings.

4.4.2 Connecting the database tables

While programming the application, **CDatabase** and **CRecordset** classes are used. A **CDatabase** object class provides an abstraction for an ODBC database connection to a data source, through which we are operating on the data source. To use **CDatabase**, we construct a **CDatabase** object and call its **OpenEx** member function. This opens a connection. When we finish using the connection, call the **Close** member function to destroy the **CDatabase** object. A function call **ExecuteSQL** is sometimes called when we need to execute an SQL commands directly.

A **CRecordset** object represents a set of records selected from a data source known as "recordsets". **CRecordset** objects are typically used in two forms: dynasets and snapshots. A dynaset stays synchronized with data updates made by other users. A snapshot is a static view of the data. Each form represents a set of records fixed at the time the recordset is opened, but when you scroll to a record in a dynaset, it

reflects changes subsequently made to the record, either by other users or by other recordsets in your application.

We are creating classes corresponding to the tables present in our domain and linguistic model. For creating such a class, we proceed as follows.

1) We Select the ClassWizard option from View option in Menu of VC++ environment.

2) Then we choose the Class Info tab and click on Add Class option and choose New option.

3) Then we need to give an appropriate name for our class. Also we select CRecordset as the Base class for our class.

4) Then we have to select the DSN name for ODBC that we fixed earlier. Also we select the dynaset option for recordset type.

5) Then the appropriate table from the database is selected. Next we click on OK to generate the recordset class.

Here **CRecordset** class acts as Base class for the newly created classes by the above approach. Since we are creating a class for the table, the created member variables correspond to the attributes in the table. To access the attribute values, we are operating on the member variables.

While constructing **CRecordset** objects for operating on the connected data source, we pass a pointer to our **CDatabase** object in the recordset constructor. Then we call the recordset's **Open** member function, where we are specifying object as a dynaset. Calling **Open** selects data from the data source. After the recordset object is opened, we are using its member functions **AddNew**, **MoveFirst**, **MoveNext**, **MovePrev**, **MoveLast**, **IsBOF**, **IsEOF**, **Edit**, **Delete**, **Requery** etc. and data members to scroll through the records and operate on them. The operations available depend on whether the object is a dynaset or a snapshot, whether it is updatable or read-only (this depends on the capability of the Open Database Connectivity (ODBC) data source), and whether you have implemented bulk row fetching. To refresh records that may have been changed or added since the **Open** call, we call the object's **Requery** member function. We call the object's **Close** member function and destroy the object when we finish with it.

The CRecordset class encapsulates a group of similar records, usually the records within a database table or returned from a query. We are executing SQL queries on database by using CRecordset object's **Open** function. The retrieved value

from the database is stored in the form of recordset records. From that recordset's records we are getting the required value.

4.4.3 Adding Controls to Interface

Since we are working with Indian language Bengali, in order to handle with the Bengali Fonts, we are using the ActiveX controls designed in the GistSDK software. All controls work with ISCII (Indian Standard Code for Information Interchange) 7 bit, 8 bit, PC-ISCII data while displaying using ISFOC font. These controls directly handle ISCII data from the Database. These components are connected in the same manner as Native English controls are connected.

The procedure to insert the controls in our project is

1) Select the Add To Project option from Project option in Menu of VC++ environment.
2) Click on the Components and Controls option and open the Registered ActiveX controls folder.
3) Then insert the required GistSDK ActiveX control (control name is staring with the letter 'G').

ActiveX Controls used in our system

GStatic Control

A Static control capable of displaying prompts in Indian languages. It can handle 8-bit ISCII, 7-bit ISCII or PC-ISCII data formats. GStatic's **Alignment** can also be set to left/right/center. We can specify the script, font name and font size according to our use.

GEdit Control

It is an ActiveX control, which is a **text box** capable of receiving and displaying text in Indian languages. GEdit can handle 8-bit ISCII, 7-bit ISCII or PC-ISCII data formats. Developer can specify the script, the font name and font size. These values can be changed at runtime. We can directly **edit** text by **typing** or **cut-copy-paste**, from another control on a Right-click. It also provides features of **Find and Replace**. GEdit is capable of being **Multiline** also. It provides vertical as well as horizontal **scroll bars** optionally.

GEdit can be also set to be **Readonly**, or as a **Password** box (default password char is *). Optionally the foreground and background colors can also be set or defaults loaded. One can also set the appearance and border style. The text can be set at design time and manipulated at runtime. GEdit is returning ISFOC values of the ISCII string it contains. The ISFOC values are determined depending on the Bengali script and font (here we are using *TLB-TTHarshapriya* font) type selected.

GPushButton Control

It is an ActiveX component, which is a push button with caption text in Indian languages. We can specify the script, the font name and font size to use. The caption to be displayed is set at design time using property-sheet. The procedure to add a GPushButton control to our project is the same as that of GEdit.

GMsgBox Control

It is a Message Box capable of displaying Messages in Indian language text. Standard styles of OK-Cancel-Retry-Ignore etc are supported with return values similar to standard Message Box. There is facility to prompt user using a **Phonetic** String. String may also be ISCII 8 bit, 7-bit, PC-ISCII or ISFOC. The control is invisible at runtime (has no User Interface at run-time) until show method is called.

GListBox Control

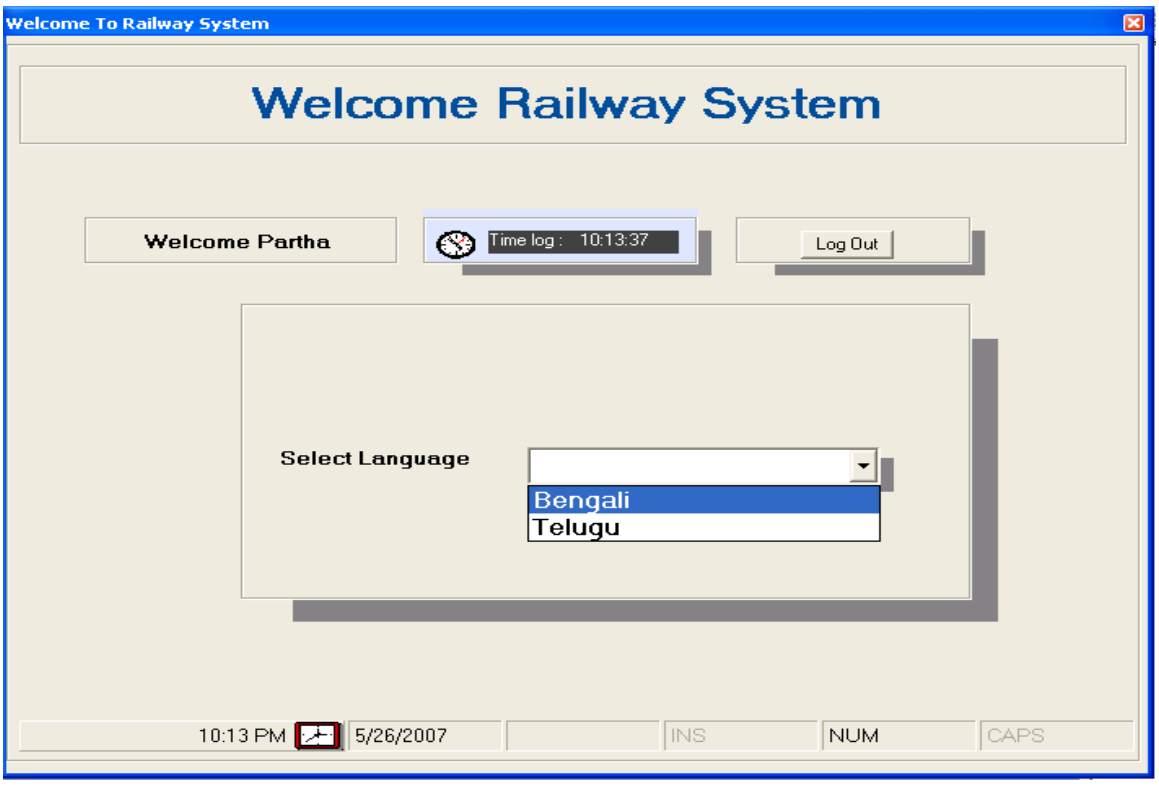
This List-Box control is capable of displaying a **list** of Indian language data items. GListBox is a list box that can show Indian language (ISCII) data. It enables the user to make one or more selections from a list of values added at the design time or added on the fly at runtime. To add or delete items in a GListBox control, use the AddItem or RemoveItem method. Alternatively, you can add items to the list by entering the List in the property-page at design time.

The elements in a GListBox can be ISCII sorted, that is sorted according to order in which the data appears in the ISCII table, and options include **Sort** (ISCII sort) and **multiple select** options. It allows selection by user by mouse and keyboard. The script, font name and font size can be set. The procedure to add a GListBox control to the project is the same as that of GEdit.

4.5 Querying the System

At first Authentication screen will come then user enter his user id and password.

Then user selects the language.



After selection of Bengali the Query screen will come.

