# A FULLY AUTOMATIC QUESTION ANSWERING SYSTEM TO IMPROVE E-LEARNING

A DISSERTATION

*Submitted in partial fulfillment of the requirements for the award of the degree of*

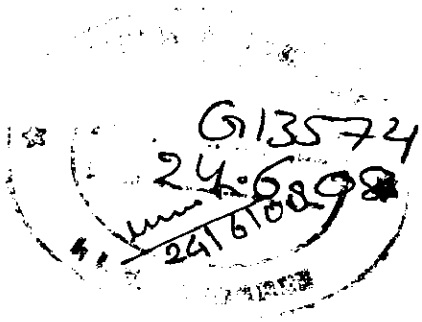
MASTER OF TECHNOLOGY

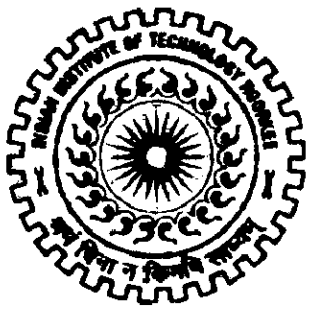
in

INFORMATION TECHNOLOGY

By

**ASHISH GUPTA**



DEPARTMENT OF ELECTRONICS & COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE -247 667 (INDIA)

JUNE, 2007

# CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in the dissertation entitled "**A FULLY AUTOMATIC QUESTION ANSWERING SYSTEM TO IMPROVE E-LEARNING**" towards the partial fulfillment of the requirement for the award of the degree of **Master of Technology** in **Information Technology** submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee (India) is an authentic record of my own work carried out during the period from July 2006 to June 2007, under the guidance of **Dr. Ankush Mittal, Associate Professor, Department of Electronics and Computer Engineering, IIT Roorkee.**

I have not submitted the matter embodied in this dissertation for the award of any other degree or diploma.

Date: 08-06-2007

Place: Roorkee

Ashish Gupta

**(ASHISH GUPTA)**

# CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 08-06-2007

Place: Roorkee

Mittal

**(Dr. Ankush Mittal)**

Associate Professor

Department of Electronics and Computer Engineering

IIT Roorkee – 247 667

# ACKNOWLEDGEMENTS

I would like to extend my heartfelt gratitude to my guide **Dr. Ankush Mittal**, Associate Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, for his able guidance, regular source of encouragement and assistance throughout this dissertation work. It is his vision and insight that inspired me to carry out my dissertation in the upcoming field of Natural Language Processing. I would state that the dissertation work would not have been in the present shape without his umpteen guidance and I consider myself fortunate to have done my dissertation under him.

I also extend my sincere thanks to **Dr. D. K. Mehra**, Professor and Head of the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee for providing facilities for the work.

I also wish to thank all my friends for their valuable suggestions and timely help.

Finally, I would like to say that I am indebted to my parents for everything that they have given to me. I thank them for the sacrifices they made so that I could grow up in a learning environment. They have always stood by me in everything I have done, providing constant support, encouragement and love.

**ASHISH GUPTA**

# ABSTRACT

Automatic Question Answering is a type of information retrieval. Given a collection of documents (such as the World Wide Web or a local collection) the system should be able to retrieve answers to questions posed in natural language. Automatic Question Answering is regarded as requiring more complex natural language processing (NLP) techniques than other types of information retrieval such as document retrieval, and it is regarded as the next step beyond search engines. Looking at increasing trend in distance education and availability of online E-Learning material; students need a question answering system to effectively utilize the material and to improve E-Learning.

This report presents an automatic Question Answering System (QAS) for E-Learning domain. The accuracy of answers has been increased by incorporating various NLP tools and a novel Word Sense Disambiguation (WSD) algorithm. The approach used is to utilize domain knowledge as much as possible to improve the performance of the system. The system utilizes template based approach to extract quality answers from passages. The WSD algorithm is designed specifically for closed domain question answering systems by utilizing the WordNet (English dictionary) and domain corpus (domain dataset). The WSD algorithm is applied in query expansion phase of the question answering system to expand query terms for relevant senses only.

The question answering system and WSD algorithm have been implemented in C/C++ on Linux platform using various tools such as Wordnet 3.1, automatic question classifier, NE recognizer, SEFT (retrieval engine) and Beagle desktop search tool.

# CONTENTS

# CHAPTER 1
# INTRODUCTION

## 1.1 Introduction

The Internet (originally known as ARPANET) was created in 1969 to provide an open network for researchers. In the last decade, the phenomenal growth and success of the Internet is changing its traditional role. Due to recent advances in technology and increased number of online users, information available on World Wide Web is increasing at exponential rate. Same trend has been seen in the field of distance education and E-Learning. Numbers of students in distance education courses are increasing at a rapid speed. More and more universities are providing online educational content on their web sites like MIT open course ware, UC Berkeley etc. As users struggle to navigate the wealth of online information now available, the need for automatic question answering becomes more urgent. We need systems that allow users to ask questions in every language and receive an answer quickly and succinctly, with sufficient text to validate the answer. Current search engines can return ranked list of documents, but do not deliver answers to users.

Automatic question answering systems address this problem. Recent successes have been reported in a series of questions answering evaluations that started in 1999 as part of the Text REtrieval Conference (TREC). The best systems now available are able to answer more than two third of the factual type questions. To answer a question, a system must analyze the question, perhaps in the context of some ongoing interaction; it must find one or more answers by consulting its online or off-line resources; and it must present the answer in some appropriate form, perhaps with associated justification or supporting material.

Automated question answering has been a topic of research and development since the earliest AI applications. Computing power has increased since the first such systems

were developed, and the general methodology has changed from the use of hand-encoded knowledge bases about simple domains to the use of text collections as the main knowledge source over more complex domains. Still, many research issues remain. The focus of this thesis is on the use of restricted or closed domains for automated question answering. A main characteristic of question answering in restricted domains is the integration of domain-specific information that is either developed for question answering or that has been developed for other purposes.

## 1.1 Statement of the Problem

The aim of this thesis is to design and implement an automatic question answering system for E-Learning domain, which helps students in accessing the E-Learning resources in an effective way by allowing them to ask questions in natural language.
The above problem can be divided into following sub-problems:

1. To design and implement a fully automatic closed domain question answering system for E-Learning which can be targeted to any course domain (like operating system, computer networks etc).

2. To design and implement a word sense disambiguation (WSD) algorithm for closed domain question answering system to remove ambiguities in questions asked by students.

## 1.2 Organization of the Dissertation

The report is divided into seven chapters including this introductory chapter. The rest of this thesis is organized as follows.

Chapter 2 gives an overview of E-Learning, automatic question answering and word sense disambiguation. It also presents main approaches to question answering and classification of question answering system. It discusses related work on E-Learning, question answering and word sense disambiguation and research gaps.

The proposed architecture for closed domain question answering system is discussed in Chapter 3. It then explains the working of overall system along with working details of individual modules of the system.

Chapter 4 provides implementation details and details on tools used to build the system.

Chapter 5 discusses various implementation issues of the system. Is also discusses the steps that led to final design of the system.

Chapter 6 discusses the experimental results performed on the system.

Chapter 7 presents final conclusions and scope for future work.

# CHAPTER 2
# BACKGROUND AND LITERATURE REVIEW

## 2.1 E-Learning

Web-based teaching materials, multimedia CD-ROMs or web sites, discussion boards, collaborative software, e-mail, wikis, computer aided assessment, educational animation, simulations, educational games and learning management software etc. The list is endless. All computer-based educational applications are grouped under E-Learning. **E-Learning** is an all-encompassing term generally used to refer to computer-enhanced learning often extended to include the use of mobile technologies such as PDAs and MP3 players. E-Learning is made up of several methods of learning, which are enhanced or facilitated by technology.

E-Learning is naturally suited to distance learning and flexible learning, but can also be used in conjunction with face-to-face teaching, in which case the term Blended learning is commonly used. Blended learning is the combination of multiple approaches to learning. Blended learning can be accomplished through the use of 'blended' virtual and physical resources. A typical example of this would be a combination of technology-based materials and face-to-face sessions used together to deliver instruction. Distance education or distance learning on the other hand, is a field of education that focuses on the pedagogy, technology, and instructional systems design that are effectively incorporated in delivering education to students who are not physically "on site" to receive their education. Instead, teachers and students may communicate asynchronously (at times of their own choosing) by exchanging printed or electronic media, or through technology that allows them to communicate in real time (synchronously).

Improvements in E-Learning hid the disadvantages of distance education like; lack of face to face interaction with teacher and feeling of isolation experienced by distance education students by use of audio/video web conferencing and collaborative learning softwares. These improvements in E-Learning brought success in distance education

courses, a large number of students utilize the facilities provided by distance education web sites from remote locations.

## 2.2 Need for Question Answering System

The rapid success of distance education has led to extensive development of course material and its placement on web. Course websites are continuously uploading new course data on their websites to make courses up to date [1]. Uses and Gratifications (U&G) is a community to examine Internet usage motivations of technology students enrolled in an Internet based distant education course and found that digital content (E-Learning content) is highly sought after by students in Internet-supported distant education classes [2]. Currently a sea of information is available in the form of power point slides, FAQ's and e-books on course websites. However a large part of such information remains unutilized because of the lack of effective information retrieval systems. Current search engines like Google are able to handle a large amount of information. Such search engines can provide very fast access to information from any domain. But the problem with such search engines is that they only return ranked list of documents. These engines are not effective for such E-Learning documents and it is very difficult for students to find answers to their queries using theses search tools.

A student does not understand and know where he can find the related terms and concepts mentioned in the lecture. Searching for answers to queries in such huge amount of data is a cumbersome task. Moreover currently available search engines does not provide much help in finding answers to queries, because search engines again redirect user to large number of documents and problem remains same. This requirement of students can only be fulfilled with a proper interface to access this information. Automatic question answering systems are a great help in such a case, which can retrieve answer to student questions from large information in a very effective way. Moreover students can ask questions directly in natural language rather than specifying only the keywords for search. The automatic question answering systems will be great help for problems of beginner students like addressed in [3], whose queries generally include

basic questions related to a new subject. The question answering system designed and developed as part of this work provides the solution to this problem of students. This question answering system provides a great ease to students in their learning process.

## 2.3 Question Answering Systems

Question Answering is a type of information retrieval. Given a collection of documents (such as the World Wide Web or a local collection) the system should be able to retrieve answers to questions posed in natural language [4]. Question answering is regarded as requiring more complex natural language processing (NLP) techniques than other types of information retrieval such as document retrieval, and it is regarded as the next step beyond search engines.

Question Answering research attempts to deal with a wide range of question types including: fact, list, definition, How, Why, hypothetical, semantically-constrained, and cross-lingual questions. Search collections vary from small local document collections, to internal organization documents, to compiled newswire reports, to the World Wide Web.

Main approaches to automatic question answering are explained in [5]. Generally question answering system use Natural Language Processing (NLP), Information Retrieval (IR) or template based approach depending upon the requirements of the system. A common feature of NLP systems is that they convert text input into formal representation of meaning such as logic (first order predicate calculus), semantic networks, conceptual dependency diagrams, or frame-based representations. Since the early days of NLP, Question answering systems simulated human intelligence within the Natural Language (NL) understanding research field. They worked as Natural Language (NL) front-end to databases [6-7], dialogue systems or story comprehension systems. Question answering systems were limited to specific and narrow domains such as algebra, astronomy and natural science.

Information Retrieval deals with the representation, storage, organization of and access to information items. Automated question answering is related to IR as users submit queries in order to find answers to questions they have in mind. IR systems are traditionally seen as document retrieval systems, i.e. systems that return documents that are relevant to the user's information need, but that do not supply direct answers. A further step towards the question answering paradigm is the development of document retrieval systems into passage retrieval systems [8], which focus on retrieving text passages rather than entire documents. Passage retrieval is now a standard component of modern IR-based question answering applications (IR QA), such as [9] and [10].

Template-based question answering extends the pattern matching approach of natural language (NL) interfaces to databases where the intelligence of the system is embodied in a collection of manually created question templates. Question templates embody the major part of the "intelligence" of the system and therefore are created manually; automatic generation of the templates would require another manually created knowledge base. Machine learning may help, but we haven't found any evidence of extensive use of machine learning in creating question templates. A fine-tuned system can reach as high recall/precision levels. Lin [11] has noted that template-based QA is effective because the distribution of user queries follows Zipf's law – a small fraction of question types account for a large number of questions asked. It means good template designs can answer most of the questions generally asked.

## 2.4 Classification of Question Answering Systems

Depending on the collection, Question Answering Systems are classified in to two categories [4]:

**2.4.1 Open Domain Question Answering Systems:** *Open-domain* question answering deals with questions about nearly everything, and can only rely on general world knowledge. These systems usually have much more data available from which to extract the answer. Such systems cannot, therefore, rely on hand

crafted domain specific knowledge to find and extract the correct answers. Example of famous open domain question answering system is Ask.com (formerly AskJeevs.com) [12], START (from MIT) [13] and AnswerBus [14]. Most of the open domain question answering systems utilizes the template based approach, the idea behind this is; question templates provide good coverage to generally asked questions. Using thorough NLP processing for question answering in open domain takes a lot of time due to unstructured and large amount of data. Therefore open domain question answering systems are empowered by keyword search and template based approaches.

**2.4.2 Closed Domain Question Answering System:** It deals with questions under a specific domain (for example, medicine or E-Learning), and can be seen as an easier task because NLP systems can exploit domain-specific knowledge. A number of closed domain question answering systems has been built e.g., LUNAR [15] and BASEBALL [16]. Question answering systems for closed domains use unstructured (Free Text), semi-structured (XML) or structured (Databases) data. Closed domain question answering systems that use structured or semi-structured data provide more accuracy than the systems with unstructured data. But accuracy comes with the price of manually structuring the domain data for question answering system. In some cases, structuring domain data for closed domain still not possible due to large amount of heterogeneous data such as in the case of E-Learning (this system). For such systems part of domain information can be structured or meta information of domain can be used to improve accuracy.

## 2.5 Word Sense Disambiguation

Word sense ambiguity is rarely thought of as a problem in our daily life. Most of time, even we are presented with a text where ambiguous word appears in more than one of its senses, we can easily identify the correct sense of the word without getting confused of the other senses. The disambiguation process which helps people identify the correct sense of a word is not difficult for us. We can perform it easily and accurately.

However, this disambiguation process is not easy for computers. Given a text including an ambiguous word like "crane", without performing some sort of disambiguation, it is impossible for machines to know whether we are talking about *the machine that lifts and moves the heavy objects or the large long necked wading bird of marshes and planes in many parts of the world.* And, unfortunately, even when after disambiguating words, machines may not be able to resolve ambiguities. The task of word sense disambiguation is to make machines perform as well as people identifying the senses the words in the context.

Question answering and Information Retrieval (IR) are major fields where word ambiguity is a problem. The rapid increase in the number of electronic documents available on World Wide Web and increased desire to obtain useful information from these documents have increased need for the development of sophisticated question answering and information retrieval systems. Especially is research oriented restricted domains the importance of precise and efficient systems is indisputable.

## 2.6 Previous Work

### 2.6.1 E-Learning

E-Learning has underlined the importance of quick access to relevant study material for effective education with the major advantage of enabling people to access learning facilities regardless of their location and at the time that is most convenient to them. Business enterprises are widely using this online learning for employee training and education because of its cost saving advantages, especially with respect to time and travel parameters [17].

Efforts have been made in the direction of providing ease to the student in extracting information from E-Learning documents with respect to effective retrieval and presentation of knowledge. A similar system COVA (Content-based Video Access) enables remote users to access specific parts of interest from a large lecture database by contents [18]. COVA is system architecture for implementing web based E-Learning

system and XML-based semi-structured model for content based lecture access. However manual development of XML schemas or annotating the vast amount of information can be laborious and impractical. Another approach introduces Genetic Algorithms into traditional question answering system which uses the concept of Case-Based Reasoning (CBR) [19]. The huge number of cases that would be generated with large repository (with continual growth) and failure in case of complex queries put limitation in its practical use.

A different approach taken in Knowledge-based Content Navigation in E-Learning Applications presents a prototype implementation of the framework for semantic browsing of a test collection of RFC documents [20]. They propose the use of fuzzy clustering algorithms to discover knowledge domains and represent those knowledge domains using TopicMaps. However success largely depends on how accurately the clusters are identified and the representation still suffers from the drawback attributed to Table-of-Content page.

E-Learning Media Navigator (ELM-N) from IBM Research is a system with which a user can access and interact with online heterogeneous course materials [17]. Their efforts are aimed to reduce human effort and manual annotation work in order to make the system viable for voluminous information. Furthermore, challenges remain in the area of easy to use content delivery, access and augmented interaction.

### 2.6.2 Question Answering System

A QAS provides direct answers to user questions by consulting its knowledge base. It attempts to allow user to ask questions in natural language and receive an answer quickly and succinctly, with sufficient context to validate answer [21]. QAS that cater to a specific domain have been developed at very early stage. LUNAR [15] was such a closed domain QAS that answered questions related to moon rocks and soil gathered by Apollo 11 mission. However it relied on having the data to be available in a highly structured form, not as completely unstructured text.

Most of the QAS that have been developed treat the web as a collection of documents and thus cater to huge variety of questions. The commercial search engine known as AskJeeves responds to natural-language questions but its recall is very limited because it uses its knowledge base (which is at least partially hand constructed) to answer questions and updates the knowledge base when asked a question which it has not encountered before. Another QAS, MULDER [22] is claimed to be the first general-purpose, fully-automated question-answering system available on the web. MULDER's architecture relies on multiple search-engine queries, natural-language parsing, and a novel voting procedure to yield reliable answers (recall of same level as Google). However, the difficulty of Natural Language Processing (NLP) has limited their ability to give accurate answer to questions that are quite specific to a domain. In addition to the traditional difficulties associated with syntactic analysis, there remains many other problems to be solved, e.g., semantic interpretation, ambiguity resolution, discourse modeling, inference, common sense, etc.