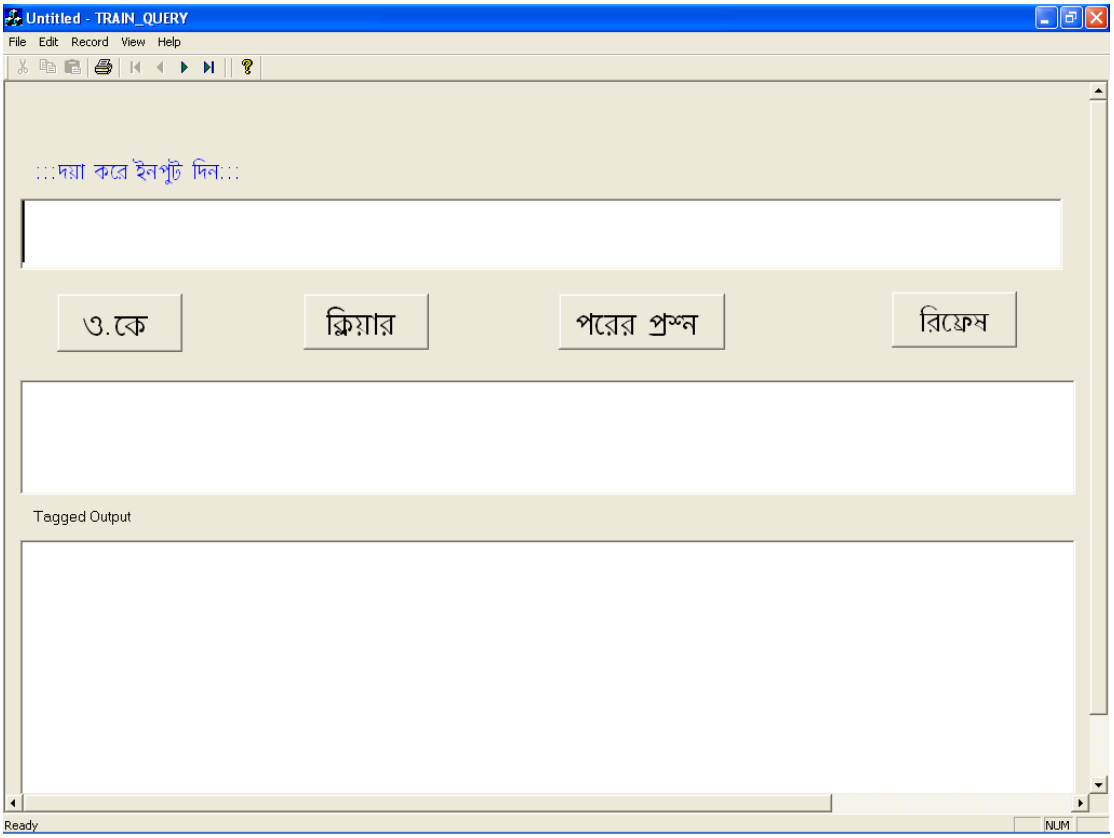
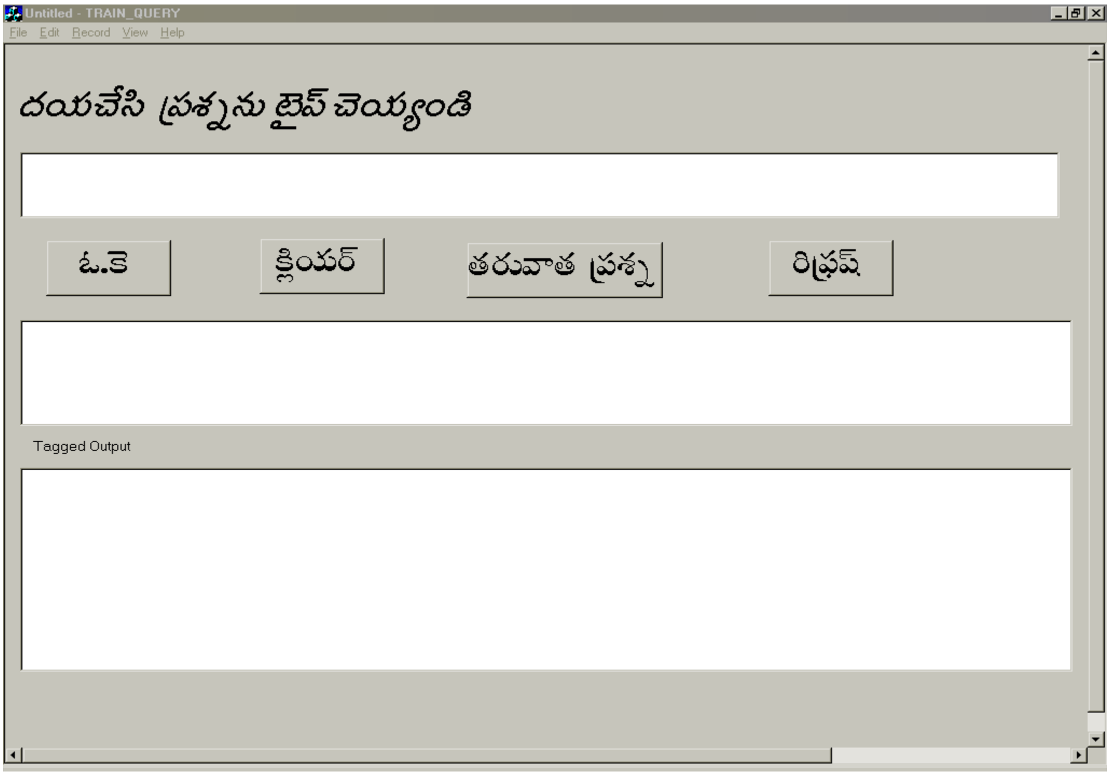


Figure 4.1 User Interface 1

If you will select language Telugu then Query screen will come



In order to query the system, user must need to type the natural language (NL) question in the GEdit control shown in figure 4.1.Here we are using 3 GEdit controls, one (small one) is used to enter the NL question, while the medium one used to show system generated NL answer and the larger one is used to show the morphosemantic tagged output of the input query.

For the example dialogue shown in the previous chapter (figure 3.5), the interaction between user and system is shown as follows. Initially we type the NL question, ˆý‰š ˜ôêÆ˂¢ê¢¨ òŸ¢ êš“ý˂¢›ô¦ p Š Ublê“àý ò›ô“ëpôò√pp (kakhan Darjeeling Mail New Jalpaiguri pouchabe [When the Darjeeling Mail goes to New Jalpaiguri])?

Then press the [illegible] (OK) button. On pressing this OK button, at first system analyzes the input question to get the semantic information from the NL query and continue the process of remaining procedures as shown in later sections. Figure 4.2 shows the input NL question, by typing it in the GEdit control.

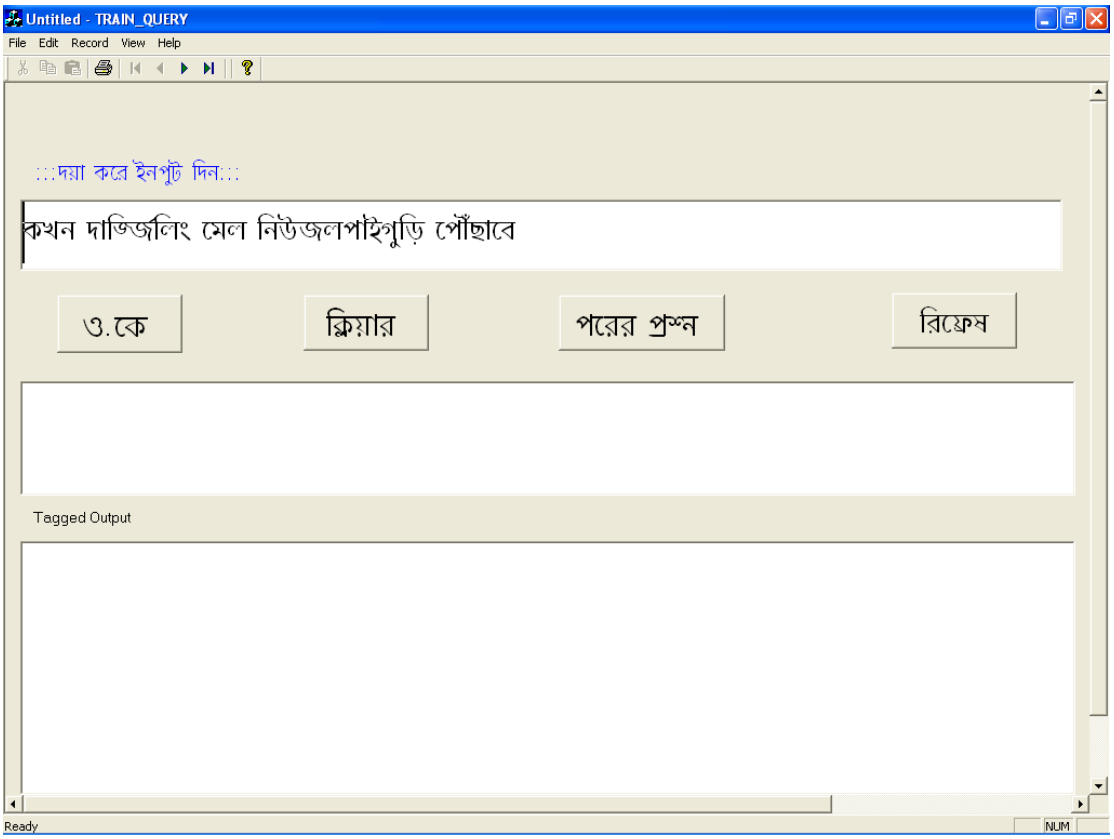


Figure 4.2 User interface 2.