START [23] is one of the first question answering systems with a web interface, having been available since 1993. Earlier START was only focused on questions about geography and the MIT InfoLab, now a number of more domains have been included to it. START uses a precompiled knowledge base in the form of subject-relation-object tuples (T-expression), and retrieves these tuples at run time to answer questions. The basic idea behind START is to represent the whole knowledge of interest in a useful manner like subject-relation-object tuples. Whenever a question is asked it answer from its knowledge base. The problem with START is that if more information is required to enter into the system, then it has to be precompiled before it can be used. Creating its knowledge base will require a lot of time, because each sentence is completely analyzed before its T-expressions are added to knowledge base. So for proper working of START we need and equal or larger size of knowledge base than the size of actual data set.

Another interesting question answering system for E-Learning is explained by Feng et al in [24], which is a discussion bot that answers questions asked by students in a discussion forum. But the system's knowledge is limited to the questions previously asked by students in discussion forum, which makes the course related E-Learning data available on course website unutilized. Answers to complex questions are generally answered by the students first; system redirects students to proper location if same question is asked again. The E-Learning data on course websites are a very good source of information to answer many queries of students and it must be utilized. QUESTAL [25] is closed domain question answering system that answers question related to Nobel Prize winners and Language technology. QUESTAL uses various NLP techniques and is extensible to multilingual support. QUESTAL's reliability was completely dependent on structured data.

QAS on web try to answer questions that require a fact or one word answer. This is difficult for specific questions because the targeted domain is unrestricted and no assumption can be judiciously made. E-Learning questions are more complex than TREC (Text REtrieval Conference) type questions as they require domain knowledge and long answers need to be extracted from multiple documents. Moreover these questions have ambiguity inherent in them. The objective here is to allow the user to submit exploratory, analytical, non-factual questions, such as *"How does Mergesort sort an array?"*. The distinguishing property of such questions is that one cannot generally anticipate what might constitute the answer. While certain types of things may be expected, the answer is heavily conditioned by what information is in fact available on the topic. Users generally prefer answers embedded in context, regardless of the perceived reliability of the source documents [26]. When users search for a topic, increasing the amount of text returned to users significantly decreases the number of queries that they pose to the system.

### 2.6.3 Word Sense Disambiguation

Words in natural language are known to be highly ambiguous. This is especially true for the frequently occurring words of a language. For example, in the WordNet dictionary,

the average number of senses per noun for the most frequent 121 nouns in English is 7.8, but that of the most frequent 70 verbs is 12.0 [27]. This set of 191 words is estimated to account for 20 percent of all word occurrences in any English free text. Therefore, word sense ambiguity is prevalent problem in Natural Language Processing (NLP).

Research into automatic resolution of word senses has been going so for at least forty five years, and there is a large literature describing a variety of different word sense disambiguation techniques (WSD). Earliest WSD methods used hand coding of knowledge to disambiguate word senses. In theses systems, each word to be disambiguated would need to be hand tagged with correct piece of information, e.g., part of speech, sense etc, which would be useful in disambiguation process. Therefore, it was difficult to come up with a comprehensive set of necessary disambiguation knowledge and even more difficult to manually maintain and further expand the disambiguation knowledge to handle real word sentences.

In order to solve this problem, some researchers decided to use pre-coded information in the form of machine readable dictionaries and thesauri [28-29]. Other started to build their own dictionaries and thesauri with information concluded from statistics over large corpora. This approach is called *corpus based approach* [28]. In contrast to manually hand-coding disambiguation information into a system, the corpus based approach uses machine learning techniques to automatically acquire disambiguation information, e.g., verb object relations, from large corpora.

Corpus based WSD systems can broadly be classified into *supervised approach* and *unsupervised approach*. Most research efforts in unsupervised WSD rely on the use of knowledge contained in a machine readable dictionary. A widely used resource is WordNet [30], which is a public domain dictionary containing more than 118,000 different words forms and more than 90,000 different word senses. The IS-A relationship in WordNet's class hierarchy is an important knowledge exploited in unsupervised WSD algorithms.

Resnik [31] gives a good example of an unsupervised WSD algorithm. In his method, Resnik used the verbs, adjectives or nouns which modify the word to be disambiguated. For example, in disambiguating the sense of *coffee* in the test sentence *drinking coffee*, Resnik's algorithm used the verb *drinking* to find other words which might be modified by it in a similar fashion to *coffee*. Words like *milk, wine, tea* are usually used with the verb *drinking* and the sense common to all these words is the *beverage* sense. Therefore, he concludes that he is looking for the *beverage* sense of the word *coffee*.

Schutze [32], describes a corpus-based disambiguation method which builds a vector representation of word meanings for the purpose of identifying different meanings of a word. For this, Schutze forms four-gram co-occurrence matrices and context vectors for each target word by normalizing the sum of the co-occurrence matrices of the four-grams around the target word. He then plots these context vectors in his multi-dimensional space marked by the co-occurrence matrices. In this space, the context vectors which are distance-wise close to each other refer to the same sense of the target word. After hand-labeling each of these clusters with the correct meaning, on a task of pair wise disambiguating 10 well-known ambiguous words, the system achieved an average accuracy of greater than 92%.

Resnik [31] provides an extension to Schutze's work by describing an automatic method of labeling the different clusters of contexts. Resnik's method is based on the observation that: "when given a list of words, a human being will assign to an ambiguous word in the list, the meaning which will match the sense of the rest of the words in the list." To perform this meaning assignment automatically, Resnik uses the IS-A hierarchy of WordNet. In this hierarchy, each word is assigned an informative-ness level which is inversely proportional to the frequency with which the word occurs in English language. Once he calculates these informative-ness levels, for each word in the hierarchy, Resnik picks the sense with the highest informative-ness measure by assigning to each sense the sum of the informative-ness measures of all the ancestors that support that particular

sense. On a task of identifying the 23 different 14 meanings of the word 'sense', the system achieved 60% accuracy.

## 2.7 Research Gaps

Several systems have been made to improve E-Learning; and some have been designed to improve information access in E-Learning documents. There are two types of systems, which have been used for information access in E-Learning till date:

- **Search Engines:** Search engines provide fast access to documents even if documents are unstructured. But search engines do not fulfill the requirements of students to answer complex queries, because they only redirect students to ranked documents.

- **Question Answering System:** The question answering systems designed till date for E-Learning; either use manually structured or semi-structured data set or semi automatic, need some help (instructor's) to answer some complex questions. Manually structuring large amount of E-Learning documents is cumbersome task and instructor may not be available for distant students for all the times.

The systems used so far for E-Learning do not fully utilize the domain knowledge to improve information access, and removing ambiguities in the questions asked by students.

The proposed system fills the addressed research gaps by providing a fully automatic system; that can answer every type of student queries from the unstructured E-Learning documents. Also system uses a novel word sense disambiguation algorithm, which utilizes domain knowledge to remove ambiguities in questions asked by students.

# CHAPTER 3

# PROPOSED SYSTEM

## 3.1 Architecture

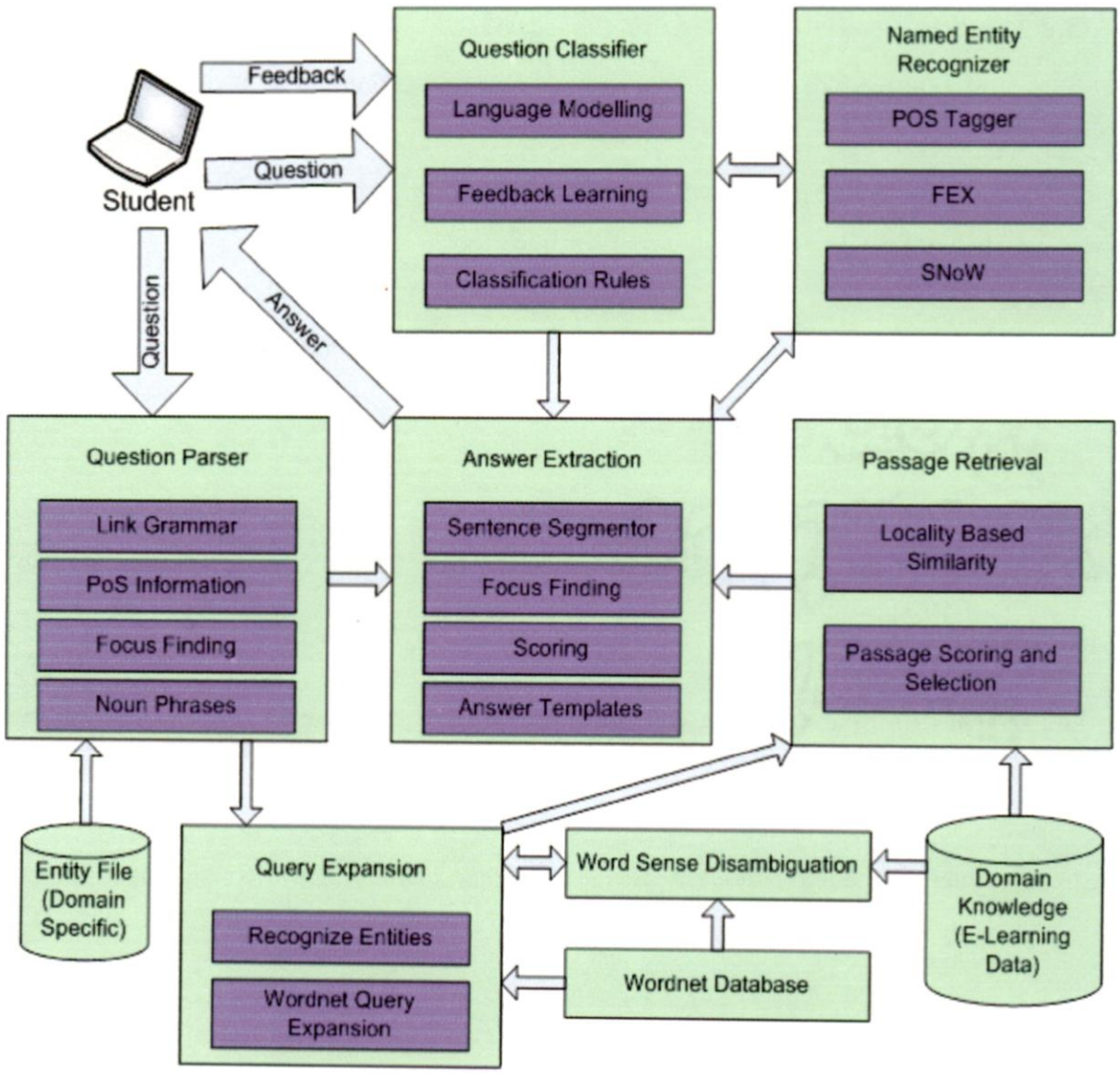



Fig 3.1: Architecture of Question Answering System

Figure 3.1 shows the architecture of proposed automatic question answering system. The thick arrows represent the information communication with the student and thin arrows represent the internal information flow of the system. Double headed arrows represent two way information flows between components of the system. The working of system starts when a student submits a question to the system.

## 3.2 Working

Step by step procedure to retrieve answers to question submitted by user is given below:

1. The question submitted is parsed using *link grammar* based parser, which generates a linkage structure of the question.

2. Simultaneously question class is identified using automatic *question classifier*. Question classifier uses *Named Entity (NE) recognizer* to automatically recognize identities in the question.

3. The linkage structure retrieved as result of question parsing is analyzed to retrieve useful information like subject, verb, object and part of speech etc.

4. The information retrieved (query terms) are expanded using *query expansion* module.

5. Query expansion module uses *word sense disambiguation* algorithm to expand only for relevant synonyms.

6. The expanded terms are now fed to *retrieval tool (SEFT)* to retrieve passages.

7. Top three passages returned by retrieval tool are further used by *answer extraction* module to extract specific answers based on the class of the question.

8. After final answers are presented to user, if user is not satisfied with the class identified by system. For example, if user has submitted a question "*who is president of India?*" and system has recognized its class as LOCATION. But correct class of question is PERSON, because answer is a person name. User can specify the correct class using *feedback system.*

9. The correct class is learned by the question classifier to improve its accuracy. System then retrieves answer based on new class and present answer to user.

Each module is described in detail in further sections.

### 3.2.1 Parsing and Semantic Analysis

The parser used in system is link grammar based parser [33]. A link grammar consists of a set of words (the terminal symbols of the grammar), each of which has a linking requirement. A sequence of words is a sentence of the language defined by the grammar if there exists a way to draw links among the words. The linking requirements of each word are contained in the dictionary used by parser. A set of links that prove that a sequence of words is in the language of a link grammar is called a *linkage*. An example of linkage structure is shown in figure 3.2.

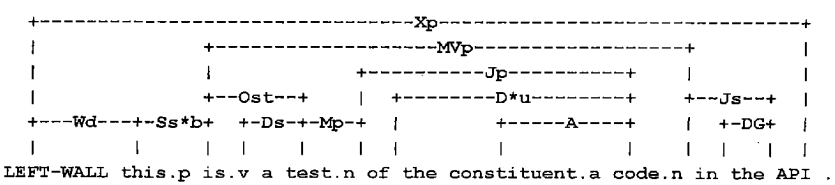



Fig 3.2: Example linage structure and phrase identification

The link grammar based parser gives the linkage structure of the sentence which specifies links between different words. These links can be link between subject and verb, link between object and verb and link between adjective and noun etc. Links in a linkage structure are labeled with characters, which specify the type of link. There can be multiple parses of a sentence, all of which are processed during semantic analysis. Parser also appends part of speech (PoS) information with words of sentence.

Constituent phrases are words that are reachable from certain links, tracing in certain direction. Example in figure 3.2 shows constituent phrases found in sentence. A verb phrase is everything reachable from an "S" link, tracing to the right that is, not tracing through the left end of the "S" link itself. For noun phrases there are several possibilities. Anything that can be reached from an "O" link tracing right is a noun phrase. The system finds all possible noun phrases and focus in the sentence.

During analyzing linkage structure of question system also takes care of the domain specific entities using entity file. The reason for storing considering domain specific entities separately is that; parser's dictionary may not contain some important domain specific keywords and can ignore theses words. But, domain keywords are important part of a question. It uses any domain term found in question as a keyword for retrieval engine.

### 3.2.2 Named Entity Recognition

Named Entity recognizer is a natural language processing tool that takes as input a sentence and marks entities in that sentence like person names, location names and organization name etc. For example:

Input: "*John Doe is American, lives in Champaign and works for the CIA.*"

Output: *"[PER John Doe] is [MISC American], lives in [LOC Champaign] and works for the [ORG CIA]."*

Named Entity (NE) [34] recognizer is taken from Cognitive laboratory of UIUC (University of Illinois at Urbana-Champaign). NE recognizer is used in task of question classification by automatic question classifier used. It is also used in the task of answer extraction to recognize entities in the passages returned by retrieval engine; for example in cases where question is classified as PERSON, LOCATION or ORGANIZATION etc. NE recognizer uses various natural language processing tools like: sentence splitter, word splitter, feature extraction tool and learning architecture to perform the task of entity recognition. The details on its working are given in next chapter.

### 3.2.3 Question Classifier

The question classifier module takes as input a question and gives category of answer or what is the class of question as output. For example:

Input: "*Who made linux operating system?*".

Output: PERSON

The answer of the question will be a person name. One approach to question classification is to determine the question type based on the sentence structure and key words, which represent syntactic and semantic information respectively. A set of patterns are defined and hard-coded, often with regular expressions. When a new question comes, it is matched against those patterns to find the class it belongs to. As the pattern set gets more complete and accurate, the performance of this approach will become better. So we always have the problem of defining more and more question patterns to improve the model.

To make the process of question classification more dynamic and automatic, system uses Li. Wei's [35] question classifier, this classifier exploits language modeling and regular expression model. In this approach, the models can be automatically constructed from

the training set, and its performance is better than other approaches. As for the question answering task, [35] has built one language model for every class of questions based on the training data set. To classify a question, the probability of generating it is calculated for each class based on its language model, and the highest probability determines the classification. The classifier used provides two language modeling techniques. This system uses combination of language modeling and regular expression model to achieve maximum accuracy. Question classifier first marks entities in the question using NE recognizer, and then it proceeds with the task of classification.

### 3.2.4 Query Expansion

| Words Removed | | | | |
|---|---|---|---|---|
| By | Is | So | As | Then |
| To | Otherwise | The | Will | That |
| An | In | For | Of | This |
| Does | At | Are | Did | On |
| Be | Over | We | Our | Upon |

Table 3.1: Example words that are removed

The result of semantic analysis of linkage structure is set of terms and noun phrases in question, which are expanded and then used to retrieve passages from domain dataset using retrieval engine. Elementary words like given in the table 3.1 are ignored. The system is based on the following idea: A document relevant to our query might contain either the words in the query or their *synonyms*. This implies that we should be able to improve recall by considering the synonyms as a part of the IR query. If we do not include synonyms in IR query; retrieval engine may skip important parts of the document affecting the recall of the system. However, if we include all possible synonyms of the

query in our retrieval, precision will suffer. In order to increase both precision and recall, we need to use only the *relevant* synonyms in a context.

The expansion of terms is performed using WordNet [36], details on WordNet are given in next chapter. There may be multiple senses present of a single term found in question, but only one sense of term is valid in any given sentence. Naïve approach of query expansion is to include synonyms from all senses or randomly from all senses. But this may bias the retrieval engine to search for wrong terms, ultimately to the retrieval of wrong and unwanted passages. To overcome this problem query terms are disambiguated using *word sense disambiguation* algorithm. We can identify these *relevant* synonyms with the help of a disambiguation algorithm and only relevant terms are then searched using retrieval engine.