4.5.1 Shallow Parser

Initially, the input query statement in user language is taken from the GEdit Box in the form of a string. The input query statement is parsed into a sequence of words according to the spaces. This is done with the help of the **split_query( )** function. The **split_query( )** function splits the input string into a sequence of words according to the specific character (here it is space) occurring in between two words in the input string.

Once the input query statement is split into a sequence of words, each word is searched into domain and linguistic model tables like Train Name, Station Name, Reservation Class and Keywords etc. as discussed in the Section 4.2. The word is searched into the tables to determine its type and semantic information that it contains. It may happen that a word may not be found in any of the above tables. In that case, the next word is concatenated with the previous one with a space in between them and searched into the tables again. Some words are not found in any table at all. Those words do not contain any semantic information and are discarded.

We have to consider the fact that a word may be inflected in a user query. So we have implemented **Check_Inflection( )** function to find out whether the word is concatenation inflected or not. If a word in any of the above table is a substring of a word present in the input query, then the extra part of the word is searched into the Bengali Inflection table discussed in Section 4.2, to check whether the extra part is an inflection or not. If the inflected word is spell changed one, then we call the **Check_Suffix( )** function to get the root word from the inflected one. After searching each word (or concatenation of words) in the tables of domain and linguistic model, the larger GEdit control is filled with the morphosemantically tagged output of input query as shown in figure 4.3.

For the example query shown in figure 4.3, নিউজলপাইগুড়ি (New Jalpaiguri) is tagged under <Station_Name> is Destination station. The word পৌঁছাবে (*pouchabe* [will reach]) is tagged under <Keyword> tag, so we are considering it as a keyword used to decide the topic of the query i.e. query frame.

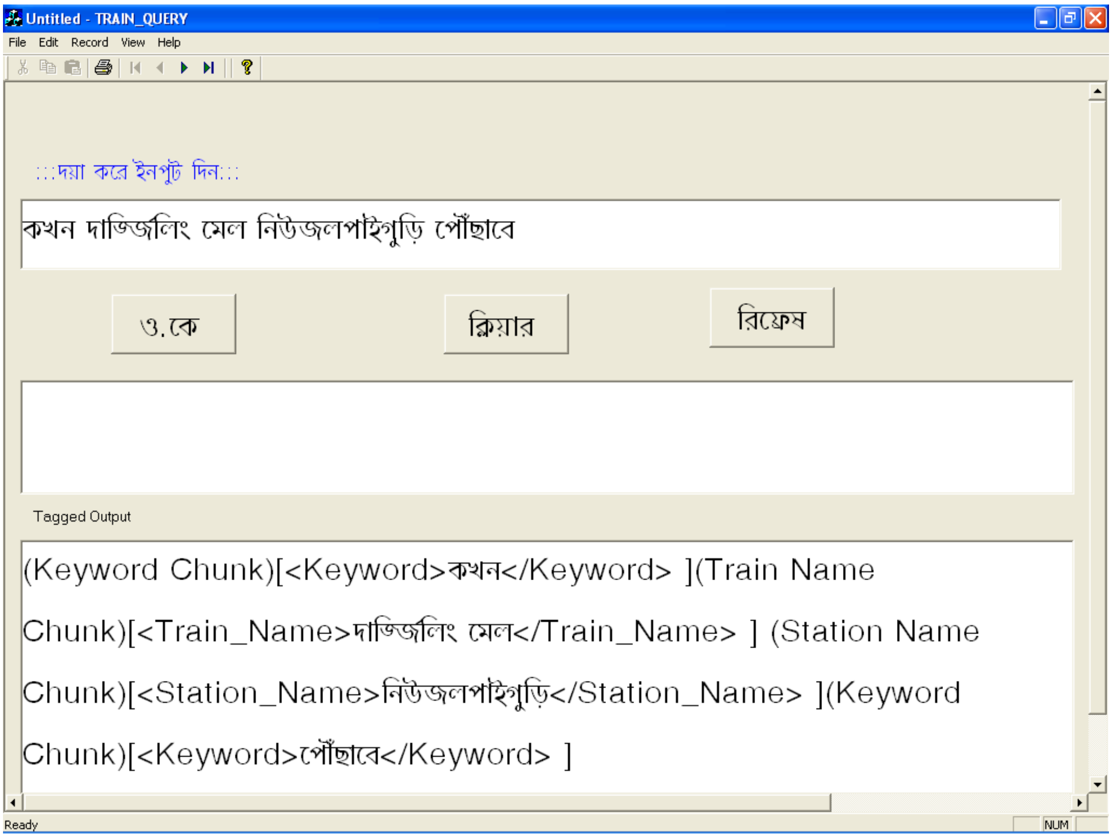


Figure 4.3 User interface 3

By looking at the keyword, we assume that the query is under [Fare] query frame. These semantic frames like source station, destination station, and query frame are updated in the *dialogue history* for future use.

4.5.2 Dialogue Manager

To calculate the fare between two stations, user must need to provide source station and destination station, or distance between those stations, train name, and reservation class name. That means [Fare] query frame expects source station and destination station, or distance between those stations, train name, and reservation class name. But, the user query given in this dialogue consist information about source (Sealdah) and destination station (New Jalpaiguri) only. After *modeling this dialogue*, system intended to ask about the train name in which user needs the fare value. That IDACT (Intended dialogue act) is converted in the form of a sub dialogue generated by the dialogue manger as shown in figure 4.4.

The response, ˆý-ý ìýĉ"ýĉ ê£ßlĉ¢˜ĉ ò—òˆý êš"チýマ¢›ĉ¦チþŠUê"àý ›l¢Ĵýý(koto bhara Sealdah to Newjalpaiguri porjonto[What is fare from Sealdah to New Jalpaiguri]) is generated by the response generator, which is a part of dialogue manager, using linguistic model knowledge and current semantic frames (like source and destination station names).

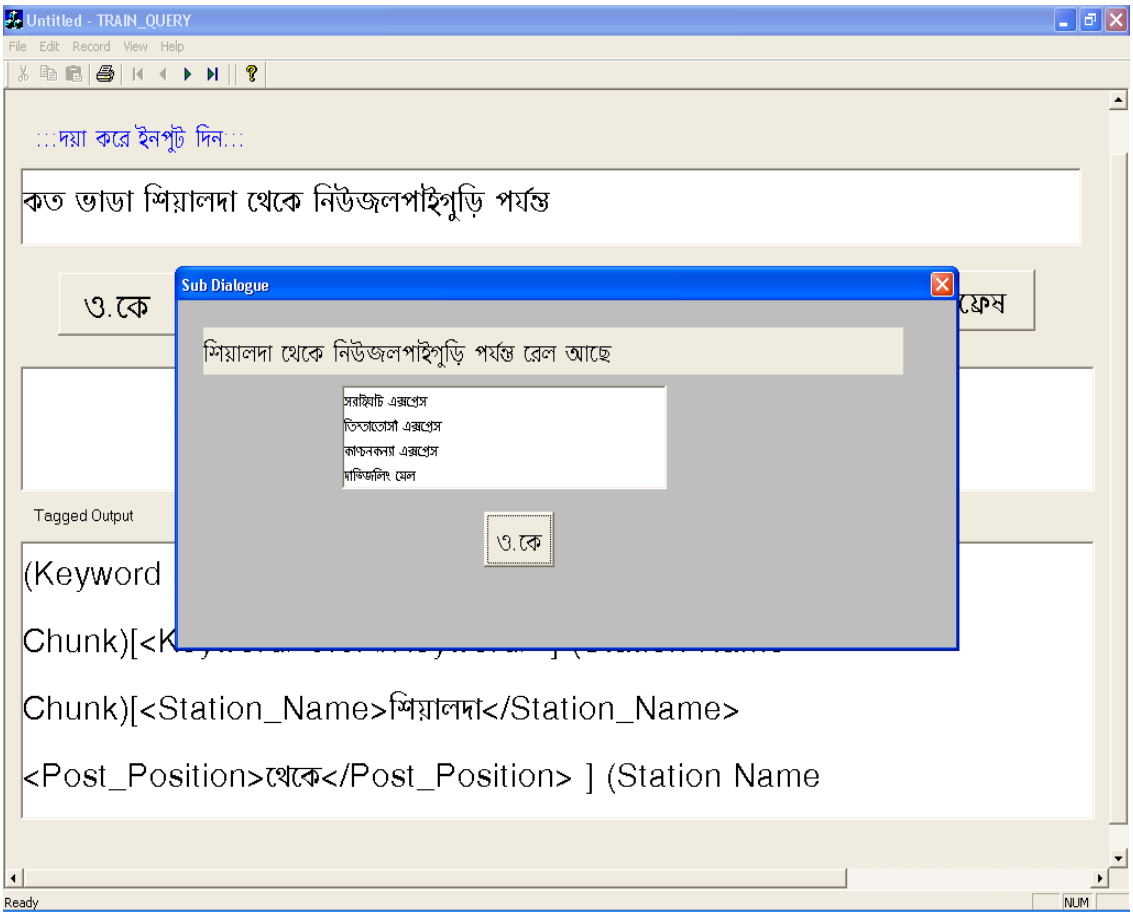


Figure 4.4 User interface 4

SQL Generation

Instead of asking the train name from the user, we are listing out, all the trains running between those stations in natural language Bengali, from which user selects one. In order to list out the trains running from Sealdah to New Jalpaiguri, we call the **Trains_Bet_Stations( )** function. This function generates a set of SQL statement(s) to find out the trains running between those stations (this SQL generation is called by the dialogue manager). At first, we execute the following SQL statement to get the route number under which those stations belong.