### 3.2.5 Word Sense Disambiguation

Traditionally, the input to a Word Sense Disambiguation (WSD) program consists of unrestricted, real-world English sentences. In the output, each word occurrence w is tagged with its correct sense number (which appears in a previously agreed dictionary) according to the context. For this work, the system uses the sense definitions as given in WordNet, which is comparable to a good desktop printed dictionary in its coverage and sense distinction [27]. Since WordNet only provides sense definitions for content words (i.e., nouns, verbs, adjectives and adverbs), the system is only concerned with disambiguating the senses of content words. Almost all previous work, as well, in WSD deals only with disambiguating content words [27].

Most WSD algorithms focus simply on better disambiguation, rather than making use of a disambiguating algorithm. For this reason, previous work has focused on different learning methods and different ways of inferring the meaning of the words in a context. The applications of WSD to other unsolved problems like IR have not been investigated in as much depth and have so far revealed contradicting answers. Word sense ambiguity is one of the causes of poor performance in IR systems. Polysemy (a single word form

having more than one meaning) and Synonymy (multiple words having the same meaning) both reduce the performance of IR systems. Polysemy reduces precision by causing false matches whereas synonymy reduces recall by causing true conceptual matches to be missed [37]. Therefore, especially in the systems which expand the query on the synonyms of the word before processing the query, the IR performance can be improved if the query can be perfectly disambiguated.

Many of the old systems try to improve the performance of IR by retrieving documents from a disambiguated corpus, rather than retrieving disambiguated queries from a regular corpus. This work focus not on retrieving information from a disambiguated corpus but on retrieving disambiguated information from a regular corpus. More specifically, if system can identify the correct synonyms of the content words in a natural language query, it will increase both precision and recall by expanding the query with the correct synonyms of these words.

Most previous corpus-based WSD algorithms determine the meanings of polysemous words by exploiting their local contexts. The basic intuition that underlies these algorithms is that two occurrences of the same word should have identical meanings if they have similar local contexts. In other words, in order to disambiguate a certain word, most previous corpus based WSD algorithms observe the previous usages of that word and learn classifiers for it. Each of these classifiers holds the information necessary for identifying one sense of the ambiguous word. There are several disadvantages to this approach. First of all, it is very difficult to learn good classifiers for each word since a word must be encountered thousands of times before a good classifier can be learned for it. There are thousands of polysemous words. For example, there are 11,562 polysemous words in WordNet. In order for each polysemous word to appear thousands of times each in a corpus, the corpus must contain billions of words. The second major drawback is that since these algorithms learn to disambiguate a word from its previous usages, these algorithms cannot deal with the words for which classifiers have not been learned yet.

In order to avoid these drawbacks, this system uses an algorithm which does not require learning classifiers for each polysemous word. Instead it tries to learn the meaning of each polysemous word by looking at the context it is used in and by comparing it with other words that appear in the same exact context. This method still allows using the information implied about the meaning of the word by its context; however, it saves the trouble of having to learn classifiers for each word.

The main idea is that the context of a word *w* indicates the meaning of *w*. Therefore, words used in the same context as *w* should have similar meanings to *w*, or at least give a good idea about which sense *w* is used in. In other words, two occurrences of a word and its synonym *belong to the same sense* if they have similar local contexts. This means that use of local contexts, and the synonyms of a word used in those contexts can be used to identify which of its senses an ambiguous word is used in.

This approach used does not require an ambiguous word to exist in the corpus since it does not need to learn the meaning of the word from its previous occurrences. Other advantages of this approach include the following:

- No specific classifier needs to be learned for each word. Instead it uses the same knowledge sources for all words. In this case the main knowledge source is the question asked by student where system finds words appearing in a context.

- In order to accomplish thorough training, most algorithms which learn classifiers use very large sense-tagged corpora. This algorithm can identify the senses of the ambiguous words that best fit into a context without needing sense-tagged corpora.

- The frequency with which a certain ambiguous word appears does not affect the performance of the system. But the frequency with which the context appears in

the corpus affects chance of finding the best sense of the word in that context. So, in theory, this algorithm should be able to deal with words that are infrequent or do not even appear in the corpus.

The WSD algorithm is based on two assumptions:

- *Assumption 1: In a given sentence, each word is represented by its single sense.*

  *Assumption 2: WordNet contains all the words needed to disambiguate, along with all senses of the word.*

Assumption 1 is very strong, except in some cases where not much contextual information is present in sentence. In such cases even humans find it difficult to disambiguate terms. Assumption 2 is also quite strong because WordNet contains more than 118,000 words, exception are some domain specific words that might be present in a sentence. Using the data, index files and utility functions of WordNet, system can extract information about the synonym sets of words. Using WordNet simplifies WSD problem to a certain degree, and disambiguation of a certain word reduces to identification of the correct synonym set of the word in a certain context.

System can identify the "correct", i.e., the most relevant, synonym set of a word in a context by matching the synonym sets of this word from WordNet against a set of words which have been used in the same context as query word $w_m$. Humans can resolve sense ambiguities by looking at a narrow window of words surrounding the ambiguous word. This fact leads to identify the correct meaning, i.e., the best synonym set of a word $w_m$, by looking at its context. What is more, the content words, i.e., nouns, verb, adjectives, and adverbs, which appear in the same context, are usually related to specific senses of each other. Therefore, identifying the correct sense of one of the words in a context can help in identifying the sense of other words in the same context. Alternatively, the use of certain content words in a context can help in identifying the general sense of the context.

The approach used here to remove ambiguity is similar to approach used by humans to remove ambiguities of words. First approach is to use the words surrounding the ambiguous words to remove ambiguity of word. For example in sentence "*In a three-dimensional space, another important way of defining a plane is by specifying a point and a normal vector to the plane.*", humans can easily identify that plane in sentence is a geometric plane. The information of surrounding words or context is used to remove ambiguity of word plane.

Second approach used in system is to utilize the domain knowledge of the system. To explain it properly lets take an example of mathematics teacher and his/her student, suppose student asks teacher a question "*what are the properties of a plane?*". Now teacher will understand that student is asking about geometric plane rather than an airplane. The reason for disambiguation is that; teacher and students most of the times talk about their subject which is mathematics in above example. Same rule will apply when a student will ask question from a question answering system.

Implementation of this algorithm takes as input terms $t_1$ $t_2$ $t_3$... $t_n$ along with their part of speech information that are extracted from question semantic analysis. For each term, we extract its synonym sets. In WordNet, the synonym sets are arranged into different groups according to the parts of speech of the word they match. For example, the word "leave" has two groups of synonym sets: one group for its noun sense, e.g., leave of absence, and the second for its verb sense. In each of these groups, the synonym sets are ordered from the most frequently used to the least frequently used. WSD algorithm extracts all synonym sets for all possible parts of speech of all words in a query. Once the synonym sets are extracted, it starts the process of disambiguation by calculating the scores of the individual synonym sets. Complete word sense disambiguation algorithm is given below:

**WSD Algorithm**

1. Given T = $\{t_1, t_2, t_3, \ldots t_n\}$ is a set of terms and noun phrases from student question for disambiguation.

2. Take an empty set X = $\{x_1, x_2, x_3, \ldots x_n\}$ which will contain disambiguated synonyms of terms present in T. Each element $x_i = \{x_iw_1, x_iw_2, x_iw_3, \ldots x_iw_m\}$ is a set of words which will contain disambiguated synonyms for term $t_i$ where $t_i$ will have m synonyms.

3. Copy terms from T to X, such that term $t_i$ will be placed in $x_i$.

4. For each term tj in T, where $1 \leq j \leq n$;

5. Look up in WordNet for every synset defined for ti and store them in set S = $\{s_1, s_2, s_3, \ldots s_p\}$ where $t_i$ has p senses/synsets

6. For each sense/synset in $s_i$ in S, where $1 \leq i \leq p$;

7. Find all synonyms for sense $s_i$ from Wordnet and store in set W = $\{w_1, w_2, w_3, \ldots w_q\}$ where sense $s_i$ has q synonyms.

8. Search the domain knowledge (E-Learning data) by query Q = ($w_1$ OR $w_2$ OR $w_3$ OR ..... OR $w_q$) AND ($x_1w_1$ OR $x_1w_2$ OR $x_1w_3$ OR ..... OR $x_1w_m$) AND ($x_2w_1$ OR $x_2w_2$ OR $x_2w_3$ OR ..... OR $x_2w_m$) AND ..... AND ($x_nw_1$ OR $x_nw_2$ OR $x_nw_3$ OR ..... OR $x_nw_m$).

9. Number of results returned by search is score of sense for its selection as disambiguated sense.

10. End for

11. Select the sense/synset with maximum score as disambiguated sense and store all synonyms for sense in $x_j$ where $t_j$ is term disambiguated.

12. End for

13. Set X will now contain synonyms for all disambiguated terms.

Term (Response) :
WN Sense: reaction
Beagle Search Command :beagle-query "Throughput" OR "Throughput" "clock time" OR "time" "reversal" OR "change of mind" OR "flip-flop" OR "turnabout" OR "Turnaround" "wait" OR "waiting" "reaction"
Total Results Found :0
Score: 0 Max Score: 0

WN Sense: answer--reply
Beagle Search Command :beagle-query "Throughput" OR "Throughput" "clock time" OR "time" "reversal" OR "change of mind" OR "flip-flop" OR "turnabout" OR "Turnaround" "wait" OR "waiting" "answer" OR "reply"
Total Results Found :16
Score:16 Max Score :0

WN Sense: reception
Beagle Search Command :beagle-query "Throughput" OR "Throughput" "clock time" OR "time" "reversal" OR "change of mind" OR "flip-flop" OR "turnabout" OR "Turnaround" "wait" OR "waiting" "reception"
Total Results Found: 5
Score: 5 Max Score: 16

WN Sense: reply
Beagle Search Command :beagle-query "Throughput" OR "Throughput" "clock time" OR "time" "reversal" OR "change of mind" OR "flip-flop" OR "turnabout" OR "Turnaround" "wait" OR "waiting" "reply"
Total Results Found :2
Score: 2 Max Score: 16

Fig 3.3: Example Beagle usage for WSD

For performing local search in E-Learning documents system uses Beagle [38] a desktop search tool. Beagle indexes the documents, so it performs very fast for local search and returns the documents names that satisfies query. Beagle allows user to build complex queries for search, for our purpose it supports AND-OR queries. Example usage of Beagle for WSD algorithm is shown in figure 3.3. Figure 3.3 shows disambiguation process of term R*esponse* in the question "*What is Throughput, Turnaround time, waiting time and Response time?*". WSD disambiguated *Response* with synonyms *Answer* and *Reply* with maximum score of sixteen.

### 3.2.6 Passage Retrieval

To extract passages from the collection of documents an Information Retrieval engine is needed which can analyze the keywords and passages in detail. The answers to a query are locations in the text where there is local similarity to the query, and similarity is assessed by a mechanism that employs as one of its parameters the distance between words [39]. For this purpose it was found that the locality-based similarity heuristic (in which every word location in each document is scored) provides retrieval effectiveness as good as the document-based technique, and has the additional advantage of presenting focused answer passages ( instead of whole document) with sufficient context to validate the answer. Therefore, the engine used SEFT (Search Engine For Text) [39] is based on this concept and has been customized for this application.

The important features of Locality-Based Retrieval (with Similarity) in this context are:

- The focus is on local context by considering top n ranked passages, instead of the top n documents.

- Each term has a certain scope, where its importance decreases with respect to the distance from that term.

- Similarity is computed as the sum of weighted overlaps between terms. It is based on intuitive notion that the distance between terms is indicative of some semantics of the sentence.

Rather than considering the text collection to be a sequence of documents, it is considered to be a sequence of words, and query term occurrences within the collection are presumed to exert an influence over a neighborhood of nearby words. Then, supposing that the influence from separate query terms is additive, the contribution of each occurrence of each query term is summed to arrive at a similarity score for any particular location in any document in the collection.

The top N (value set by the user) ranked passages (window surrounding the location) is returned after scoring all the locations of the query term according to the weightage assigned to them. The implementation also handles *case folding* and *Stemming* (to match up a keyword with any of its other grammatical forms) of word while searching the word.

### 3.2.7 Answer Extraction

Answer Extraction is module of the system where actual answers to question are extracted. This module is provided with question class, question parse information (e.g. focus, noun phrases), top N ranked passages. Now based on the class of the question, answer extraction module uses an answer template to find answer to that specific question. Answer template is a rule to obtain answer from passage given the class of that question.

Answer extraction module tries to answer factual questions in one word to one sentence of length and for lengthy questions it gives user sufficient text to validate answer. This module divides passages into sentences using sentence splitter. Then compares focus of question with focus of sentence, if found similar, then it uses NE recognizer to find identities in sentence. If there is any entity of interest (like PERSON etc) in sentence

then it is extracted as answer, and continues to apply same procedure on all sentences; example given in table 3.2. Finally it displays extracted answers.

| Question | Class | Template |
|---|---|---|
| Who made Linux operating system? | PERSON | The <focus> is made by <person> |
| What is maximum size of memory a 32-bit processor can address? | NUMBER | <focus> can address <number> of <focus>. |
| Where does Indian president live? | LOCATION | <focus> lives in <location>. |

**Table 3.1: Example templates in Answer Extraction**

Similarly answer templates have been designed for LOCATION, ORGANIZATION, URL, EMAIL and NUMBERS which includes sub classes like MONEY, TEMPARATURE, SPEED, MASS, DENSITY, LENGTH etc.

### 3.2.8 Feedback

Once answer along with its recognized class is given to user and user is not satisfied with the answer or classifier might have wrongly classified. In that case user can select correct class for it (where class specifies what user wants), then system will find answer to question according to that class, moreover the system will learn that class. This closed loop system also provides system a feedback learning, which improves accuracy of system.

# CHAPTER 4
# SYSTEM IMPLEMENTATION

The question answering system is implemented in C/C++ using Eclipse on Linux platform. Eclipse an open source integrated development environment (IDE) which provides support for C/C++ using an extension CDT (C/C++ Development tools). Not all parts of the complete question answering system are programmed in C/C++. NE recognizer used is developed in Perl. Question classifier used is developed in Java. Except these two all other component are developed in C/C++.

The details of the tools used in implementation of system are given in further subsections.

## 4.1 Link Grammar Based Parser

The parser used in system is link grammar based parser [33]. Most sentences of most natural languages have the property that if arcs are drawn connecting each pair of words that relate to each other, then the arcs will not cross. This well-known phenomenon, which is call *planarity*, is the basis of link grammars a formal language system described in [33].

A link grammar consists of a set of words (the terminal symbols of the grammar), each of which has a linking requirement. A sequence of words is a sentence of the language defined by the grammar if there exists a way to draw arcs (also called links) among the words so as to satisfy the following conditions:

> *Planarity:* The links do not cross (when drawn above the words).
> *Connectivity:* The links suffice to connect all the words of the sequence together.
> *Satisfaction:* The links satisfy the linking requirements of each word in the sequence.

The linking requirements of each word are contained in a dictionary. To illustrate the linking requirements, figure 4.1 shows a simple dictionary for the words a, the, cat, snake, Mary, ran, and chased. The linking requirement of each word is represented by the diagram above the word.

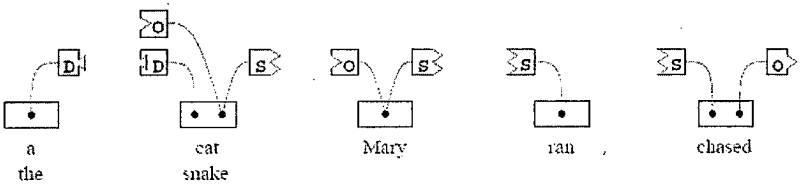



Fig 4.1: Example of linking requirements of words

Each of the intricately shaped labeled boxes is a *connector*. A connector is satisfied by "plugging it into" a compatible connector (as indicated by its shape). If the mating end of a connector is drawn facing to the right, then its mate must be to its right facing to the left. Exactly one of the connectors attached to a given black dot must be satisfied (the others, if any, must not be used). Thus, *cat* requires a D connector to its left, and either an O connector to its left or a S connector to its right. Plugging a pair of connectors together corresponds to drawing a link between that pair of words.

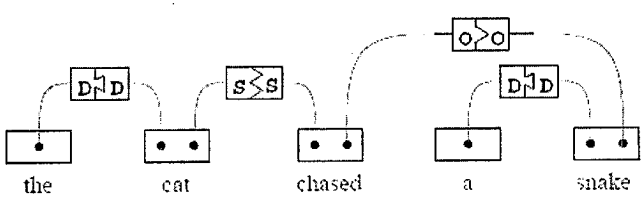



Fig 4.2: Example of satisfied linking requirements in a sentence

Figure 4.2 shows how the linking requirements are satisfied in the sentence "*The cat chased a snake*". (The unused connectors have been suppressed here.) It is easy to see that *Mary chased the cat*, and *the cat ran* are also sentences of this grammar. The sequence of words: *the Mary chased cat* is not in this language. Any attempt to satisfy the linking requirements leads to a violation of one of the three rules. Here is one attempt in figure 4.3.

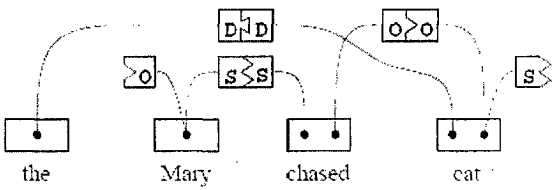



Fig 4.3: Example of unsatisfied linking requirements in a sentence

Similarly *ran Mary*, and *cat ran chased* are not part of this language.

A set of links that prove that a sequence of words is in the language of a link grammar is called a *linkage*. There is a succinct computer readable notation for expressing the dictionary of linking requirements. The linking requirement for each word is expressed as a formula involving the operators (like &, or), parentheses and connector names.