SELECT Route_No FROM GetRoute WHERE Source_Station='SEALDAH'AND Destination_Station='NEW JALPAIGURI' ……...…… (1)

Once we got the route number (Here it is 3), we find the name of the route table using the **Route name** table knowledge of domain model. After getting the name of the route table (**ROUTE_NJP_HWH**), we are executing one more SQL statement to get the train numbers.

SELECT Train_No FROM ROUTE_* NJP_HWH *WHERE [SEALDAH] =1 AND [NEW JALPAIGURI] =1 …….. (2)

From these train numbers, we find the name of the trains using *domain model's* **Train Name** table. These train names are listed in the GListBox control as shown in figure 4.3. Here user selects for the ˜ôêÁ¹¢ê¢¨ ò¨Ÿ¢ (Darjeeling Mail) in which he needs fare value. DM updates this Train Name and its number (here it is 2343) information in the dialogue history.

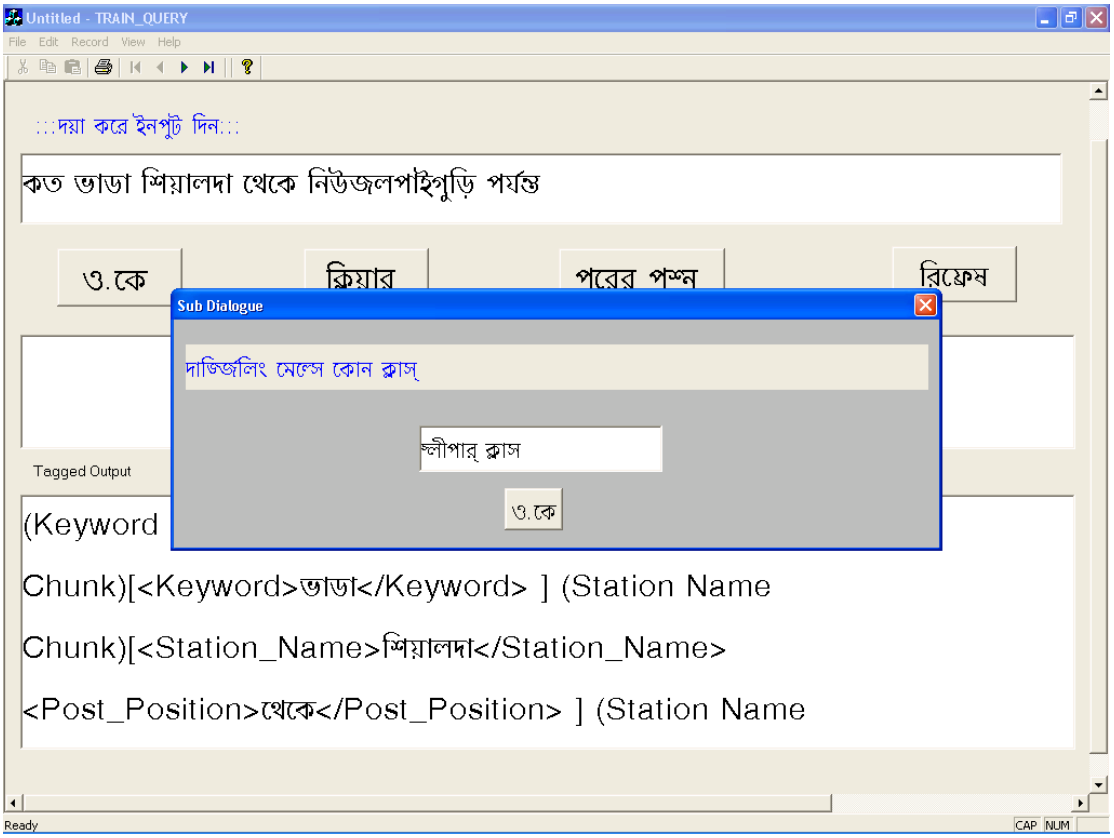


Figure 4.5 User interface 5

Now, system gets the train name information, but still lack in reservation class name to find the fare. So, DM sends one more sub dialogue to get the reservation class name as shown in figure 4.5.The system response, ˜ôêÁ¹¢ê¢¨ ò¨Ÿò¢ßÝþ ò^ýôš ^Ôýô¤ (*Darjeeling mailer kon class* [For which class, you need fare in *Darjeeling* Mail]) is also generated by the response generator. In response to the system's request, user asks for Ĵ¢íòßÝþP ^Ôýô¤ (*sleeper class*), which is analyzed by shallow parser and identified it is Sleeper Class reservation class name. The morphosemantically tagged output of this user response is shown in figure 4.6. After getting this reservation class name from the user, dialogue manager calls the

SQL generation procedure for the [Fare] query frame to generate the necessary SQL statement(s) in finding the fare.

4.5.3 SQL Generation

Before finding the fare between Sealdah and New Jalpaiguri stations in the *Darjeeling* Mail for Sleeper class by executing a sequence of SQL statements, it checks whether reservation class given by the user is present in the train or not. To check this, we are executing the following SQL statement that outputs a single row in which reservation class names as columns.

*SELECT *FROM ReservationClass WHERE Train_No LIKE '%2343%'*………… (3)

The values under the columns are in the Boolean form ('1' or '0'). Since our reservation class name is Sleeper class, we check the value under Sleeper Class (SL) column, it is founded as '1' (because *Darjeeling* Mail has Sleeper Class) and we are continuing the fare finding process along with updating reservation class in the dialogue history. If the user asks about First Class, SQL statement (3) returns '0' (because Darjeeling Mail doesn't have First Class), then the system generates a **cooperative message** like ˜ôêÆ˂¢ê¢¨ òŸò¢ œýôĴ˝ýÞ …ê¤ý ,˝Úý˝ýôß/ôß˘/þ …ê¤ý,–áýí˝ýôß/ôß˘/þ …ê¤,Ĵ¢í›ôß˘/þÞ ˆÔýô¤,ò¤ˆý¨“ýÞ ˆÔýô¤ (*darjeeling mail first ac, twotier ac, threetier ac, sleeper class, second class* [*Darjeeling* Mail has First AC, Two Tier AC, Three Tier AC, Sleeper Class and Second Class only]), instead of finding the fare. In order to generate this cooperative message, we check the column values of the output row obtained by the execution of SQL statement (3). If the column value is '1', that means, reservation class is present in that train. So, we include all the reservation classes in the cooperative message whose value is '1'.

Fortunately, the reservation class given by the user (Sleeper Class) is present in the Darjeeling Mail. So we find the fare between the stations in Sleeper class. To find the fare between those stations, we need to identify the type of the train (Rajdhani, Mail/Express, Shatabdi etc.), because based on the train type, fare structure will differ. To identify the train type, the SQL query executed is

SELECT Type_Of_Train FROM Train_Index WHERE Train_No=2343……… (4)

The result of above query says *Darjeeling* Mail is an Mail train. In Express trains, we find the fare based on the distance between the stations, unlike the Rajdhani Express, whose fare is obtained just based on the source and destination stations.

To calculate the distance between the stations, we execute the following SQL statements on the schedule table of the train (**Schedule2343)** (2343 is the train number of *Darjeeling* Mail running from Sealdah to New Jalpaiguri).

SELECT Distance FROM Schedule2343 WHERE Station_Name='SEALDAH' **……..** (5)

SELECT Distance FROM Schedule2343 WHERE Station_Name='NEW JALPAIGURI'**.....** (6)

After getting the distance, we find the fare based on the distance slab, in which fare will be same for all the distances within the *Dist_From* and *Dist_To* slab of the fare table. The SQL statement used to find the fare is

SELECT FARE_SL FROM Fare WHERE Dist_From <= 569 AND Dist_To>=569**……** (7)

The output of above SQL statement is 210, which is updated in dialogue history for future use and send it to the Answer generator via DM to generate NL answer.

4.5.4 Answer Generator

We are using template based answer generator to generate natural language answer, in which slots are filled by the retrieved result and the semantic frames updated in the dialogue history. Apart from the slots, some form of linguistic knowledge is used as fixed parts.

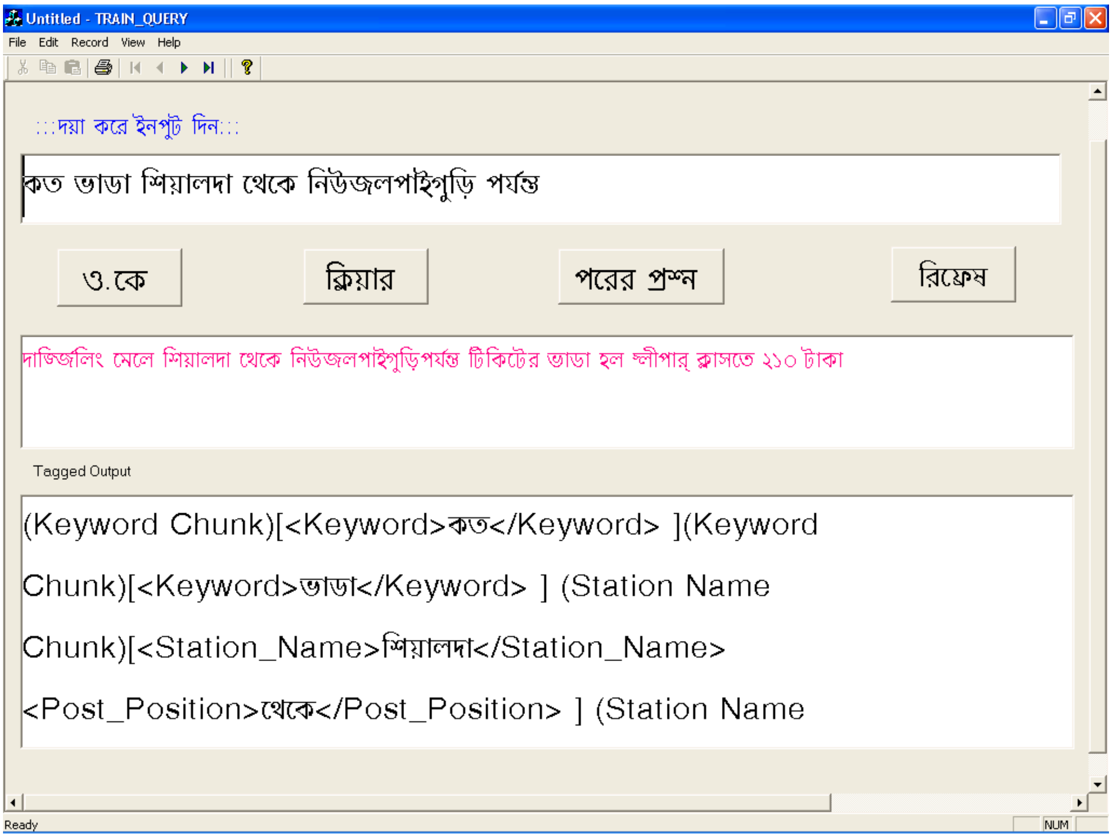


Figure 4.6 User interface 6.

The answer template used for this [Fare] query frame

The slots like [Train_Name], [Source_Station], [Destination_Station], and [Reservation_Class] are filled using the updated information present in the dialogue history. The [Fare Value] is filled by using the retrieved result.

The NL answer for the above query ˜ôêÆ२¢êç¨ ò Ÿõç êÆ ƒß/ôç˜ô ò—ò ˆý êš "Ŧý२¢›ô¦ Ŧþ ঁ Üê"àý›/ô Ÿý ê 'Ŧýê ˆýò Ð 'Ŧýƒß'þ ïýô "ýô ¦þ¢ Îç›ôƒß'þ ˆÔýô¤ò–ý 210 'Ŧýô ˆýô (Darjeeling maile *sealdah theke new jalpaiguri porjonto tikiter bhara holo sleeper classe 210 taka* [Fare from Sealdah to New Jalpaiguri by Darjeeling Mail in Sleeper Class is Rs.210]) as shown in figure 4.6.

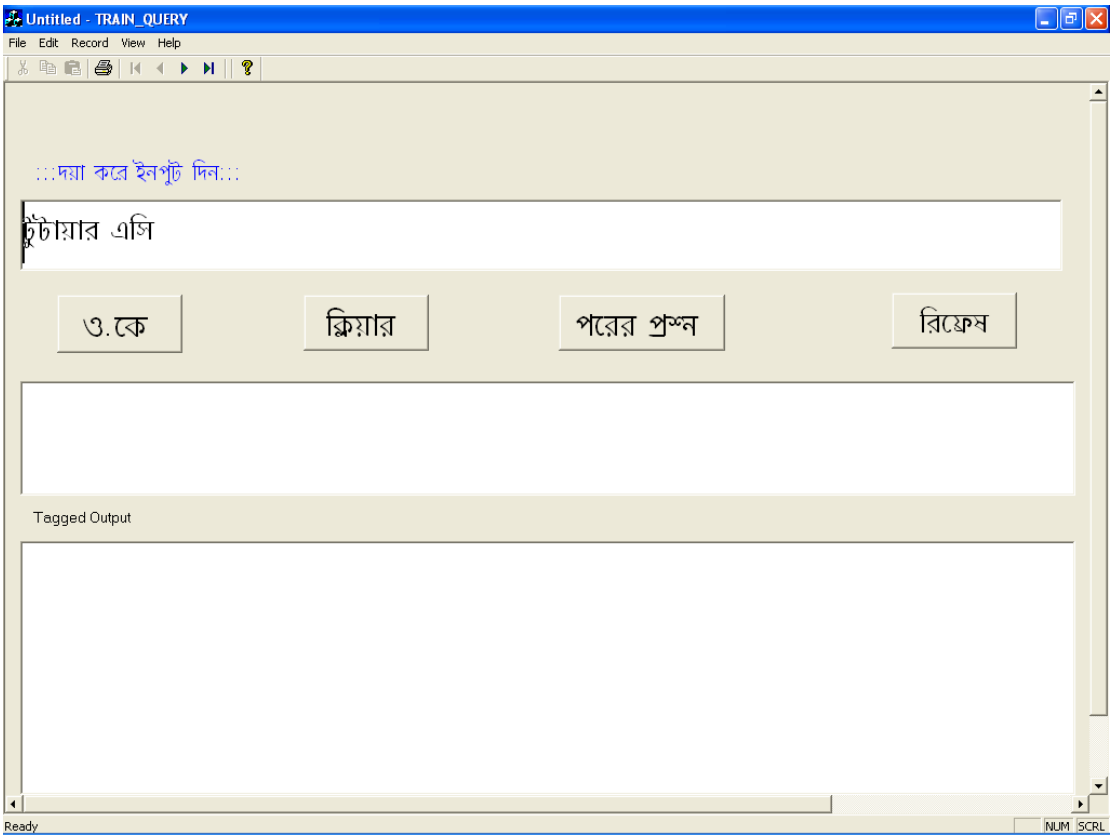


Figure 4.8 User interface 8

In order to get the answer for the above question, user just needs to press the ‡.ô^ý (OK) GPushButton as in the previous case. Initially query is analyzed by shallow parser and produces the tagged output shown in figure 4.9.From the morphosemantically tagged output, the semantic frame we identified is a reservation class name (Three Tier AC).

4.5.5 Dialogue History

To process any query, user must need to provide query frame information in terms of keywords. But, here by looking at the tagged output, we are unable to find any keywords used to identify the query frame, because the question is an elliptical expression. So, dialogue manager looks at the dialogue history (DH) for the query frame. In the DH, the query frame is stored as [Fare], so system assumes the present query also under [Fare] query frame.

For the [Fare] query frame, user must need to provide Train name, Source station and Destination station/distance between stations and reservation class name, but in the present query user has given reservation class information only. In this case also, system gets the Train name, Distance value from the DH itself (this information

is updated in the DH from the previous query). To find the fare between the stations, SQL generation procedure needs either source and destination stations or the distance between those stations. We are using the distance value stored in DH that reduces the number of SQL statements execution (No need to execute SQL statements (5), (6) to calculate distance.

Now, we got the Train name and distance value from the DH, and the reservation class name from the present query. These values are sufficient to find the fare. Before finding the fare, we are checking the reservation class name presence by executing the SQL statement (4) as shown earlier.

The value we got under Two Tier AC column is '1',so we are continuing the fare finding process instead of generating cooperative message as shown earlier. The SQL statement executed to calculate the fare between Sealdah and New Jalpaiguri in Darjeeling Mail for Two Tier AC is

SELECT FARE_AC2 FROM Fare WHERE Dist_From <= 569 AND Dist_To>=569**.....** (8)