The parser has a grammar of roughly 700 definitions that captures many phenomena of English grammar. It handles: noun-verb agreement, questions, imperatives, complex and irregular verbs (*wanted, go, denied,* etc.), different types of nouns (mass nouns, those that take *to*-phrases, etc.), past- or present-participles in noun phrases, commas, a variety of adjective types, prepositions, adverbs, relative clauses, possessives, and many other things.

The parser reads in a dictionary and parses sentences according to the link grammar. It does an exhaustive search – it finds every way of parsing the given sequence with the given link grammar. It can parse a sentence with complexity $O(n^3)$, where n is the number of words in a sentence. It also makes use of several very effective data structures and heuristics to speed up parsing. The parser is comfortably fast can parse typical newspaper sentences in a few seconds.

## 4.2 Named Entity Recognizer

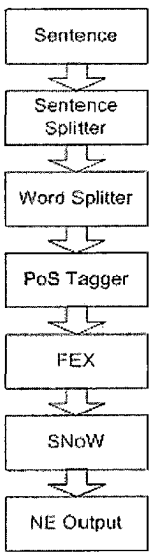



Fig 4.4: Named Entity Recognition Process

Named Entity (NE) [34] recognizer used in system is taken from Cognitive laboratory of UIUC (University of Illinois at Urbana-Champaign). NE recognizer used in this system is very sophisticated and uses various linguistic tools like sentence splitter, feature

extraction tool (FEX), learning tool (SNoW) to perform the task of NE recognition. NE Recognizer divides input to sentences and then to words. It then applies part of speech (POS) tagging. POS tagging marks verb, noun and adjectives in sentence. Now Feature Extraction (FEX) tool is used to extract features from the tagged sentence, followed by Sparse Network of Winnows (SNoW) learning architecture which marks entities in text. The NE recognizer used can be trained on domain specific data to improve its accuracy, which in turn leads to overall system accuracy. The complete process of NE recognition is shown in figure 4.4.

The Feature EXtractor (FEX) generates consistent feature indices or feature vectors. FEX takes POS tagged text as input and generates SNoW format representation of text input. It processes input to output (SNoW format) based on user's script describing feature types. FEX also creates and maintains lexicon. It remembers every specific feature encountered and maps feature to index.

SNoW (Sparse Network of Winnows) is a learning architecture framework that is specifically tailored for learning in presence of very large number of features and can be used as a general purpose multi class classifier. It is sparse network of linear units over a Boolean and real valued feature space. Two layer networks are maintained in SNoW's basic architectural instantiation. The input layer is features layer provided by FEX. Nodes in this layer are allocated to features observed in training example. Second layer consists of target nodes. Each target nodes corresponds to a concept (a class label) one wants to represent as a function of input features.

## 4.3 Question Classifier

The question classifier used in the system is Wei Li's [35] classifier, which exploits a combination of language modeling (LM) and regular expression based model. The question classifier first recognizes the entities in question using question classifier then it identifies the class of the question. The two models used in classification are explained below:

- **Regular Expression model:** The basic idea of this model is to determine a question type based on the sentence pattern, which includes the interrogative word, certain sequences of words and some representative terms of particular question classes. Those patterns are defined with regular expressions. For example, a question starting with "*how many*" is very likely to be looking for a number, and a question starting with "*where*" is probably a location question. For a "*what*" question, we can look for some key words to make our decision. For example, "agency", "company" and "university" are related to the organization class. Here are some regular expressions used for certain classes of questions:

  Questions that start with "what" and ask for a person entity:

  *(actor | actresse? | attorne( y | ie) | teacher| ... | senator)s?*

  Questions that start with "*how*" and ask for a length entity:

  *long | short | wide | far | close | big.*(diameter | radius)*

- This approach is very efficient and effective on some question patterns, such as "*how many*" questions. It seldom makes mistakes for this type of question. But there are difficult cases that it can hardly handle. For instance, the answer to a "*who*" question might be a person, an organization, and even a location. Let's take the question "*Who is the largest producer of laptop computers in the world?*" as an example. People can easily tell this is asking for an organization, but regular expression model cannot decide its type just based on the question pattern. Classifier needs additional semantic information, which is not available in the regular expression model. The same problem occurs with the "where" questions. Many "*where*" questions are classified as "*location*" while they are actually "*organization*" questions. The only way to solve this kind of problem is to build a more complete and accurate pattern set, which involves a great deal of human

work. Instead of building a larger and larger question pattern model, classifier uses an automatic and flexible approach: language modeling.

- **Language Modeling:** Classifier uses one language model for each category C of sample questions. When a new question Q comes, classifier calculate the probability $P(Q|C)$ for each C and pick the one with the highest probability. The major advantage of language model over the regular expression model is its flexibility. The regular expression model is composed of hard-coded rules, which need to be modified to handle new cases. The language model, however, can be automatically maintained. And with larger sets of training data, the performance of the language model can be improved. Two language models are used in classifier: unigram and bigram models. The difference between them is the smoothing technique and the combination method. The probability calculation with these models are done with relations given below:

  Unigram:

  $$P(Q \mid C) = P(w_1 \mid C) * P(w_2 \mid C) * \ldots * P(w_n \mid C)$$

  Bigram:

  $$P(Q \mid C) = P(w_1 \mid C) * P(w_2 \mid C, w_1) * \ldots * P(w_n \mid C, w_{n-1})$$

The accuracy of question classifier is shown in table 4.1. It can be seen that combination of regular grammar and language modeling (LM) techniques for question classification is much better than regular grammar based approach.

| Model | | Accuracy (%) |
|---|---|---|
| Regular Expression Model | | 57.57 |
| Experiment 1 | LM only | 81.54 |
| | LM combined with RE model | 85.43 |
| Experiment 2 | LM only | 80.96 |
| | LM combined with RE model | 83.56 |

Table 4.1: Accuracy of question classifier

## 4.4 Search Engine For Text (SEFT)

The retrieval engine used in system is Search Engine For Text (SEFT) [39], which works on locality based similarity heuristics. The locality-based retrieval engine determines the precise location (or, if there is more than one, the precise locations) in the body of each answer document at which the similarity heuristic has triggered. This allows result presentation to be greatly improved, since answer documents can be opened for user inspection at the exact point of maximum similarity, considerably accelerating the speed.

SEFT does not make use of any pre-computed index information and gathers the required query term locations and collection statistics on the fly. To process a query, SEFT proceeds as follows. First, an initialization phase applies case-folding and stemming to the query terms, if needed, and stores them in a lookup data structure. It uses a ternary search tree for this purpose as it provides an efficient implementation of string symbol tables and provides slightly better running times than a more traditional hash table lookup.

Next, a parsing phase reads the text of the source files being searched, breaking the input stream into a sequence of words (including numbers) which are retained, and white-space

and punctuation, which are discarded. As each word is extracted from the text, and case-folded and stemmed as applicable, it is checked for membership within the search structure that holds the query terms. If the word appears in the search tree and is a query term occurrence within the collection, its number, ordinal location and an initialized accumulator are appended to an array. Other information corresponding to the word, such as a filename identifier, line number and file byte offset, are also recorded at this time. On the other hand, if the word does not exist in the query term ternary search tree, it is simply ignored. Once all of the text has been parsed and all occurrences of query terms located, the similarity calculation phase commences.

To reduce computation costs of similarity calculation in SEFT to a tractable level, calculation of the relevance function is restricted to locations at which query terms appear, rather than every location in the collection. This approximation means that all locations returned are query term locations. Both height and spread values are calculated for each query term and placed in a lookup table. Each accumulator is processed in turn, with pointers moving both forward and backward through the array of accumulators, adding influence components to the accumulators of locations that are within the spread of the term (and within the boundaries of the current file, if the input text is spread over multiple files).

Once all query term locations have been processed, a partial sort is used to extract the required number of ranked answers. Finally, for each answer that is to be presented to the user the corresponding file is opened at the relevant byte location and a small window of text extracted and formatted for display.

## 4.5 WordNet

WordNet is a semantic lexicon for the English language. It groups English words into sets of synonyms called *synsets*, provides short, general definitions, and records the various semantic relations between these synonym sets. The purpose is twofold: to produce a combination of dictionary and thesaurus that is more intuitively usable, and to

support automatic text analysis and artificial intelligence applications. WordNet contains more than 118,000 different word forms and more than 90,000 different word senses. Approximately 17% of the words in WordNet are polysemous; approximately 40% have one or more synonyms. WordNet includes the following semantic relations:

- *Synonymy* is WordNet's basic relation, because WordNet uses sets of synonyms (*synsets*) to represent word senses. Synonymy (*syn* same, *onyma* name) is a symmetric relation between word forms.

- *Antonymy* (opposing-name) is also a symmetric semantic relation between word forms, especially important in organizing the meanings of adjectives and adverbs.

- *Hyponymy* (sub-name) and its inverse, *hypernymy* (super-name), are transitive relations between synsets. Because there is usually only one hypernym, this semantic relation organizes the meanings of nouns into a hierarchical structure.

- *Meronymy* (part-name) and its inverse, *holonymy* (whole-name), are complex semantic relations. WordNet distinguishes *component* parts, *substantive* parts, and *member* parts.

- *Troponymy* (manner-name) is for verbs what hyponymy is for nouns, although the resulting hierarchies are much shallower.

- *Entailment* relations between verbs are also coded in WordNet.

Each of these semantic relations is represented by *pointers* between word forms or between synsets. More than 116,000 pointers represent semantic relations between WordNet words and word senses.

## 4.6 Beagle

Beagle [38] is a Linux desktop-independent search service which transparently and unobtrusively indexes your data in real-time and let user search for whatever he/she wants to search. For example:

- Files are immediately indexed when they are created, are re-indexed when they are modified, and are dropped from the index upon deletion.
- E-mails are indexed upon arrival.
- IM conversations are indexed as you chat, a line at a time.
- Web pages are indexed as you view them (with a browser extension).

Beagle supports many different data sources and file formats. It supports search syntax similar to major search engines like Google. Along with required words user can also search for phrases, partial words, excluding words, optional words, file date queries, file extension, file property queries etc. It also performs stemming on every keyword given in search query.

# CHAPTER 5
# IMPLEMENTATION ISSUES

## 5.1 Scalability

Two factors that affect the scalability of the system are:

1. **Data Size:** Data size is a very important scalability factor for the system. The running time of retrieval engine (SEFT) is proportional to the amount of E-Learning documents. As SEFT does not maintain any index of the data, it may take a lot of time to extract passages from the E-Learning dataset. SEFT usually wastes its time on analyzing documents that does not contain query terms. One optimization to improve the performance of the system is to use Beagle first to retrieve names of the files that contain query terms. Then run SEFT only on document names returned by Beagle. Beagle always maintains index of data and gives results (document names) in very less time. This little optimization can provide great improvement when there are large amount of E-Learning documents. Also this optimization will be more effective, when students asks a complex question than, when student asks a general question. Because a general question may contain terms that are present in all the documents.

2. **Number of users:** When using question answering system on a web large number of user request can become bottleneck for system. Because the whole process of answer extraction is very computation intensive. To provide good availability to students, multiple high performance servers will be required. To further shed the load of servers, caching of recently asked question and generally asked can be maintained. Answers to these questions can be given without computation saving a little time of server. Further web optimization techniques may help to improve availability of the system.

## 5.2 Accuracy Issues

The factors that affect the accuracy of the system are:

1. Accuracy of NLP tools used like NE recognizer, Question Classifier is the limiting criterion for overall system accuracy.

2. System requires more templates to extract specific answers to complex questions.

3. Another limiting criterion is unstructured data. Question answering systems can perform much better on structured data. But structuring large amount of data is very difficult task and will take a lot of time.

## 5.3 Efficiency

The retrieval engine used in system is very efficient; it considers every probable location in dataset while extracting passages. It means retrieval engine utilizes the dataset very efficiently. The only issues regarding efficiency in system is while extracting answers from passages. While extracting answers from passages system only uses top three passages for further processing. Even if the probability of getting an answer in top three passages is very high, but there can be a case in which ignored passages may contain answer to question. In this case system will only try to find answer in top three passages. Processing all passages returned by retrieval tool will be very time consuming and improvement will not be significant.

## 5.4 Steps that led to final design and implementation

The earlier implementation of the question answering system was implemented on windows platform using Visual Studio .Net 2003; which lacks the support of automatic question classifier and NE recognizer. The automatic question classifier used in system is Li Wei's classifier which requires a NE recognizer for its working. In its default implementation it used a commercial online NE recognizer, which is not available now. So, NE recognizer from UIUC's which is a non commercial tool available is used in

system. The NE recognizer used in system only works on Linux platform; so whole system is ported to Linux. To make question classifier work, it is modified to use the offline UIUC's NE recognizer. Once system was able to identify the classes of the questions; system structure is modified and Answer Extraction module it added to system and templates for factual questions were designed and implemented. Question classifier is modified to add support for missing classes like REASON and DEFINITION etc. System structure is modified when WSD algorithm is implemented in system. Earlier WSD algorithm uses Google search rather than local dataset search for sense scoring. Using Google for sense scoring is better for open domain question answering, but for closed domain question answering local dataset search performs better.

# CHAPTER 6
# RESULTS AND DISCUSSION

## 6.1 Evaluation of WSD Algorithm

Word Sense Disambiguation (WSD) algorithm is evaluated on Semcor 2.0 [40] files. Semcor is a collection of manually disambiguated files from Brown's corpus using WordNet senses and part of speech information. Domain data is built form the Semcor files, by removing the extra information and storing the files in plain text format.

| Files | Nouns | Disambiguated | Accuracy (%) |
|---|---|---|---|
| br-a01 | 573 | 387 | 67.54 |
| br-a02 | 611 | 389 | 63.67 |
| br-a11 | 582 | 401 | 68.90 |
| br-a12 | 570 | 347 | 60.88 |
| br-a13 | 575 | 373 | 64.87 |
| br-a14 | 542 | 311 | 57.38 |
| br-a15 | 535 | 356 | 66.54 |
| br-b13 | 505 | 283 | 56.04 |
| br-b20 | 458 | 274 | 59.83 |
| br-c01 | 512 | 334 | 65.23 |
| Total | 5463 | 3455 | 63.09 |

Table 6.1: Accuracy of word sense disambiguation

The evaluation was performed on nouns of first 10 files of Semcor. Steps for evaluation are:

- Read sentences from Semcor file one by one.
- Perform our disambiguation on a sentence.
- Compare the disambiguated senses with senses given in Semcor file.

- Perform above operation on next file.

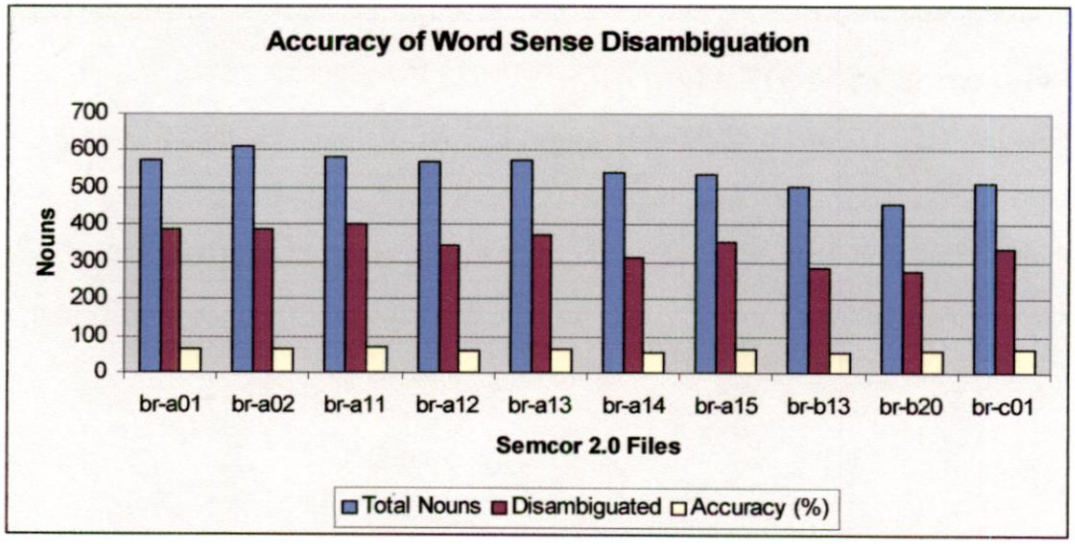



Fig. 6.1 Accuracy of Word Sense Disambiguation

Table 6.1 and figure 6.1 show the results of the experiments performed on WSD algorithm. Total nouns found in first 10 files of Semcor are 5463. The WSD algorithm managed to disambiguate terms with an accuracy of 63.09%. Results have shown that, WSD algorithm has comparable performance to other unsupervised algorithms.

One of the weaknesses of WSD algorithm is the case when for a term in the question; synonyms of correct sense are not present in the local dataset. But local dataset contains term itself and term's another synonym for different sense. For example, if question contains "*...creation of universe...*". The senses of "universe" as extracted from WordNet are given below:

1. **Universe**, existence, nature, creation, world, cosmos, macrocosm
2. **Universe**, cosmos
3. Population, **universe**
4. **Universe**, universe of discourse

Now if our corpus is natural science oriented and every instance of universe with first sense contains "universe" word and some instances of universe with third sense contains word "population". Then even if correct sense is first sense in the question asked and its synonyms are not present in corpus; this algorithm will find third sense more appropriate due to more number of results found for third sense. In this case algorithm must prevent expansion of this term. This problem can be removed with the use of a combined approach of distance based (distance between terms in WordNet) WSD and approach used in algorithm.