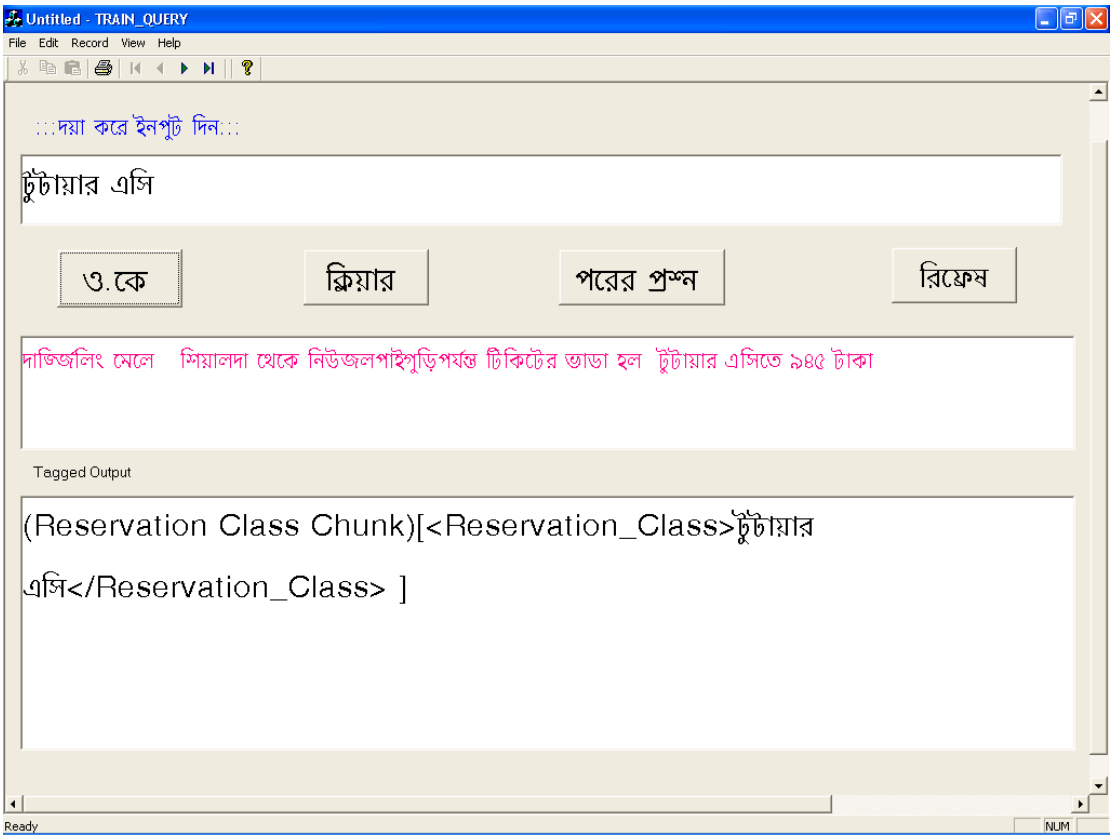


Figure 4.9 User interface 9.

The output of above SQL statement is 945, which is updated in dialogue history for future use and send it to the Answer generator via DM to generate NL

answer like [illegible] ...[illegible] 945 [illegible] (*Darjeeling maile sealdah theke new jalpaiguri porjonto tikiter bhara two tier ac te 945 taka* [Fare from Sealdah to New Jalpaiguri by *Darjeeling Mail* for Two Tier AC is Rs.945]) using the answer template of [Fare] query frame, as shown in figure 4.9. To fill the slots of [Fare] answer template, it gets the source and destination names from the DH and the retrieved result from the database.

The last question of the user dialogue (U_5) [illegible] (Kokhan charbe [When it will starts]) is typed in the GEdit control by pressing the [illegible] (porer prosno [Next Question]) GPushButton. To process the query, press the [illegible] (OK) GPushButton as shown in figure 4.10.

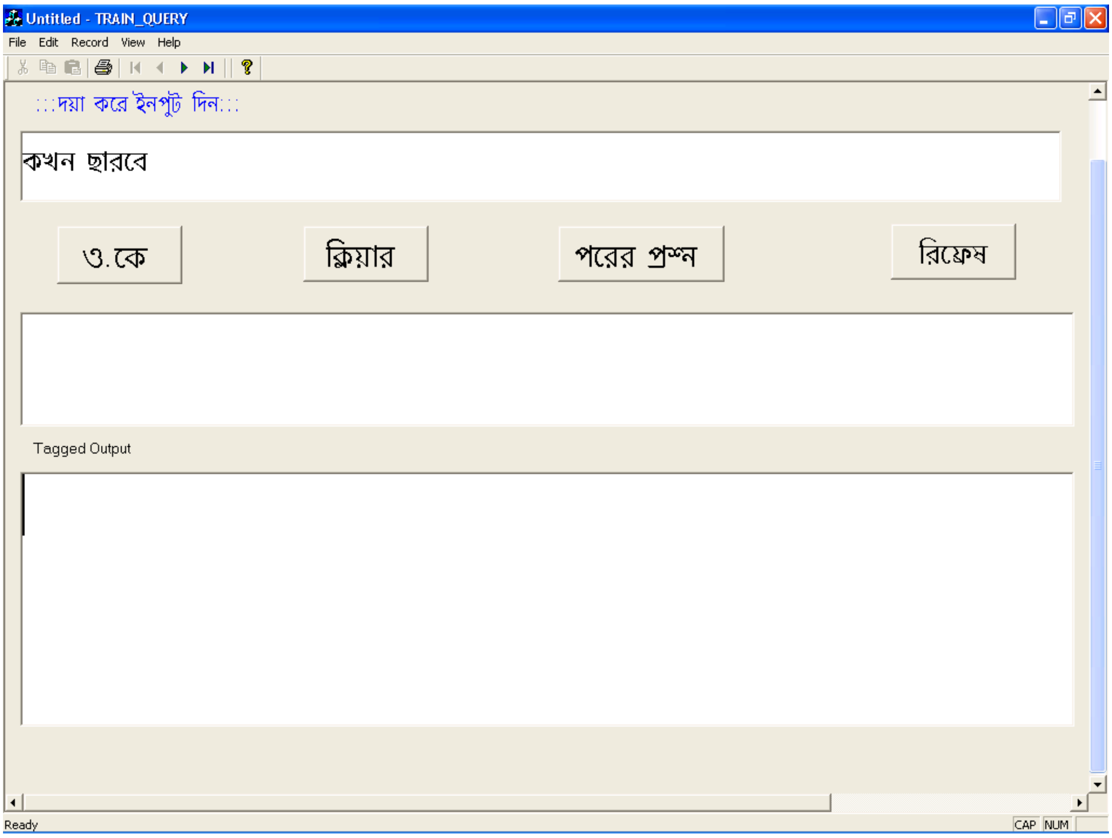


Figure 4.10 User interface 10.

Initially query is analyzed by the shallow parser, which produces the semantically tagged output as shown in figure 4.11.From the morphosemantically

tagged output, we got two keyword chunks, one is ˆý‰š (*kokhan* [When]) and the another is ëþôfš/þò/þ (*charbe* [will start]). These keywords are used to identify the query frame. The word ˆý‰š (*kokhan* [When]) is related time, where as the word ëþôfš/þò/þ (*charbe* [will start]) related to departure. Both together identified the query under [Dep_Time] (Departure Time) query frame.

Once the query frame is identified, we are checking whether the required information for the [Dep_Time] query frame is present in the user query or not. [Dep_Time] query frame expects train name or number and source station from the user query. In this case there is no such information, so dialogue manager looks at the DH for the required information. If DH contains the required information it uses that, otherwise it generates a sub dialogue with the user asking the required information. Here DH contains both train number (2343) and source station (Sealdah) information; use this information to generate SQL statement(s). To find the departure time of a train SQL generation procedure uses Schedule (**Schedule2343)** table.

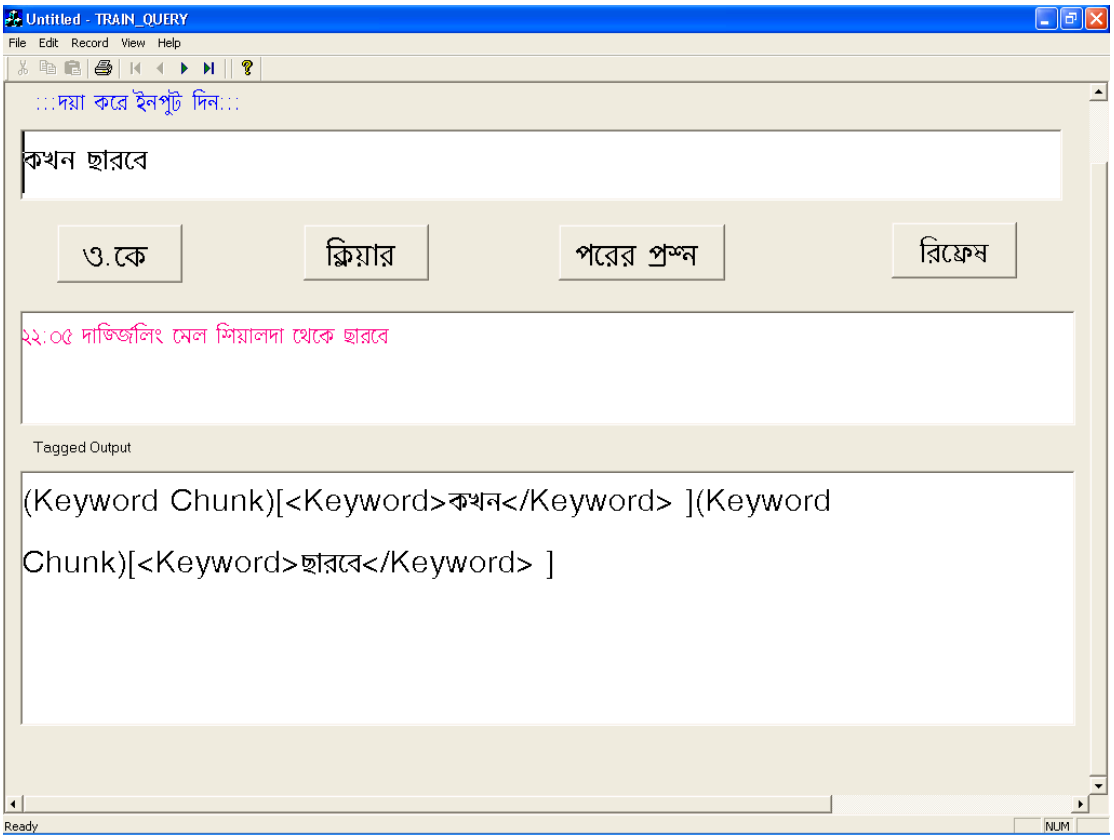


Figure 4.11 User interface 11

The SQL statement generated to get the departure time of Darjeeling Mail from Sealdah station is

SELECT Departure_Time FROM Schedule2343 WHERE Station_Name='SEALDAH'…… (9)

Output generated on executing the above SQL statement is 22:05, which is forward to Answer generator to generate NL answer like *22:05 ~êêÁ¢ê¨ è¨Ÿ¢ êÆßíê¢~ê è—è^ý ëþêß¹þè¹þ* (22:05 *darjeeling mail sealdah theke charbe* [At 18:00 hrs, D*arjeeling Mail* starts from Sealdah]) as shown in figure 4.11, using [Dep_Time] answer template discussed in section 3.8.