## 6.2 Evaluation of Question Answering System

For evaluation of our Question Answering System (QAS), we have taken various books related to operating systems and some text taken from slides. The sample questions were taken from internet, FAQ's. Also collections of questions were gathered from students. We ourselves designed questions to provide more coverage to data.

| | FAQ's | Expert | Naive |
|---|---|---|---|
| Questions | 125 | 35 | 35 |
| Passage 1 | 73 | 13 | 17 |
| Passage 2 | 14 | 7 | 5 |
| Passage 3 | 4 | 2 | 2 |
| Relevant | 14 | 6 | 4 |
| Feedback | 12 | 2 | 4 |
| Failed | 8 | 5 | 3 |

Table 6.2: Accuracy of QAS without WSD

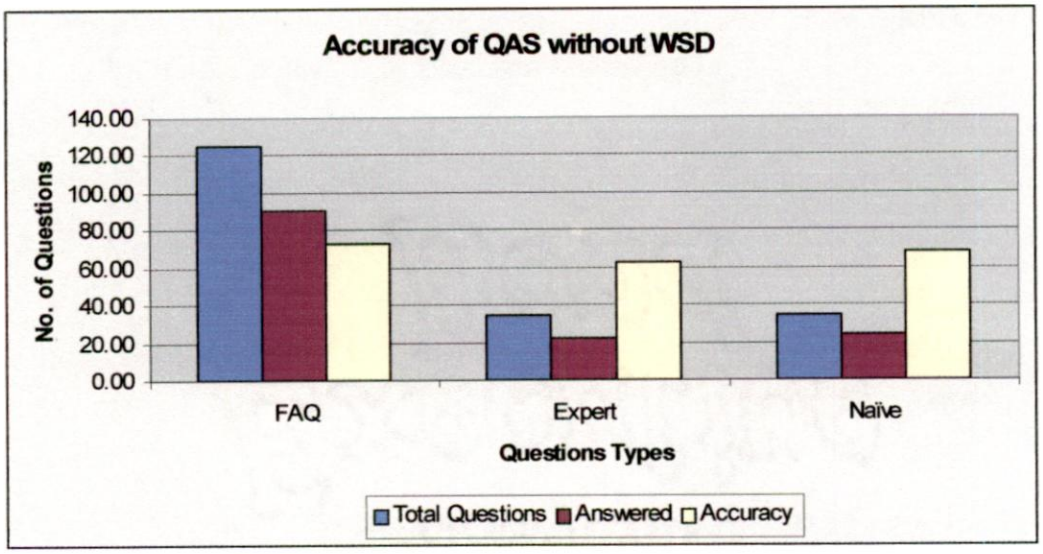



Fig. 6.2 Accuracy of QAS without WSD

In our system, we have retrieved top three passages and found results for them. The percentage of confidence (on average) the system had that the answer was present in first, second and third passage were on average 100%, 85%, 65%. First column shows results for FAQ's and second and third column for questions gathered from students.

Table 6.2 and figure 6.2 show results of experiments on QAS without the usage of WSD algorithm. The accuracy of question answering system without WSD algorithm for FAQ's is 72%, for expert questions is 62% and for naïve questions is 68%. Table 6.3 and figure 6.3 show results of experiments performed on QAS with WSD algorithm. The accuracy of question answering system with WSD algorithm for FAQ's is 74%, for expert questions is 65% and for naïve questions is 71%. The dataset used for experiments is not very large, with large dataset containing a large number of files QAS may perform much better with WSD algorithm. Figure 6.4 shows a snapshot of the system.

| | FAQ's | Expert | Naive |
|---|---|---|---|
| Questions | 125 | 35 | 35 |
| Passage 1 | 74 | 14 | 18 |
| Passage 2 | 14 | 7 | 5 |
| Passage 3 | 5 | 2 | 2 |
| Relevant | 14 | 5 | 5 |
| Feedback | 10 | 2 | 2 |
| Failed | 8 | 5 | 3 |

CENTRAL LIBRARY
Acc. No. G13574
Date 24.6.08
M. U. ROORKEE

Table 6.3: Accuracy of QAS with WSD

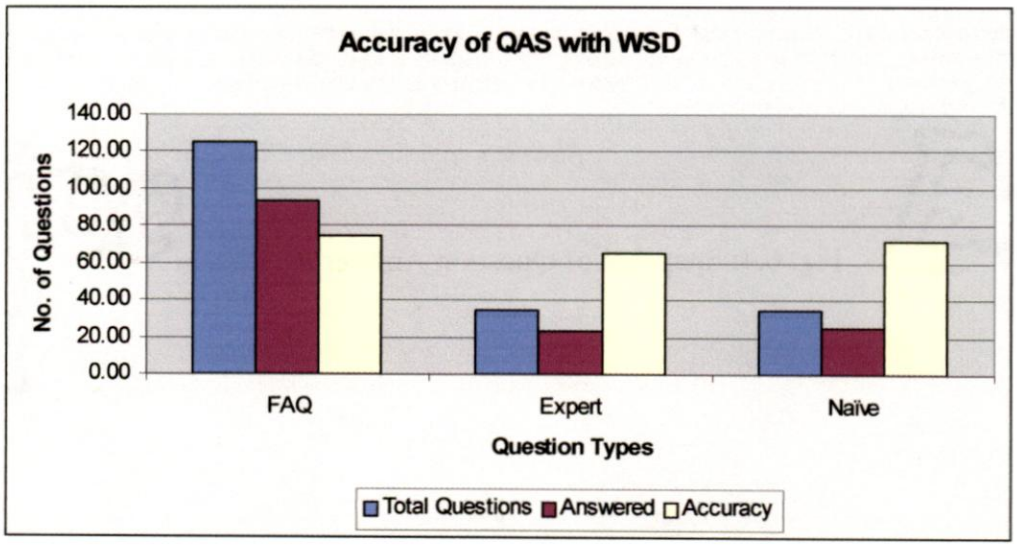



Fig. 6.3: Accuracy of QAS with WSD

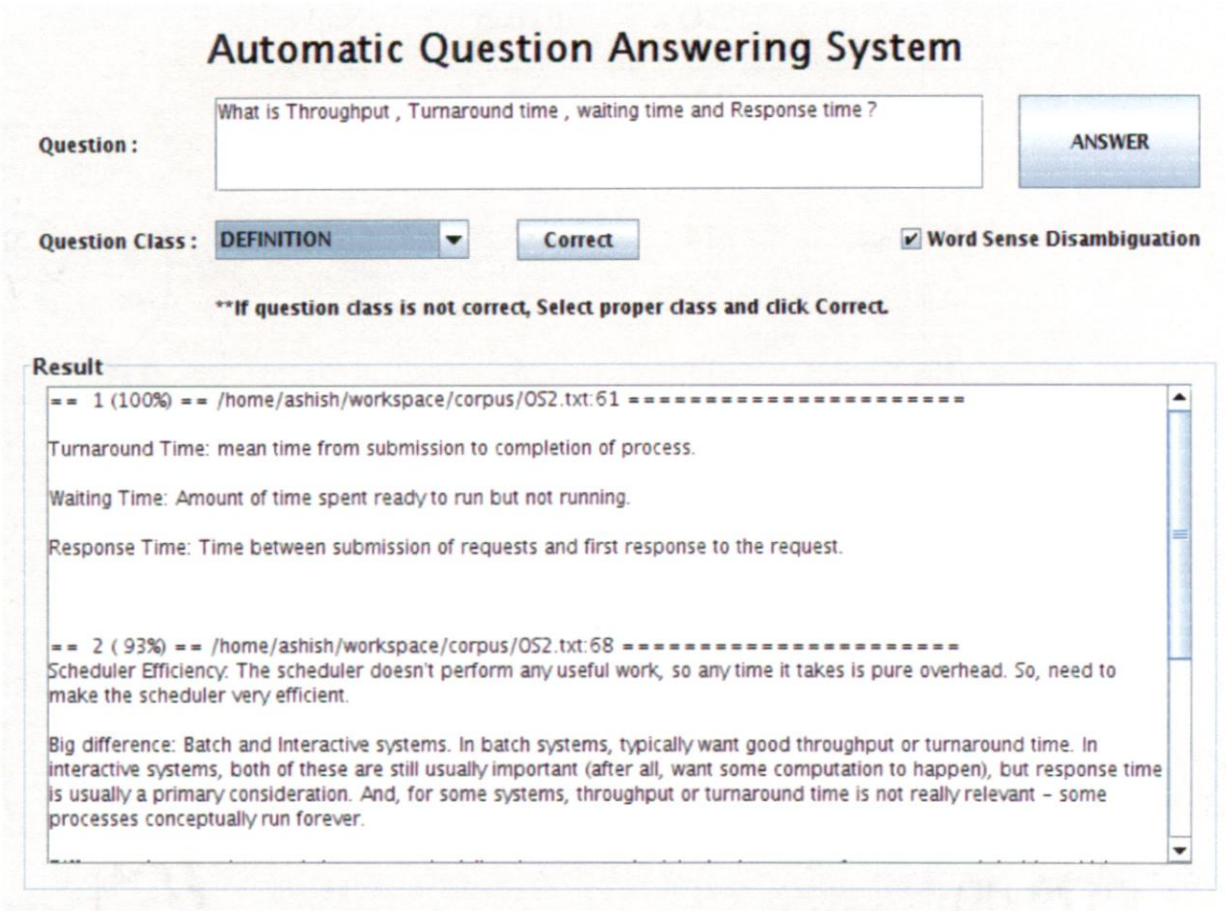



Fig. 6.4: Snapshot of Question Answering System

# CHAPTER 7
# CONCLUSIONS AND FUTURE WORK

## 6.1 Conclusions

In this thesis a new architecture for closed domain question answering system for E-Learning has been presented. This system will help students use E-Learning documents (e-books, slides, journals etc.) effectively. Students can ask question in natural language. Question answering system is enhanced by automatic question classifier, NE recognizer and a novel word sense disambiguation algorithm for closed domain. The system takes advantage of domain knowledge to improve the accuracy of the question answering. Moreover by designing templates for different classes of factual questions quality of answers has been improved. But the unstructured domain data and capabilities of NLP tools limits the performance of system.

Results have shown that, the WSD algorithm which utilizes context and domain data for sense disambiguation has comparable performance to other unsupervised WSD techniques. Benefit of new unsupervised WSD algorithm is no need of classifiers and no need of large sense tagged corpuses. Further WSD can disambiguate domain specific terms easily by sticking to domain and automatically ignoring senses unrelated to domain. But WSD algorithm has some weaknesses where it fails to perform (see Chapter 6). Future work to improve WSD algorithm to combine distance based technique and more utilization of WordNet can resolve these issues.

## 6.2 Suggestions for Future Work

The proposed system gives adequate amount of scope for extension. Some suggestions for the further work are as follows:

1. Till now, templates for factual questions have been designed; more templates for non-factual questions will greatly improve the answer quality.
2. The passage retrieval used only supports plain keyword search, a more sophisticated passage retrieval system which can allow more complex queries (AND-OR queries) will improve efficiency. This will also help in designing of complex templates.
3. WSD algorithm can be modified to use combination of distance based techniques and current technique to improve its accuracy as discussed in results and discussions chapter.
4. New state-of-the-art and fast NE recognizer can reduce runtime of overall system as the NE recognizer used in system is one of the major time consuming component.

# REFERENCES


[1] A. A. Hopgood and A. J. Hirst. "Keeping a distant-education course current through e-learning and contextual assessment", *IEEE Trans. on Education*, vol. 50 no. 1, pp. 85-96, 2007.

[2] T. F. Stafford, "Understanding motivations for Internet use in distant education", *IEEE Trans. on Education*, vol. 48 no. 2, pp. 301-306, 2005.

[3] B. C. Grau, "How to tech basic quantum mechanics to computer scientists and electrical engineers", *IEEE Trans. on Education*, vol. 47 no. 2, pp. 220-226, 2004.

[4] Introduction to Automatic Question Answering Systems, available at: http://en.wikipedia.org/wiki/Question_answering, Last accessed on 19-05-2007.

[5] E. Sneiders and A. Andrenucci, "Automated question answering: review of the main approaches", *Proceedings of the 3rd International Conference on Information Technology and Applications* (ICITA'05), July 4-7, Sydney, Australia, IEEE, Vol. 1, pp.514-519, 2005.

[6] N. Ott, "Aspects of Automatic Generation of SQL statements in Natural Language Query Interace", Information Systems, vol. 17 no. 2, 1991, pp. 21-48, 1992.

[7] E. Sneiders, "Automated Question Answering Using Question Templates that Cover the Conceptual Model of Database", *Proceedings of the 6th International Conference on Applications of Natural Language to Information Systems*, vol. 2553, pp. 235-239, 2002.

[8] G. Salton, J. Allan and C. Buckley, "Approaches to passage retrieval in full text information systems", *Proceedings of SIGIR'93 ACM Press*, NY, USA, pp. 49-58, 1993.

[9] D. Ravichandran and EH. Hovy, "Learning Surface Text Patterns for Question Answering", *Proceedings of 40th annual meeting on ACL*, pp. 41-47, 2001.

[10] C. Clarke et. al., "Web Reinforced Question Answering", *Proceedings of TREC 2001*, Gaithersburg, USA, 2001.

[11] J. Lin, "The Web as a Resource for Question Answering: Perspective and Challenges", *Proceeding of LREC*, 2002.

[12] ASK open domain question answering system available at: http://www.ask.com/, Last accessed on 19-05-2007.

[13] START open domain question answering system available at: http://start.csail.mit.edu/, Last accessed on 19-05-2007.

[14] ANSWERBUS open domain question answering system available at: http://answerbus.coli.uni-saarland.de/index.shtml, Last accessed on 19-05-2007.

[15] W. A. Woods, "Progress in Natural Language Understanding: An Application to Lunar geology", *AFIPS Conference proceedings*, vol. 42, pp. 441-450, 1973.

[16] W. Green, C. Chomsky and K. Laugherty, "BASEBALL: An automatic question answerer", *Proceedings of western joint computer conference*, pp. 219-224, 1986.

[17] C. Dorai, P. Kermani and A. Stewart, "ELM-N: E-Learning Media Navigator", *International Multimedia Conference Proceedings of the ninth ACM international conference on multimedia*, vol. 9, pp. 634-635, 2001.

[18] G. Cha, "COVA: A system for content based distant learning", *Proceedings of international WWW conference (11)*, Honolulu, Hawaii, USA, 2002.

[19] Y. Fu and R. Shen, "GA based CBR approach in Q&A system", *Expert systems with applications*, vol. 26 no. 4, pp. 167-170, 2004.

[20] M. E. S. Mendes, E. Martinez and L. Sacks, "Knowledge based content navigator in e-learning applications", *The London communications symposium*, 2002.

[21] L. Hirschman and R. Gaizauskas, "Natural Language Question Answering: A view from here", *Natural Language Engineering*, vol. 7 issue. 4, pp. 275-300, 2001.

[22] C. T. K. Cody, E. Oren and S. W. Daniel, "Scaling question answering to web", *Proceedings of 10th international conference on WWW*, pp. 150-161, 2001.

[23] B. Katz, "From sentence processing to information access on the World Wide Web", *Natural Language Processing for World Wide Web AAAI spring symposium*, pp. 77-94, 1997.

[24] D. Feng, E. Shaw, J. Kim and E. Hovy, "An intelligent discussion bot for answering student queries threaded discussions", *Proceeding of 11th conference on intelligent user interfaces table of contents*, Australia, pp. 171-177, 2006.

[25] A. Frank, H. U. Krieger, F. Xu, H. Uszkoreit, B. Crysmann, B. Jorg and U. Schafer, "Question answering from structured knowledge sources", *Journal of applied logic, Special issue on questions and answers: theoretical and applied perspective*, 1, pp. 29, 2006.

[26] J. Lin, D. Quan, V. Sinha, K. Bakshi, D. Huynh, B. Katz and D. R. Karger, "The role of context in question answering system", *Proceedings of conference on Human factors in computing systems*, pp. 1006-1007, 2003.

[27] H. T. Ng and H. B. Lee, "Integrating multiple knowledge sources to disambiguate word sense: An exemplar-based approach", *Proceedings of 34th annual meeting of ACL*, pp. 40-47, 1996.

[28] Gina-Anne Levow, "Corpus based techniques for word sense disambiguation", *Technical report AIM, MIT*, 1997.

[29] G. A. Miller, M. Chodorow, S. Landes, C. Leacock and R. G. Thomas, "Using a semantic concordance for sense identification", *Proceedings of the workshop on Human Language Technology*, pp. 240-243, 1994.

[30] G. A. Miller, "WordNet: A lexical database for English", *Communications of the ACM*, vol. 38 issue 11, pp. 39-41, 1995.

[31] P. Resnik, "Disambiguating noun groupings with respect to WORDNET senses", *Proceedings of Third Workshop on Very Large Corpora ACL*, pp. 54-68, 1995.

[32] H. Schutze, "Word Space", *Advances in Neural Information Processing 5*, pp. 895-902, 1992.

[33] D. Temperley, D. Sleator and J. Lafferty, "Parsing English with a link grammar", *Third International workshop on Parsing Technologies*, 1993.

[34] NE Recognizer of UIUC' Cognitive Laboratory, available at: http://l2r.cs.uiuc.edu/~cogcomp/asoftware.php?skey=NE, Last accessed on 19-05-2007.

[35] L. Wei, "Question Classification using Language Modeling", *CIIR Technical Report*, University of Massachusetts, 2002.

[36] J. Gonzalo, F. Verdejo, I. Chugur and J. Ciggaran, "Indexing with WordNet synsets can improve text retrieval", *Proceedings of COOLING/ACL'98 workshop on usage of WordNet for NLP*, Montreal, Canada, pp. 38-44, 1998.

[37] E. M. Voorhees, "Using WordNet to disambiguate word senses for text retrieval", *Proceedings of sixth annual international ACM SIGIR conference on Research and Development in Information Retrieval*, pp. 171-180, 1993.

[38] Beagle Desktop Search tool for Linux available at: http://beagle-project.org/Main_Page, Last accessed on 19-05-2007.