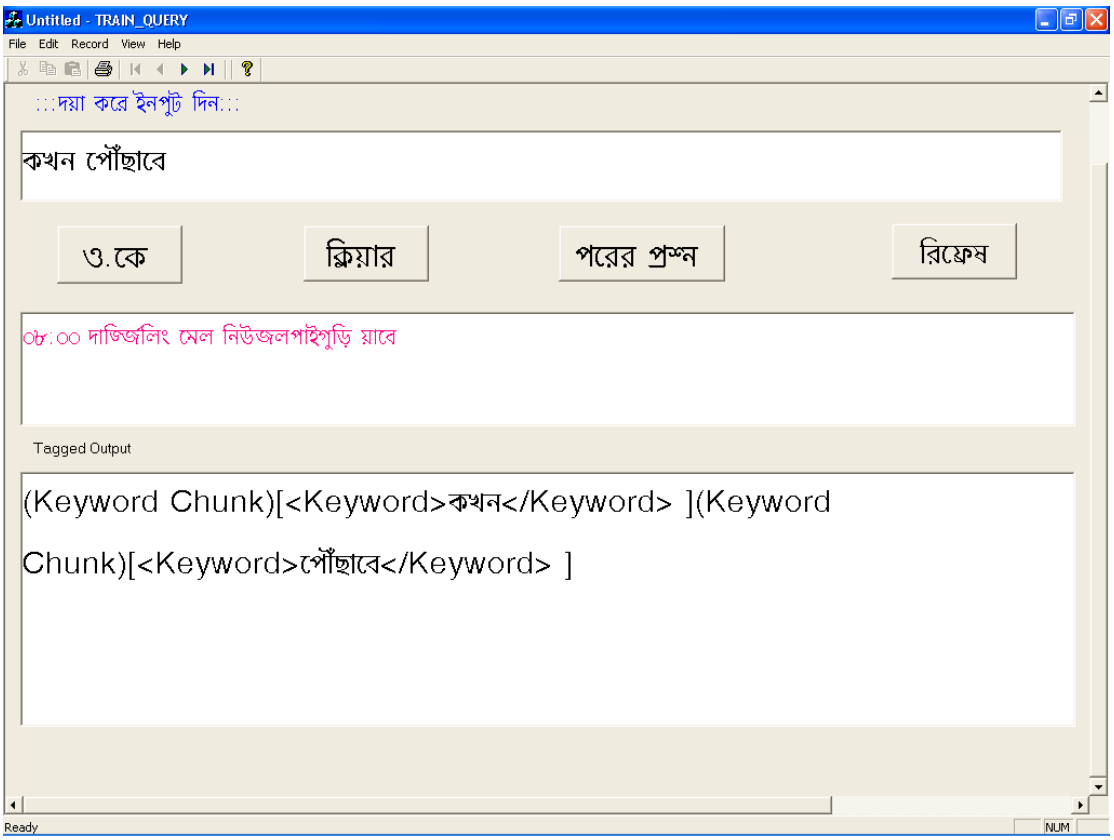


Figure 4.12 User interface 12

The GPushButton *ê^Ôýßíêß¹þ* (Clear) is used to clear the contents of the GEdit controls, where as the GPushButton *êß¹þè×œãýŸ* (Refresh) is used to

refresh the dialogue history. On clicking this button, all the dialogue history contents are deleted, in which user can proceed to another dialogue.

CHAPTER V

EVALUATION AND CONCLUSION

5.1 Evaluation

To evaluate our system, we had taken queries from our Bengali-speaking friends. They have also been told about the constraints on the nature of queries in the system. We had shown a list of example queries accepted by the system. We evaluated our system, with and without the dialogue management.

Evaluation of system without dialogue management:

The system without dialogue management means, one question, and one answer, there is no such an interaction between user and system.

Here we are considering two measures for evaluating the system without dialogue management. Those are **Precision** and **Recall.**

Precision = (Number of correct answers generated by the system / Number of answers generated by the system) * 100.

Recall = (Number of correct answers generated by the system / Number of natural language questions given to the system) * 100.

❖ **Bengali Query System:**

For this case, our system is evaluated by giving a set of 70 questions. Out of 70 questions, system has generated answers for the 64 questions. Out of 64 answers, 56 are identified as correct answers, for the remaining 8 questions, system unable to generate the exact answer.

Number of answers generated = 64.

Number of correct answers generated = 56.

Precision = (56/64)*100 = 87.50%.

Number of natural language questions given to the system = 70.

Recall = (56/70)*100 = 80.00%.

❖ **Telugu Query System:**

For Telugu System is evaluated by giving a set of 132 questions. Out of 132 questions, system has generated answers for the 127 questions. Out of 127 answers, 124 are identified as correct answers, for the remaining 3 questions, system unable to generate the exact answer.

Number of answers generated = 127.

Number of correct answers generated = 124.

Precision = (124/127)*100 = 97.63%.

Number of natural language questions given to the system = 132.

Recall = (124/132)*100 = 93.93%.

❖ **For Bengali Query System:**

This low recall rate is due to coverage of domain is not extensive enough i.e. the queries does not belong to any of the query frame. Sometimes, the information given by the user in the query was inadequate and the system was not able to identify the missing information because of the incorrect choice of the query frame. One more cause is that the system is generating a cooperative message to the user (in this case system could not generate the exact answer).My system not implements negative queries. If information given by the user is spelling error then the result will not be identify.

➢ **Evaluation of system with dialogue management:**

The system with dialogue management is evaluated by measuring the system's ability to help the users in achieving their goals, the system robustness in detecting and recovering from errors of understanding, and the overall quality of the system's interaction with users.

In order to measure the system's ability, we considered two metrics: **Dialogue Success rate** and **Precision**. The QA system was evaluated by giving 32 sets of dialogue consisting 62 natural language queries in total. Each set of dialogue consists of around 1 to 4 natural language queries.

Dialogue success rate for each set= (Number of Answers or Responses generated by the system / Number of turns issued by the user)

Dialogue success rate = ((∑ Dialogue success rate for each set) / Number of sets of dialogues)*100.

Precision= (Number of correct answers given by the system/ (Number of answers given by the system)*100.

The number of turns issued by the user in a dialogue is the total of the number of questions issued to the system and the number of responses provided by the user to the system.

❖ **Bengali Query System:**

The total dialogue success rate for the 24 sets was obtained as 17.5. The Dialogue success rate for the system is calculated as

Dialogue success rate= (17.5/24)*100= 72.91%.

Out of 58 questions, system generated answers for 49 questions of which 41 were correct answers. So, the precision of the system is calculated as

Precision= (41/49)*100= 83.67%.

❖ **Telugu Query System:**

The total dialogue success rate for the 32 sets was obtained as 28.5. The Dialogue success rate for the system is calculated as

Dialogue success rate= (28.5/32)*100= 89.06%.

Out of 62 questions, system generated answers for 57 questions of which 55 were correct answers. So, the precision of the system is calculated as

Precision= (55/57)*100= 96.49%.

For Bengali Query System:

This low dialogue success rate is due to the fact that the system coverage of the domain is not extensive enough, i.e., query frames for some natural language queries were not correctly identified and some question are out of database coverage, because, we are considering a few database tables. The information given by the user during the dialogue was inappropriate to context, and the system was not able to generate proper SQL statement(s) to generate the answer Sometimes the system is unable to obtain chunks correctly from the input query even if it had identified the right query frame, thereby generating wrong answers. Misinterpretation of dialogue history is also another problem.

5.2 Conclusion and Future Scope of the work

The multilingual restricted domain question answering system separates dialogue control from the application logic, provides a portable dialogue manger and a user-friendly interface. The Dialogue manager is language independent. Separate modules

have been developed to generate the necessary SQL statement(s) to retrieve the result, which is used in generating the natural language answer by the answer generator. Some preliminary evaluation of the system has been carried out. We are developing more robust shallow parser and the modules for the remaining query frames with dialogue management. Similar modules have to be developed for other Indian languages. The scope of the system should be extended to handle information retrieval from documents.

The system needs to be upgraded so that a user can query for railway information over phone. The speech input can be converted to textual query. This textual query can be input of our system and the textual output can be converted to speech again to answer the user. The multilingual system architecture can be converted into cross-lingual one by allowing the user to specify the target language in which all output will be generated and by enabling the Dialogue Manager to retrieve data and answer / response templates in the target language from the Railway Information database and the Answer / Response template base.

The system needs to be upgraded so that a user can query for railway information over phone. The speech input can be converted to textual query. This textual query can be input of our system and the textual output can be converted to speech again to answer the user. It is not realistic to assume that text-based dialogue systems can be converted into speech-based dialogue systems trivially, e.g. by adding speech recognition and synthesizer components. The focus of these systems is different, and some of the research questions, especially those dealing with the nuances of written text, are not particularly relevant in speech systems. Nevertheless, speech systems can utilize knowledge from text-based dialogue systems.

REFERENCES

[1] Abney.S Parsing by chunks. In *Principle-Based Parsing*, pages 257-278.Kluwer Academic Publishers, Dordrecht, 1991.

[2] Adelheit Stein, Jon Atle Gulla, and Ulrich Thiel. User-tailored planning of mixed initiative information-seeking dialogues. *User Modeling and User-Adapted Interaction*, (9):133-166.1999.

[3] Androutsopoulos. I, Ritchie G. D, and Thanisch P.Natural Language Interfaces to Databases – An Introduction. *Natural Language Engineering*, Vol 1, Part1, 29–81. 1995.

[4] Bennacef. S, Devillers.L.Roset. S and Lamel.L. Dialog in RAILTEL telephone based system. In *Proceedings of International Conference on Spoken Language Processing, ICSIP'96*, Volume 1,550-553, Philadelphia, USA. 1996.

[5] Bilange. E. A task independent oral dialogue model. In *Proceedings of the Fifth Conference of the European Chapter of the Association for Computational Linguistics, EACL'91*, 83-88, Berlin, Germany.1991.

[6] Daniel Jurafsky and James.H.Martin, Speech and Language Processing. Third Edition, 2004.

[7] Diekema A.R, Yilmazel Ozgur, and Liddy E.D. Evaluation of Restricted Domain Question-Answering Systems. In *Proceedings of the ACL2004 Workshop on Question Answering in Restricted Domain*, 2-7. 2004.

[8] Fleischman. M, Hovy. E and Echihabi A. Offline strategies for online question answering: Answering questions before they are asked. *ACL*.2003.

[9] Flycht-Eriksson Annika. A survey of knowledge sources in dialogue systems. In *Proceedings of the IJCAI-99 Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, pages 41-48, Murray Hill, New Jersey. International Joint Conference on Artificial Intelligence.1999.

[10] Flycht-Eriksson Annika and Jonsson Arne. Dialogue and Domain Knowledge Management in Dialogue Systems. In *Proceedings of 1st SIGDIAL workshop at ACL2000*. 2000.

[11] Green W, Chomsky C, and Laugherty K. BASEBALL: An automatic question answerer. In *Proceedings of the Western Joint Computer Conference*, 219-224. 1961.

[12] Grice H.P.Logic and conversation.In Cole, P. and Morgan, J.L. (Eds), Speech acts: syntax and semantics Volume 3, PP 41-58.Academic Press, New York.1975.

[13] Hendrix, G. Natural Language Interface (panel).*Computational Linguistic*s, 8(2):55-61, April-June, 1982.

[14] Hoojung Chung, Young-In Song, Kyoung-Soo Han, Do-Sang Yoon, Joo-Young Lee, Hae-Chang Rim and Soo-Hong Kim. A Practical QA System in Restricted Domains. *In Proceedings of the ACL 2004 Workshop on Question Answering in Restricted Domain, 39-45.* 2004.

[15] Hovy E, Gerber L, Hermjakob U, Junk M, and Lin C.Y. Question answering in webclopedia. *TREC*.2000.

[16] Hovy E, U. Hermjakob, and Chin-Yew Lin. The Use of External Knowledge of Factoid QA. Proceedings of *TREC-10*, Gaithersburg, MD, U.S.A.2001.

[17] Hovy E, Hermjakob U, Lin C.Y, and Ravichnadran D. Using knowledge to facilitate factoid answer pinpointing .*COLING*.2002.

[18] James Allen,Lenhart Schubert,George Ferguson,Peter Heeman,Chung He Hwang,Tsuneaki Kato,Mark Light,Natheniel Martinm,Bradford Miller,Massimo Poesio, and David Traum. The TRAINS project: a case study in building a conversational planning agent. *Journal of Experimental and Theoritical Artificial Intelligence*, 7:7-48, 1995.

[19] Kristiina Jokinen, Antti Kerminen, Mauri Kaipainen, Tommi Jauhiainen, Graham Wilcock, Markku Turunen Jaakko Hakulinen, Jukka Kuusisto and Krista Lagus. Adaptive Dialogue Systems - Interaction with Interact. In *Proceedings of the 3rd SIGdial Workshop on Discourse and Dialogue, pages 64--73, Philadelphia.*2002.

[20] Lamel. L, Rosset. S, Gauvain J.L, Bennacef. S, Garnier-Rizet .M, and Prouts.B. The LIMSI ARISE system. *Speech Communication*, 31(4):339–354.2000.

[21] Lars Ahrenberg, Arne Jonsson, and Nils Dahlback. Discourse representation and discourse management for natural language interfaces. In *Proceedings of the Second Nordic Conference on Text Comprehension in Man and Machine*, Taby, Sweden. 1990.

[22] Li.X and Roth.D. Exploring evidence for shallow parsing. In *Proceedings of the Fifth ComputationalNatural Language learning workshop (CoNLL).*2001.

[23] Magnini. B, Romagnoli. S, Vallin. A, Herrera. J, Penas. A, Peiado. V, Verdejo. F and M. de Rijke. The multiple language question answering track at clef 2003. *CLEF*.2003.

[24] Ravichandran. D, Ittycheriah. A and Roukos. S. Automatic derivation of surface text patterns for a maximum entropy based question answering system. *HLT-NAACL*.2003.

[25] Srihari, R., W. Li, C. Niu and T. Cornell. InfoXtract: A Customizable Intermediate Level Information Extraction Engine. *HLTNAACL03 Workshop on the Software Engineering and Architecture of Language Technology Systems (SEALTS)*. Edmonton, Canada.2003.

[26] Stephanie Seneff, David Goddeau, Christine Pao, and Joseph Polifroni. Multimodal discourse modeling in a multi-user multi-domain environment. In *Proceedings of International Conference on Spoken Language Processing, ICSLP'98*, volume 3, pages 931-934, Sydney, Australia, December, 1998.

[27] Sutton. S, Novick D. G, Cole R. A, and Fanty. M., Building 10,000 spoken-dialogue Systems. In *Proceedings 4th International Conference on Spoken Language Processing (ICSLP-96)*, 1996.

[28] Traum R.D.1996.Conversational agency: The TRAINS-93 dialogue manager. *In Proceeding of Twente workshop on Language Technology, TWLT-II*

[29] Ulf Krum, Hartwig Holzapfel and Alex Waibel. Clarification Questions to Improve Dialogue Flow and Speech Recognition in Spoken DialogueSystems.*INTERSPEECH 2005*, pages 3417-3420.2005.

[30] Voorhees E.M. Overview of the TREC 2003 question answering track. In *Proceedings of the 12th Text REtrieval Conference*.2004.

[31] Walker M. A., Litman D. J, Kamm C. A and Abella A. Evaluating Spoken Dialogue Agents with PARADISE: Two Case Studies. *Computer Speech and Language*, 12(3):317–347.1998.

[32] Wolfgang Wahlster, editor. Verbmobil: Foundation of Speech-to Speech Translation, *Springer*, 2000.

[33] Woods W.A, Kaplan R.M, and Webber B.N. The Lunar Sciences Natural Language Information System: Final Report. BBN Report 2378, *Bolt Beranek and Newman Inc.*, Cambridge, Massachusetts.1972.

[34] Xu J, Licuanan A and Weischedel R. TREC 2003 qa at bbn: Answering definitional questions. *TREC.* 2003.

[35] Yang .H, Chua T.S, Wang. S and Koh C.K. Structured use of external knowledge for event based open domain question answering. *SIGIR*. 2003.

PUBLICATIONS, PAPER PRESENTATIONS AND PARTICIPATIONS OUT OF THE PRESENT WORK

➢ Publications

Multilingual Restricted Domain QA System with Dialogue Management Srinivasa Rao Godavarthy, Partha Pakray and Sivaji Bandyopadhyay. Published in the Proceedings of Twentieth IJCAI 2007 workshop on Cross Lingual Information Access-CLIA07, International Institute of Information Technology,Hyderabad;p.p.20-27.January 6-12,2007.

➢ Participations

- Fifth International Conference on Natural Language Processing (ICON-2007) held at IIIT,Hyderabad,from 4-6 January 2007.
- Third International Workshop on Knowledge and Reasoning in Answering Questions(KRAQ-07), IJCAI-07, held at IIIT, Hyderabad, from 6th January 2007.
- Workshop on Cross Language Information Access (CLIA 2007) – Addressing the Information Need of Multilingual Societies, IJCAI-07, held at IIIT, Hyderabad, from 6th January 2007.
- Workshop on Shallow Parsing for South Asian Languages (SPSAL-2007), IJCAI-07, held at IIIT, Hyderabad, from 8th January 2007.

- Workshop on Modeling and Representation in Computational Sematics (MRCS-2007), IJCAI-07, held at IIIT, Hyderabad, from 7th January 2007.
- Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07) held at IIIT,Hyderabad,from 6-12 January 2007.