[39] O. D. Kretser and A. Moffat, "Effective document presentation with locality based similarity heuristics", *Proceedings of 22nd annual international ACM SIGIR conference on Research and Development in Information Retrieval*, Sam Francisco, USA, pp. 113-120, 1999.

[40] Semcor manually sense tagged Brown corpus available at: http://www.cs.unt.edu/~rada/downloads.html, Last accessed on 19-05-2007.

----------Main.cpp----------

```
/* STARTING POINT OF THE PROGRAM */

#include <iostream>
#include <string>
#include <fstream>

#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

#include "Common.h"
#include "AnswerExtraction.h"
#include "QClassification.h"
#include "QParser.h"

const char * const QC_SERVER_START = "./LoadNEServer.sh";
const char * const QC_SERVER_STOP = "./KillNEServer.sh";

using namespace std;

int main(int argc, char *argv[])
{

      if(argc != 5 && argc != 6)
      {
            cerr << "Usage: QASystem Q_Filename Ans_filename
            Data_Folder_Path disamb=0|1 class" << endl;
            return -1;
      }

      system(QC_SERVER_START);
      QParser::loadDict();

      string      Q_fname, Ans_fname;
      string      ques, data_path;
      string      cls;
      char        buf[300];
      int         flg=1;

      Q_fname = argv[1];
      Ans_fname = argv[2];
      data_path = argv[3];

      ifstream Q_if(argv[1]);
      if(!Q_if)
            cout << "Error Opening File!" << endl;
      Q_if.getline(buf, 200);
      ques = buf;
      Q_if.close();

      Common::printDelim();
      cout << "Question: " << ques << endl << "Path: " << data_path <<
      endl;
```

```
Common      *obj = Common::getObject();
obj->loadEntities();
obj->loadCommon();

QASystem QA(ques, data_path, Q_fname, Ans_fname);
QClassification   QClassify;

pid_t pid_child;
string q = ques;
if(argc == 5)
{
      // Classificatoin code

      if( (pid_child = fork()) < 0 )
      {
            flg = 0;
            cerr << "Can't create child!" << endl;
            QClassify.classify(q);
            if(QClassify.cls != "")
                  cout << "Question class :" << QClassify.cls <<
                  endl;
            else
                  cerr << "Failed to retrieve question class!" <<
                  endl;
      }
      else
      {
            if( pid_child == 0 )
            {
                  QClassify.classify(q);
                  exit(0);
            }
      }
}
else
{
      cls = argv[5];
      QClassify.setClass(cls);
      cout << "Specified Question class :" <<
      QClassify.getClass() << endl;
}
Common::Disambiguation = (argv[4][(strlen(argv[4])-1)] == '1');
cout << "Disambiguation :" <<
((Common::Disambiguation)?"on":"off") << endl;

QA.runQASystem();

if(argc == 4 && flg)
{
      int status;
      waitpid(pid_child, &status, 0);
      if( QClassify.readClass() != "" )
            cout << "Question class :" << QClassify.cls << endl;
      else
            cerr << "Failed to retrieve question class!" << endl;
```

```
	}

	vector<struct Answer> Answers;

	Common::printDelim();

	AnswerExtraction AnsExt;
	AnsExt.extractAnswer(QClassify.cls, QA, Answers);

	Common::printHeader("FINAL ANSWER");
	if(Answers.size() > 0)
	{
		Common::writeToFile(Ans_fname, Answers);
		for(unsigned int i=0; i<Answers.size(); i++)
			cout << Answers[i].answer << endl << endl;
	}
	else
	{
		Common::writeToFile(Ans_fname, QA.TopPasg);
		for(unsigned int i=0; i<QA.TopPasg.size(); i++)
			cout << QA.TopPasg[i] << endl;
	}

	Common::printDelim();
	delete obj;

	QParser::unloadDict();
	system(QC_SERVER_START);

    return 0;
}
```

**----------QASystem.h----------**

```
/* HEADER FILE FOR QASYSTEM CLASS */

#ifndef QASYSTEM_H_
#define QASYSTEM_H_

#include <string>
#include <vector>

using namespace std;

class QASystem
{
	public:
		QASystem(string Ques, string Data_path, string Qfname,
		string Afname);

		void runQASystem();

	public:

		vector<string> TopPasg;
```

```
            vector<string>    Keywords,
                              Other,
                              Focus,
                              Phrases,
                              Synonyms;

      private:
            string            ques,
                              qfname,
                              afname,
                              data_dir;
};
```

**----------QASystem.cpp----------**

```
/* MAIN CLASS THAT CALLS DIFFERENT PARTS OF QASYSTEM */

#include <vector>
#include <string>

#include "QASystem.h"
#include "Common.h"
#include "SeftSearch.h"
#include "QParser.h"
#include "QueryExpansion.h"

using namespace std;

QASystem::QASystem(string Ques, string Data_path, string Qfname, string
Afname)
{
      putenv("WNHOME=/usr/local/WordNet-3.0" );
      putenv("WNSEARCHDIR =/usr/local/WordNet-3.0/dict");

      ques = Ques;
      data_dir = Data_path;
      qfname = Qfname;
      afname = Afname;
}

void QASystem::runQASystem()
{
      vector<string>    allkeywords;
      string                  line = ques;
      string                  temp = "QUESTION :";
      temp.append(line);

      Common::printHeader(temp);
      //cout << "\Question :" << line << endl;

      QParser qparse;
      qparse.Parse(line, Focus, Keywords, Other, Phrases);

      cout << "\nFocus :" ;
      Common::printVector(Focus);
```

```
        cout << "\nDomain Keywords :";
        Common::printVector(Keywords);
        cout << "\nPhrases :";
        for(unsigned int f=0; f<Phrases.size(); ++f)
              cout << "'" << Phrases[f] << "' ";
        cout << endl;
        //Common::printVector(Phrases);
        cout << "\nOther Keywords :";
        Common::printVector(Other);
        cout << endl;

        for(unsigned int f=0; f<Focus.size(); ++f)
              allkeywords.push_back(Focus[f]);

        for(unsigned int f=0; f<Keywords.size(); ++f)
              if(!Common::isPresent(allkeywords, Keywords[f]))
                    allkeywords.push_back(Keywords[f]);

        for(unsigned int f=0; f<Other.size(); ++f)
              if(!Common::isPresent(allkeywords, Other[f]))
                    allkeywords.push_back(Other[f]);

        cout << "Running Query Expansion...." << endl;
        vector<string>    Expansion;
        QueryExpansion    QE;
        QE.ExpandTerms(allkeywords, Expansion);

        for(unsigned int f=0; f<Phrases.size(); ++f)
              if(!Common::isPresent(allkeywords, Phrases[f]))
                    allkeywords.push_back(Phrases[f]);

        cout << "Removing pos's..." << endl;
        for(unsigned int f=0; f<allkeywords.size(); ++f)
              qparse.removePos(allkeywords[f]);

        for(unsigned int f=0; f<Expansion.size(); ++f)
                    allkeywords.push_back(Expansion[f]);

        cout << "Running Search..." << endl;
        SeftSearch  srchObj;
        srchObj.search(allkeywords, data_dir, TopPasg);

        cout << "Passages Retrieved: " << endl;
        for(unsigned int i=0; i<TopPasg.size(); i++)
              cout << TopPasg[i] << endl << endl;

}
```

**----------QParser.h----------**

```
/*    HEADER FILE FOR PARSER */

#endif /*QASYSTEM_H_*/

#ifndef QPARSER_H_
```

```
#define QPARSER_H_

#include <string>
#include <vector>

#include "link-includes.h"

using namespace std;

class QParser
{
    public:
        int Parse(const string input, vector<string> &Focus,
        vector<string> &Keywords, vector<string> &Other,
        vector<string> &Phrases);

        static void loadDict();
        static void unloadDict();

        static int  removePos(string &str);

    private:
        void print_words_with_prep_phrases_marked(CNode *n,
        vector<string> &Phrase, int flag=0, int stack=0, bool
        first=true);

        bool addPos(Sentence sent, Linkage linkage, int index,
        string &ret);

    public:
        Sentence                    sent;

    private:
        static Dictionary       dict;
        static Parse_Options    opts;

};

Dictionary      QParser::dict;
Parse_Options   QParser::opts;

#endif /*QPARSER_H_*/

-------------------------QParse.cpp----------------------

/*    PARSER FILE INCLUDES ROUTINES FOR PARSING
      PART OF SPEECH ADDTION AND REMOVAL, PHRASE DETECTION
*/

#include <vector>
#include <string>

#include "QParser.h"
#include "Common.h"
#include "wn.h"
```

```
#include "link-includes.h"

using namespace std;


bool QParser::addPos(Sentence sent, Linkage linkage, int index, string
&ret)
{
      bool  retval = false;
      char  *word = sentence_get_word(sent, index);
      char  *temp = new char[strlen(word) + 2];
      strcpy(temp, word);
      if(!Common::isCommon(word))
      {
            char  *wordl = linkage_get_word(linkage, index);
            int   len = strlen(wordl);
            if(wordl[len-2] == '.')
            {
                  switch(wordl[len-1])
                  {
                        case 'a':   strcat(temp, "3");      break;
                        case 'v':   strcat(temp, "2");      break;
                        case 'n':   strcat(temp, "1");      break;
                        default:    strcat(temp, "1");      break;
                  }
            }
            else
            {
                  strcat(temp, "1");
            }
            ret = temp;
            //cout << "word :" << ret << endl;
            retval = true;
      }
      delete []temp;
      return retval;
}

int QParser::removePos(string &str)
{
      int pos = str[str.length()-1] - '0';
      str.erase(str.length()-1);
      return pos;
}

int QParser::Parse(const string input, vector<string> &Focus,
vector<string> &Keywords,
                                                      vector<string>
&Other, vector<string> &Phrases)
{
      char  *line = new char[input.length() + 2];

      strcpy(line , input.c_str());

      sent = sentence_create(line, QParser::dict);
```

```
if (sent == NULL)
{
      cout << "Error creating sentence for parse!" << endl;
      return -1;
}

int nLinkages = sentence_parse(sent, QParser::opts);
cout << "No. of linkages :" << nLinkages << endl;

for(int linkageno=0; linkageno < nLinkages; ++linkageno)
{
      Linkage linkage = linkage_create(linkageno, sent,
                              QParser::opts);
      linkage_compute_union(linkage);
      linkage_set_current_sublinkage(linkage,
      linkage_get_num_sublinkages(linkage)-1);
      // Printing linkage structure
      char  *diagram = linkage_print_diagram(linkage);
      cout << diagram << endl;
      string_delete(diagram);

      int nLinks = linkage_get_num_links(linkage);
      for(int linkno=0; linkno<nLinks; ++linkno)
      {
            char *llabel = linkage_get_link_label(linkage,
                        linkno);
            //cout << "Link Label :" << llabel << endl;
            if(   llabel[0] == 'A' || llabel[0] == 'B' ||
            llabel[0] == 'D' || llabel[0] == 'G' ||
                  llabel[0] == 'J' || llabel[0] == 'M' ||
            llabel[0] == 'R' || llabel[0] == 'S')
            {
                  string ret;
                  if(addPos(sent, linkage,
                  linkage_get_link_lword(linkage, linkno), ret)
                  && !Common::isPresent(Focus, ret))
                        Focus.push_back(ret);
                  if(addPos(sent, linkage,
                  linkage_get_link_rword(linkage, linkno), ret)
                  && !Common::isPresent(Focus, ret))
                        Focus.push_back(ret);
            }
      }

      CNode *cn = linkage_constituent_tree(linkage);
      print_words_with_prep_phrases_marked(cn, Phrases);
      linkage_free_constituent_tree(cn);

      linkage_delete(linkage);
}

int l = sentence_length(sent);
for(int z=1; z<l-1; z++)
{
      char  *buf = sentence_get_word(sent, z);
```

```
            string      temp = buf;
            if(!Common::isCommon(buf) && !Common::isPresent(Keywords,
            temp))
            {
                  if(Common::isEntity(buf))
                  {
                        Keywords.push_back(temp + "1");
                  }
            }
      }

      sentence_delete(sent);

      return 0;
}

// METHOD TO EXTRACT PHRASES
// flag and stack defaults to 0
void QParser::print_words_with_prep_phrases_marked(CNode *n,
vector<string> &Phrases, int flag, int stack, bool first)
{
    CNode   *m;
    string  temp, buf;

    if(first)
      Phrases.push_back("");

    if (n == NULL)
    {
      Phrases.pop_back();
      return;
    }

    if (strcmp(n->label, "NP") == 0)
    {
      flag=1;
      stack++;
    }

    for (m = n->child; m != NULL; m = m->next)
      {
       if (m->child == NULL)
       {
            if(flag)
            {
                  temp = Phrases[Phrases.size()-1];
                  buf = m->label;
                  temp = temp + buf + " ";
                  Phrases.pop_back();
                  Phrases.push_back(temp);
            }

       }
       else
       {
```

```
                print_words_with_prep_phrases_marked(m, Phrases, flag,
                stack, false);
          }
      }
      if (strcmp(n->label, "NP")==0)
        {
                stack--;
                if(!stack)
                {
                        flag=0;
                        temp = Phrases[Phrases.size()-1];
                        temp[temp.length()-1] = '1';
                        Phrases.pop_back();
                        if(!Common::isPresent(Phrases, temp))
                                Phrases.push_back(temp);
                        Phrases.push_back("");
                }
        }
        if(first)
                Phrases.pop_back();
}

void QParser::loadDict()
{
      QParser::opts = parse_options_create();
      QParser::dict = dictionary_create("4.0.dict", "4.0.knowledge",
                        "4.0.constituent-knowledge", "4.0.affix");
}

void QParser::unloadDict()
{
      dictionary_delete(QParser::dict);
      parse_options_delete(QParser::opts);
}
```

**----------QClassification.h----------**

```
/* HEADER FILE FOR QUESTION CLASSIFIER CLASS */

#include <string>
#include <vector>

const char * const QC_CMD = "./QClassify.sh";
const char * const QC_FILE = "../QC/tques.txt";
const char * const QC_CLS = "../QC/tclass.txt";

using namespace std;

class QClassification
{
      public:
            QClassification();

            string classify(string ques);
            string readClass();
```

```
            void setClass(string clss);
            string getClass() {return cls;}

      public:
            string cls;
};
```

**----------QClassification.cpp----------**

```
/* CALLS QUESTION CLASSIFIER */

#include <iostream>
#include <fstream>
#include <sstream>

#include "QClassification.h"
#include "Common.h"

using namespace std;

QClassification::QClassification()
{
      cls = "";
}

string QClassification::readClass()
{
      Common::readFromFile(QC_CLS, cls);

      return cls;
}

string QClassification::classify(string ques)
{
      cls = "";

      if( Common::writeToFile(QC_FILE, ques) == -1)
      {
            return cls;
      }

      system(QC_CMD);

      Common::readFromFile(QC_CLS, cls);

      return cls;
}

void QClassification::setClass(string clss)
{
      cls = clss;
}
```

**----------NERecognizer.h----------**

```
/* HEADER FILE FOR NE RECOGNIZER CLASS */

#ifndef NERECOGNIZER_H_
#define NERECOGNIZER_H_

#include <string>

using namespace std;

const char * const QC_NE = "./NE.sh";
const char * const QC_LINE = "line.txt";
const char * const QC_NELINE = "NEline.txt";

class NERecognizer
{
      public:
            string runNERecognizer(string line);
};

#endif /*NERECOGNIZER_H_*/
```

**----------NERcognizer.cpp----------**

```
/* NE RECOGNIZER CLASS CALL MAIN NE RECOGNIZER */

#include <string>

#include "NERecognizer.h"
#include "Common.h"

using namespace std;

string NERecognizer::runNERecognizer(string line)
{
      string NEline = "";

      if ( Common::writeToFile(QC_LINE, line) == -1)
      {
            cout << "Error using NE!" << endl;
            return NULL;
      }

      system(QC_NE);

      if( Common::readFromFile(QC_NELINE, NEline) == -1)
      {
            cout << "Error using NE!" << endl;
            return NULL;
      }

      return NEline;
}
```

**----------QueryExpansion.h----------**

```
/* HEADER FILE FOR QUERY EXPANSION */

#ifndef QUERYEXPANSION_H_
#define QUERYEXPANSION_H_

#include <vector>
#include <string>

using namespace std;

class QueryExpansion
{
      public:
            int ExpandTerms(vector<string> &Terms, vector<string>
            &Expansion);

      private:
            long long int getHits(const string &SearchString,
            vector<string> &args);
            int FindSimilar(char *wd, int pos, vector<string> &Words);
            void exploreWord(char *word, int pos);
            int withoutWSD(vector<string> &Terms, vector<string>
            &Expansion);
            int withWSD(vector<string> &Terms, vector<string>
            &Expansion);
};

#endif /*QUERYEXPANSION_H_*/
```

**----------QueryExpansion.cpp----------**

```
/*    QUERY EXPANSION CLASS INCLUDES MOTHODS FOR
      WORD SENSE DISABIGUATION, BEAGLE SEARCH, WITHOUT WSD EXPANSION
      TO EXPLORE A WORD IN WORDNET
*/

#include <iostream>
#include <vector>
#include <string>

#include "pstream.h"
#include "QueryExpansion.h"
#include "wn.h"
#include "QParser.h"
#include "Common.h"

using namespace std;

int QueryExpansion::withoutWSD(vector<string> &Terms, vector<string>
&Expansion)
{
      for(unsigned int i=0; i<Terms.size(); ++i)
      {
            string      term1(Terms[i], 0, Terms[i].length()-1);
            int         pos1 = Terms[i][Terms[i].length()-1] - '0';
```

```
            char  *str = new char[Terms[i].length()];
            strcpy(str, term1.c_str());
            FindSimilar(str, pos1, Expansion);
            delete []str;
      }
      for(unsigned int i=0; i<Expansion.size(); ++i)
            Common::replaceChar(Expansion[i], '_', ' ');
      return 1;
}

int QueryExpansion::withWSD(vector<string> &Terms, vector<string>
&Expansion)
{
      if(wninit() == -1)
      {
            cout << "Wordnet initialization failed!" << endl;
            return -1;
      }
      exploreWord("time", 1);
      cout << "Running WSD algorithm..." << endl;
      vector<string>    *maxSynonyms = new
                  vector<string>[Terms.size()];
      for(unsigned int i=0; i<Terms.size(); ++i)
      {
            string s = Terms[i];
            QParser::removePos(s);
            maxSynonyms[i].push_back(s);
      }
      for(unsigned int i=0; i<Terms.size(); ++i)
      {
            char  *term1 = new char[Terms[i].length()+1];
            int         pos1 = Terms[i][Terms[i].length()-1] - '0';
            string      curTerm = Terms[i];
            Common::replaceChar(curTerm, ' ', '_');
            strcpy(term1, curTerm.c_str());
            term1[strlen(term1)-1] = '\0';
            curTerm = Terms[i];
            curTerm[curTerm.length()-1] = '\0';

            cout << "Term (" << curTerm << ") :";
            SynsetPtr syn = findtheinfo_ds(term1, pos1, SYNS,
            ALLSENSES);
            vector<string>    args;
            string      QueryString = "";
            string      quote = "\"";
            for(unsigned int j=0; j<Terms.size(); ++j)
            {
                  if(j == i) continue;
                  for(unsigned int k=0; k<maxSynonyms[j].size()-1; ++k)
                  {
                        args.push_back(quote + maxSynonyms[j][k] +
                        quote);
                        args.push_back("OR");
                        QueryString += quote + maxSynonyms[j][k] +
                        quote + " OR ";
```

```
        }
        args.push_back(quote +
        maxSynonyms[j][maxSynonyms[j].size()-1] + quote);
        QueryString += quote +
        maxSynonyms[j][maxSynonyms[j].size()-1] + quote + "
        ";
}
cout << "SearchString--:" << QueryString << endl;
long long int     maxScore = 0;
for(SynsetPtr synPtr=syn; synPtr; )//synPtr=synPtr->nextss)
{
        int               argCount = 0;
        vector<string>    Synonyms;
        long long int     totalHits = 0;
        long long int     score = 0;
        string            SearchString = QueryString;
        string            tempStr;

        cout << "WN Sense (" << *synPtr->wnsns << ") :";
        for(int wi=0; wi<synPtr->wcount-1; ++wi)
        {
                tempStr = synPtr->words[wi];
                Common::replaceChar(tempStr, '_', ' ');
                if(!Common::compare(tempStr.c_str(),
                curTerm.c_str()))
                        continue;
                args.push_back(quote + tempStr + quote);
                args.push_back("OR");
                argCount += 2;
                SearchString += quote + tempStr + quote + " OR
                ";
                Synonyms.push_back(tempStr);
                cout << tempStr << "-";
        }
        tempStr = synPtr->words[synPtr->wcount-1];
        Common::replaceChar(tempStr, '_', ' ');
        if(Common::compare(tempStr.c_str(), curTerm.c_str()))
        {
                args.push_back(quote + tempStr + quote);
                argCount++;
                SearchString += quote + tempStr + quote;
                Synonyms.push_back(tempStr);
                cout << tempStr << endl;
        }else if(argCount > 0)
        {
                argCount--;
                args.pop_back();
        }
        if(argCount > 0)
        {
                totalHits = getHits(SearchString, args);
                score = totalHits;
                if(score > maxScore)
                {
```

```
                                    cout << "Score :" << score << "Max Score
                                    :" << maxScore << endl;
                                    maxSynonyms[i].clear();
                                    for(unsigned int j=0; j<Synonyms.size();
                                    ++j)

                                    maxSynonyms[i].push_back(Synonyms[j]);
                                    maxScore = score;
                              }
                              while(argCount--)
                                    args.pop_back();
                        }
                        SynsetPtr t = synPtr;
                        synPtr = synPtr->nextss;
                        free_synset(t);
                  }
                  maxSynonyms[i].push_back(curTerm);
                  cout << endl;
                  delete []term1;
            }
            for(unsigned int j=0; j<Terms.size(); ++j)
                  for(unsigned int i=0; i<maxSynonyms[j].size(); ++i)
                        Expansion.push_back(maxSynonyms[j][i]);
            delete []maxSynonyms;
            return 1;
}

int QueryExpansion::ExpandTerms(vector<string> &Terms, vector<string>
&Expansion)
{
      return (Common::Disambiguation) ? withWSD(Terms, Expansion) :
      withoutWSD(Terms, Expansion);
}

long long int QueryExpansion::getHits(const string &SearchString,
vector<string> &args)
{
      const string      beaglesearch = "beagle-query";
      long long int     totalHits = -1;
      string                  temp = beaglesearch;
      for(unsigned int i=0; i<args.size(); ++i)
            temp += " " + args[i];
      cout << "Beagle Search Command :" << temp << endl;
      redi::ipstream    f(beaglesearch.c_str(), args);
      const int         LINESIZE = 4096;
      char              line[LINESIZE];
      while (!f.eof())
      {
            f.getline(line, LINESIZE);
            totalHits++;
      }
      f.close();
      cout << "Total Results Found :" << totalHits << endl;
      return totalHits;
}
```

```
inline void printWords(SynsetPtr synPtr)
{
      for(int wi=0; wi<synPtr->wcount; ++wi)
            cout << "'" << synPtr->words[wi] << "' ";
}

void QueryExpansion::exploreWord(char *word, int pos)
{
      wninit();
      SynsetPtr   syn = findtheinfo_ds(word, pos, SYNS, ALLSENSES);
      cout << "EXPLORING WORD :" << word << endl;
      int i=1;
      for(SynsetPtr synPtr=syn; synPtr; ++i)
      {
            cout << "Sense " << i << ":" << endl;
            cout << "Syns :";
            printWords(synPtr);
            cout << endl;

            int j=1;
            for(SynsetPtr ptr = synPtr->ptrlist; ptr; ++j)
            {
                  cout << "PtrList " << j << ":" << endl;
                  cout << "Syns :";
                  printWords(ptr);
                  cout << endl;

                  SynsetPtr t = ptr;
                  ptr = ptr->ptrlist;
                  free_synset(t);
            }
            j=1;
            for(SynsetPtr ptr = synPtr->nextform; ptr; ++j)
            {
                  cout << "NextForm " << j << ":" << endl;
                  cout << "Syns :";
                  printWords(ptr);
                  cout << endl;

                  SynsetPtr t = ptr;
                  ptr = ptr->nextform;
                  free_synset(t);
            }

            SynsetPtr t = synPtr;
            synPtr = synPtr->nextss;
            free_synset(t);
      }
}

int QueryExpansion::FindSimilar(char *wd, int pos, vector<string>
&Words)
{
      int         count=0;
```

```
Synset              *next;
Synset              *ptr;
string              temp;

wninit();

SynsetPtr t = findtheinfo_ds(wd, pos, SYNS, ALLSENSES);
if(t)
{
      for(int k=0;k<t->wcount;k++)
      {
            if(Common::compare(wd,t->words[k]))
            {
                  temp = t->words[k];
                  Words.push_back(temp);
                  count++;
                  if( count > 4 )
                        return count;
            }
      }
      next=t->nextss;

      while(next)
      {
            for(int k=0;k<next->wcount;k++)
            {
                  if(Common::compare(wd,next->words[k]))
                  {
                        temp = next->words[k];
                        Words.push_back(temp);
                        count++;
                        if(count>4)
                              return count;
                  }
            }
            ptr=next->ptrlist;
            while(ptr)
            {
                  for(int k=0;k<ptr->wcount;k++)
                  {
                        if(Common::compare(wd,ptr->words[k]))
                        {
                              temp = ptr->words[k];
                              Words.push_back(temp);
                              count++;
                              if(count>4)
                                    return count;
                        }
                  }
                  ptr=ptr->ptrlist;
            }
            next=next->nextss;
      }
}
return count;
```

```
}
```

**----------SeftSearch.h----------**

```
/* HEADER FILE FOR PASSAGE RETRIEVAL */

#pragma once

#include <vector>
#include <string>

using namespace std;

class SeftSearch
{
      private:
            int   window_size;
            int   window_num;

      public:
            SeftSearch()
            {
                  window_size = 7;
                  window_num = 3;
            }

            void setWindowSize(int size);
            void setWindowNum(int num);
            void search(vector<string> &allkeywords, string dir,
            vector<string> &passages);
            void getPassages(string command, string dir, vector<string>
            &passages, vector<string> &args);
            string buildCommand(vector<string> &allkeywords, string
            dir, vector<string> &args);

};
```

**----------SeftSearch.cpp----------**

```
/*    SEFTSEARCH CLASS CALLS MAIN PASSAGE RETIEVAL (SEFT), SETS VARIOUS
      PARAMETERS FOR SEARCH */

#include <iostream>
#include <sstream>
#include <fstream>

#include "pstream.h"
#include "SeftSearch.h"
#include "Common.h"

using namespace std;

void SeftSearch::setWindowSize(int size)
{
      window_size = size;
```

```
}

void SeftSearch::setWindowNum(int num)
{
      window_num = num;
}

void SeftSearch::search(vector<string> &allkeywords, string dir,
vector<string> &passages)
{
      if(allkeywords.size() == 0)
      {
            cout << "No Keywords to seach!" << endl;
            return;
      }

      vector<string>    args;
      string command = buildCommand(allkeywords, dir, args);
      string temp = command;
      for(unsigned int i=0; i<args.size(); ++i)
            temp += " " + args[i];
      cout << "Running SEFT command :" << temp << endl;
      getPassages(command, dir, passages, args);
}

string SeftSearch::buildCommand(vector<string> &allkeywords, string
dir, vector<string> &args)
{
      string command = "../seft/seft"; //seft path
      std::stringstream numstr;
      string                        arg;

      //Turn off highlights
      arg = "-x";
      args.push_back(arg);

      //Turn on case folding and stemming
      arg = "-s";
      args.push_back(arg);
      arg = "2";
      args.push_back(arg);

      arg = "-w";
      args.push_back(arg);
      numstr << this->window_size;
      arg = "";
      arg += numstr.str();
      args.push_back(arg);

      numstr.str("");

      arg = "-m";
      args.push_back(arg);
      numstr << this->window_num;
      arg = "";
```

```
	arg += numstr.str();
	args.push_back(arg);

	string keywords = allkeywords[0];
	for(unsigned int i=1; i<allkeywords.size(); i++)
		keywords += " " + allkeywords[i];

	arg = "\"";
	arg += keywords;
	arg += "\"";
	args.push_back(arg);

	Common::getDataFileNames(dir, args);

	return command;
}

void SeftSearch::getPassages(string command, string dir, vector<string>
&passages, vector<string> &args)
{
	redi::ipstream	f(command, args);
	const int	LINESIZE = 4096;
	char		line[LINESIZE];
	string		passage = "";
	string		phdr = "== " + dir;

	f.getline(line, LINESIZE);
	while (!f.eof())
	{
		if(strstr(line, phdr.c_str()))
		{
			if(passage.size() > 0)
			{
				passages.push_back(passage);
				passage = "";
			}
		}
		passage += line;
		passage += "\n";
		f.getline(line, LINESIZE);
	}
	if(passage.size() > 0)
	{
		passages.push_back(passage);
		passage = "";
	}
	f.close();
}
```

**----------AnswerExtraction.h----------**

```
/* HEADER FILE FOR ANSWER EXTRACTION CLASS */

#ifndef ANSWEREXTRACTION_H_
#define ANSWEREXTRACTION_H_
```

```
#include <vector>
#include <string>

#include "QASystem.h"

using namespace std;

class AnswerExtraction
{
      public:
            int extractAnswer(string clss, QASystem &QA, vector<struct
            Answer> &Answers);

      private:
            bool isNumber(string clss);
            int checkKeywords(const vector<string> &qFocus, const
            vector<string> &qKeywords, const vector<string> &qOther,
            const vector<string> &qPhrases, const vector<string>
            &aFocus, const vector<string> &aKeywords, const
            vector<string> &aOther, const vector<string> &aPhrases);

            int checkKeyword(const string Keyword, const vector<string>
            &Focus, const vector<string> &Other, const vector<string>
            &Phrases);

};

#endif /*ANSWEREXTRACTION_H_*/
```

**----------AnswerExtraction.cpp----------**

```
/* ANSWER EXTRACTION CLASS INCLUDES TEMPLATES TO EXTRACT ANSWER */

#include <vector>
#include <string>

#include "AnswerExtraction.h"
#include "Common.h"
#include "QASystem.h"
#include "NERecognizer.h"
#include "QParser.h"

using namespace std;

int AnswerExtraction::extractAnswer(string clss, QASystem &QA,
vector<struct Answer> &Answers)
{
      QParser                 qparse;
      vector<string>          Keywords,
                              Other,
                              Focus,
                              Phrases;

      Matches                 matches;
```

```
string                  pattern;
struct Answer           ans;

Common::printDelim();
cout << "Extracting Answers......" << endl;

if( isNumber(clss) )
{
      pattern = "-?[0-9]+.?[0-9]*.?";

      for(unsigned int k=0; k<QA.TopPasg.size(); k++)
      {
            string text = QA.TopPasg[k];
            text = text.substr(text.find_first_of("\n"));

            string lines[10];
            int nlines = Common::getLines(text, lines, 10);
            if( nlines < 0)
            {
                  return -1;
            }
            for( int x=0; x<nlines; x++ )
            {
                  if(Common::trim(lines[x]).size() < 2)
                        continue;

                  string str1 = "Analyzing line :";
                  str1.append(lines[x]);
                  Common::printDelim();
                  Common::printHeader(str1);
                  //cout << "Line :" << lines[x] << endl;

                  int ret = Common::searchPattern(pattern,
                  lines[x], matches);
/*                for(int xx = 0; lines[x].length(); xx++)
                  {
                        if(lines[x][xx] >= '0' && lines[x][xx] <=
                        '9')
                        {
                              ret = 1;
                              break;
                        }
                  }
*/
                  if(ret == -1 || ret == -2)
                  {
                        cout << "No Match found!" << endl;
                        continue;
                  }

                  qparse.Parse(lines[x], Focus, Keywords, Other,
                  Phrases);

                  cout << "Checking Focus similarity...." <<
                  endl;
```

```
                        ans.score = checkKeywords(Focus, Keywords,
                        Other, Phrases, QA.Focus, QA.Keywords,
                        QA.Other, QA.Phrases);
                        int tot = Focus.size() + Keywords.size() +
                        Other.size() + Phrases.size();
                        if(tot > 0)
                              ans.score = ans.score / tot;

                        if( ans.score != 0.0)
                        {
                              ans.answer = lines[x];
                              //Answers.push_back(ans);
                              Common::insertAnswer(Answers, ans);

                              cout << "Added Answer :" << ans.answer <<
                              endl;
                              cout << "Score :" << ans.score << endl;
                              cout << "Total Answers :" <<
                              Answers.size() << endl;
                        }

//                      cout << "\nParse Focus:";
//                      Common::printVector(Focus);
                        Focus.clear();

//                      cout << "\nParse Keywords: ";
//                      Common::printVector(Keywords);
                        Keywords.clear();

//                      cout << "\nParse Other Keywords:---";
//                      Common::printVector(Other);
                        Other.clear();

//                      cout << "\nParse Phrases:---";
//                      Common::printVector(Phrases);
                        Phrases.clear();

                        cout << endl ;

                  }
            }
            return 0;
      }

      if( clss == "PERSON" || clss == "LOCATION" || clss ==
      "ORGANIZATION" )
      {
            for(unsigned int k=0; k<QA.TopPasg.size(); k++)
            {
                  string text = QA.TopPasg[k];
                  text = text.substr(text.find_first_of("\n"));

                  string lines[10];
                  int nlines = Common::getLines(text, lines, 10);
```

```
if( nlines < 0)
      return -1;
for( int x=0; x<nlines; x++ )
{
      if(Common::trim(lines[x]).size() < 2)
            continue;

      string strl = "Analyzing line :";
      strl.append(lines[x]);
      Common::printDelim();
      Common::printHeader(strl);
//    cout << "Line :" << lines[x] << endl;

      qparse.Parse(lines[x], Focus, Keywords, Other,
      Phrases);

      cout << "Checking Focus similarity...." <<
      endl;

      ans.score = checkKeywords(Focus, Keywords,
      Other, Phrases, QA.Focus, QA.Keywords,
      QA.Other, QA.Phrases);
      int tot = Focus.size() + Keywords.size() +
      Other.size() + Phrases.size();
      if(tot > 0)
            ans.score = ans.score / tot;

      if( ans.score != 0.0)
      {
            NERecognizer NER;
            string NEline =
            NER.runNERecognizer(lines[x]);
            if(NEline == "") return -1;

            cout << "NE output:" << NEline << endl;

            if( clss == "PERSON" )
                  pattern = "[PER";
            if( clss == "LOCATION" )
                  pattern = "[LOC";
            if( clss == "ORGANIZATION" )
                  pattern = "[ORG";

            unsigned int ret = NEline.find(pattern,
0);
            while(ret != string::npos)
            {
                  unsigned int st = ret;
                  ret = NEline.find("]", ret);
                  if(ret == string::npos)
                  {
                        ret = st + 4;
                  }
                  else
                  {
```

```
                                        ans.answer =
                                        NEline.substr(st+4, ret-
                                        (st+4));
                                        //Answers.push_back( ans );
                                        Common::insertAnswer(Answers,
                                        ans);
                                        cout << "Added answer " <<
                                        Answers.size() << " :" <<
                                        ans.answer << endl;
                                        cout << "Score :" <<
                                        ans.score << endl;
                                }
                                ret = NEline.find(pattern, ret);
                        }
                }

//                      cout << "\nParse Focus:";
//                      Common::printVector(Focus);
                        Focus.clear();

//                      cout << "\nParse Keywords: ";
//                      Common::printVector(Keywords);
                        Keywords.clear();

//                      cout << "\nParse Other Keywords:---";
//                      Common::printVector(Other);
                        Other.clear();

//                      cout << "\nParse Phrases:---";
//                      Common::printVector(Phrases);
                        Phrases.clear();


                        cout << endl ;

                }
        }
        return 0;
}

if( clss == "EMAIL" || clss == "URL" )
{
        if(clss == "EMAIL")
                pattern = "[a-z0-9_-]+(.[a-z0-9_-]+)*@[a-z0-9_-
]+(.[a-z0-9_-]+)+";
                //"^[a-z0-9,!#\$%&'\*\+/=\?\^_`\{\|}~-]+(\.[a-z0-
9,!#\$%&'\*\+/=\?\^_`\{\|}~-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*\.([a-
z]{2,})$";
        else
                pattern = "[[a-z]+://]*[a-zA-Z0-9]+.*[a-zA-Z0-9]+.[a-
zA-Z0-9]+";

        for(unsigned int k=0; k<QA.TopPasg.size(); k++)
        {
                string text = QA.TopPasg[k];
```

```
text = text.substr(text.find_first_of("\n"));

string lines[10];
int nlines = Common::getLines(text, lines, 10);
if( nlines < 0)
      return -1;
for( int x=0; x<nlines; x++ )
{
      if(Common::trim(lines[x]).size() < 2)
            continue;

      string str1 = "Analyzing line :";
      str1.append(lines[x]);
      Common::printDelim();
      Common::printHeader(str1);
//    cout << "Line :" << lines[x] << endl;

      int ret = Common::searchPattern(pattern,
      lines[x], matches);
      if(ret == -1 || ret == -2)
      {
            cout << "No Match found!" << endl;
            continue;
      }

      qparse.Parse(lines[x], Focus, Keywords, Other,
      Phrases);

      cout << "Checking Focus similarity...." <<
      endl;

      ans.score = checkKeywords(Focus, Keywords,
      Other, Phrases, QA.Focus, QA.Keywords,
      QA.Other, QA.Phrases);
      int tot = Focus.size() + Keywords.size() +
      Other.size() + Phrases.size();
      if(tot > 0)
            ans.score = ans.score / tot;

      if( ans.score != 0.0)
      {
            for(unsigned int i=0; i<matches.num; i++)
            {
                  ans.answer =
                  lines[x].substr(matches.start[i],
                  (matches.end[i]-matches.start[i])
                  );
                  //Answers.push_back( ans );
                  Common::insertAnswer(Answers, ans);
                  cout << "Added Answer :" <<
                  ans.answer << endl;
                  cout << "Score :" << ans.score <<
                  endl;
                  cout << "Total Answers :" <<
                  Answers.size() << endl;
```

```
                        }
                    }

//                  cout << "\nParse Focus:";
//                  Common::printVector(Focus);
                    Focus.clear();

//                  cout << "\nParse Keywords: ";
//                  Common::printVector(Keywords);
                    Keywords.clear();

//                  cout << "\nParse Other Keywords:---";
//                  Common::printVector(Other);
                    Other.clear();

//                  cout << "\nParse Phrases:---";
//                  Common::printVector(Phrases);
                    Phrases.clear();

                    cout << endl ;

                }
            }
            return 0;
        }

        return 0;
}

bool AnswerExtraction::isNumber(string clss)
{
        string clses[] = { "NUMBER", "DATE", "PERCENT", "MONEY",
        "TEMPERATURE", "LENGTH", "HEIGHT", "MASS", "PERIOD", "AREA",
        "SPACE", "SPEED", "DENSITY", "ENERGY", "POWER", "TIME",
        "ORDEREDNUMBER" };

        for(int i=0; i<17; i++)
        {
            if(clss == clses[i])
                return true;
        }

        return false;
}

int AnswerExtraction::checkKeywords(const vector<string> &qFocus, const
vector<string> &qKeywords, const vector<string> &qOther, const
vector<string> &qPhrases, const vector<string> &aFocus, const
vector<string> &aKeywords, const vector<string> &aOther, const
vector<string> &aPhrases)
{
        int ret = 0;
        for(unsigned int i=0; i<aKeywords.size(); i++)
        {
            for(unsigned int j=0; j<qKeywords.size(); j++)
```

```
            {
                  if( Common::compare(aKeywords[i].c_str(),
                  qKeywords[j].c_str()) == 0 )
                        ++ret;
            }
      }

      for(unsigned int i=0; i<aFocus.size(); i++)
      {
            if( checkKeyword(aFocus[i], qFocus, qOther, qPhrases) )
                  ++ret;
      }

      for(unsigned int i=0; i<aOther.size(); i++)
      {
            if( checkKeyword(aOther[i], qFocus, qOther, qPhrases) )
                  ++ret;
      }

      for(unsigned int i=0; i<aPhrases.size(); i++)
      {
            if( checkKeyword(aPhrases[i], qFocus, qOther, qPhrases) )
                  ++ret;
      }

      return ret;
}

int AnswerExtraction::checkKeyword(const string Keyword, const
vector<string> &Focus, const vector<string> &Other, const
vector<string> &Phrases)
{
      int ret=0;
      for(unsigned int i=0; i<Focus.size(); i++)
      {
            if( Common::compare(Keyword.c_str(), Focus[i].c_str())== 0)
                  ++ret;
      }

      for(unsigned int i=0; i<Other.size(); i++)
      {
            if( Common::compare(Keyword.c_str(), Other[i].c_str())==0 )
                  ++ret;
      }

      for(unsigned int i=0; i<Phrases.size(); i++)
      {
            if(Common::compare(Keyword.c_str(),Phrases[i].c_str())==)0)
                  ++ret;
      }

      return ret;
}
```

----------Common.h----------

```
/* HEADER FILE FOR COMMON FUNCTIONS USED IN SYSTEM */

#pragma once

#include <iostream>
#include <vector>

const char * const TEXT = "text.txt";
const char * const LINES = "lines.txt";
const char * const SENTSEG =
"../sentenceboundary/sentence-boundary.pl -d
../sentenceboundary/HONORIFICS -i text.txt -o lines.txt";

#include "QAConstants.h"

using namespace std;

class Matches
{
      public:
            unsigned int num;
            unsigned int *start, *end;

            Matches()
            {
                  num = 0;
                  start = NULL;
                  end = NULL;
            }

            ~Matches()
            {
                  if(start) delete []start;
                  if(end)     delete []end;
            }
};

struct Answer
{
      string      answer;
      double      score;
};

class Common
{
      public:
            static bool Disambiguation;

      private:
            Common(){};
            Common(const Common &obj){};

            vector<string>    Entities;
            vector<string>    CommonWords;
```

```
public:
        void loadEntities();
        void loadCommon();
        static Common* getObject();
        static int getLines(string text, string lines[], int
        nlines);
        static string trim(string text);
        static int compare(const char *A, const char *B);
        static void freeArray(char *Data[MAX_KEYWORDS], int &size);
        static void printArray(const char * const
        Data[MAX_KEYWORDS], const int &size, ostream &out=cerr);
        static int getDataFileNames(const string &data_dir,
        vector<string> &Fnames);
        static int searchPattern(const string pattern, const string
        text, Matches &matches);
        static int writeToFile(const string &fname, const
        vector<string> &data);
        static int writeToFile(const string &fname, const
        vector<struct Answer> &data);
        static int writeToFile(const string &fname, const string
        data[], const int size);
        static int writeToFile(const string &fname, const string
        data);
        static int readFromFile(const string &fname, vector<string>
        &data);
        static int readFromFile(const string &fname, string data[],
        int &size);
        static int readFromFile(const string &fname, string &data);
        static bool isCommon(const char     *string);
        static bool isEntity(const char *string);
        static bool isPresent(const vector<string> &array, const
        string &item);
        static void printVector(const vector<string> &Data, ostream
        &out=cout);
        static void printDelim();
        static void printHeader(const string text);
        static void insertAnswer(vector<struct Answer> &Answers,
        const struct Answer &ans);
        static void replaceChar(string &source, const char oldCh,
        const char newCh);
};

bool Common::Disambiguation;
```

**----------Common.cpp----------**

```
/* COMMON FUNCTION USED IN THE SYSTEM DEFINED HERE */

#include <fstream>

#include <regex.h>

#include "Common.h"
#include "pstream.h"
```

```
int Common::getLines(string text, string lines[], int nlines)
{
      ofstream out(TEXT);
      if( !out )
      {
            cerr << "Error writing file:" << TEXT << endl;
            return -1;
      }
      out << text;
      out.close();

      system(SENTSEG);

      ifstream in(LINES);
      if( !in )
      {
            cerr << "Error reading file :" << LINES << endl;
            return -1;
      }
      char buf[512];
      int i;
      for (i=0; i<nlines && !in.eof(); i++)
      {
            in.getline(buf, 512);
            lines[i] = buf;
            lines[i] = Common::trim(lines[i]);
            if( lines[i].size() == 0 )
                  i--;
      }
      in.close();
      return i;
}

string Common::trim(const string text)
{
      int len = text.size();
      int beg = 0, end = len-1;
      while( (text[beg] == ' ' || text[beg] == '\t') && beg < end)
            beg++;
      while( (text[end] == ' ' || text[end] == '\t') && end > beg)
            end--;
      return text.substr(beg, (end-beg));
}

int Common::compare(const char *A, const char *B)
{
      int lenA = strlen(A);
      int lenB = strlen(B);
      if (lenA != lenB)
            return 1;
      for(int i=0; i<lenA; i++)
      {
            if (A[i] == B[i])
                  continue;
```

```
            else if((A[i] >= 'a' && A[i] <= 'z')&&(B[i]==(A[i]-32)) )
                  continue;
            else if((A[i] >= 'A' && A[i] <= 'Z')&&(B[i] == (A[i]+32)) )
                  continue;
            else
                  return 1;
      }
      return 0;
}

void Common::freeArray(char *Data[MAX_KEYWORDS], int &size)
{
      for(int z=0; z<size; z++)
            delete []Data[z];
      size = 0;
}

void Common::printArray(const char *const Data[MAX_KEYWORDS], const int
&size, ostream &out)
{
      for(int z=0; z<size; z++)
            out << Data[z] << " ";
}

int Common::getDataFileNames(const string &data_dir, vector<string>
&Fnames)
{
      unsigned int stnFname = Fnames.size();

      string path = data_dir;
      if(path[path.length()-1] != '/')
            path += "/";

      string command = "ls " + data_dir;
      redi::ipstream    f(command.c_str());
      const int         LINESIZE = 4096;
      char              line[LINESIZE];
      f.getline(line, LINESIZE);
      while (!f.eof())
      {
            command = line;
            Fnames.push_back(path + command);
            f.getline(line, LINESIZE);
      }
      f.close();

      cout << "Found " << Fnames.size() - stnFname << " files in
      directory" << endl;

      return (Fnames.size() - stnFname);
}

int Common::searchPattern(const string pattern, const string text,
Matches &matches)
{
```

```
        re_pattern_buffer       buffer;
        char                    map[256];

        buffer.translate = 0;
        buffer.fastmap = map;
        buffer.buffer = 0;
        buffer.allocated = 0;

        re_registers      regs;

        re_set_syntax(RE_SYNTAX_POSIX_EXTENDED);

        const char  *status =
        re_compile_pattern(pattern.c_str(),pattern.size(),&buffer);
        if (status)
        {
            cout << "Regex Error: " << status << endl;
        }
        re_compile_fastmap(&buffer);

        int ret = re_search(&buffer, text.c_str(), text.size(), 0,
        text.size(), &regs);
        if ( ret == -2)
        {

              cout << "Regex Search error!" << endl;
              return ret;
        }
        else if (ret == -1)
              return ret;

        matches.num = regs.num_regs;
        matches.start = new unsigned int [matches.num];
        matches.end = new unsigned int [matches.num];

        for(unsigned int i=0; i<matches.num; i++)
        {
              matches.start[i] = regs.start[i];
              matches.end[i] = regs.end[i];
        }

        //regfree(&buffer);

        cout << matches.num << " Matches Found!" << endl;

        return ret;
}

int Common::writeToFile(const string &fname, const vector<string>
&data)
{
        ofstream out(fname.c_str(), ios::trunc);
        if( !out )
        {
```

```
            cout << "Can't open file for writing:" << fname << "!" <<
            endl;
            return -1;
      }
      for(unsigned int k=0; k<data.size(); k++)
            out << data[k] << endl;

      out.close();
      return 0;
}

int Common::writeToFile(const string &fname, const vector<struct
Answer> &data)
{
      ofstream out(fname.c_str(), ios::trunc);
      if( !out )
      {
            cout << "Can't open file for writing:" << fname << "!" <<
            endl;
            return -1;
      }
      for(unsigned int k=0; k<data.size(); k++)
            out << data[k].answer << endl << endl;

      out.close();
      return 0;
}

int Common::writeToFile(const string &fname, const string data[], const
int size)
{
      ofstream out(fname.c_str(), ios::trunc);
      if( !out )
      {
            cout << "Can't open file for writing:" << fname << "!" <<
            endl;
            return -1;
      }
      for(int k=0; k<size; k++)
            out << data[k] << endl;

      out.close();
      return 0;
}

int Common::writeToFile(const string &fname, const string data)
{
      ofstream out(fname.c_str(), ios::trunc);
      if( !out )
      {
            cout << "Can't open file for writing:" << fname << "!" <<
            endl;
            return -1;
      }
      out << data << endl;
```

```
	out.close();
	return 0;
}

int Common::readFromFile(const string &fname, vector<string> &data)
{
	ifstream in(fname.c_str());
	if( !in )
	{
		cout << "Can't open file for reading:" << fname << "!" <<
		endl;
		return -1;
	}
	char buf[4069];
	string line;
	while (!in.eof())
	{
		in.getline(buf, 4096);
		line = buf;
		data.push_back(line);
	}

	in.close();
	return 0;
}

int Common::readFromFile(const string &fname, string data[], int &size)
{
	ifstream in(fname.c_str());
	if( !in )
	{
		cout << "Can't open file reading:" << fname << "!" << endl;
		size = 0;
		return -1;
	}
	char buf[4096];
	int i;
	for (i=0; !in.eof() && i<size; i++)
	{
		in.getline(buf, 4096);
		data[i] = buf;
	}
	size = i;

	in.close();
	return 0;
}

int Common::readFromFile(const string &fname, string &data)
{
	ifstream in(fname.c_str());
	if( !in )
	{
		cout << "Can't open file reading:" << fname << "!" << endl;
```

```
            return -1;
      }
      char buf[2048];
      if ( !in.eof() )
      {
            in.getline(buf, 2048);
            data = buf;
      }

      in.close();
      return 0;
}

Common* Common::getObject()
{
      static Common *obj = (Common *)0; //null
      if(!obj)
      {
            obj = new Common;
      }
      return obj;
}

bool Common::isEntity(const char *str)
{
      Common      *obj = getObject();
      for(unsigned int i=0; i<obj->Entities.size(); ++i)
            if(Common::compare(str,obj->Entities[i].c_str()) == 0)
                  return 1;
      return 0;
}

bool Common::isCommon(const char *str)
{
      Common      *obj = getObject();
      for(unsigned int i=0; i<obj->CommonWords.size(); ++i)
            if(Common::compare(str,obj->CommonWords[i].c_str()) == 0)
                  return 1;
      return 0;
}

bool Common::isPresent(const vector<string> &array, const string &item)
{
      for(unsigned int i=0; i<array.size(); ++i)
            if(Common::compare(item.c_str(),array[i].c_str()) == 0)
                  return true;
      return false;
}

void Common::loadEntities()
{
      Common      *obj = getObject();
      obj->Entities.clear();
      ifstream in("Entities.txt");
      if( !in )
```

```
            return ;
        char  buf[100];
        string      str;
        while ( !in.eof() )
        {
            in.getline(buf, 100);
            str = buf;
            obj->Entities.push_back(str);
        }
        cout << obj->Entities.size() << " entities loaded!" << endl;
        in.close();
}

void Common::loadCommon()
{
        Common      *obj = getObject();
        obj->CommonWords.clear();
        ifstream in("Common.txt");
        if( !in )
            return ;
        char  buf[100];
        string      str;
        while ( !in.eof() )
        {
            in.getline(buf, 100);
            str = buf;
            obj->CommonWords.push_back(str);
        }
        cout << obj->CommonWords.size() << " common words loaded!" <<
        endl;
        in.close();
}
void Common::printVector(const vector<string> &Data, ostream &out)
{
        for(unsigned int i=0; i<Data.size(); i++)
            out << Data[i] << ' ';
}
void Common::printDelim()
{
        cout << endl ;
        cout <<
"*************************************************************************
*************************************************************************"
<< endl;
        cout << endl ;
}

void Common::printHeader(const string text)
{
        cout << endl ;
        cout <<
"*************************************************************************
*************************************************************************" <<
endl;
        cout << text << endl;
```

```
      cout <<
"**************************************************************************
**************************************************************************" <<
endl;
      cout << endl ;
}

void Common::insertAnswer(vector<struct Answer> &Answers, const struct
Answer &ans)

      unsigned int i;
      Answers.push_back(ans);
      for(i=Answers.size()-1; i>0; )

            if(Answers[i-1].score >= ans.score)
                  break;
            Answers[i] = Answers[i-1];
            --i;
      }
      Answers[i] = ans;
}
void Common::replaceChar(string &source, const char oldCh, const char
newCh)
{
      for(unsigned int i=0; i<source.length(); i++)
            if(source[i] == oldCh)
                  source[i] = newCh;
}
```