

**CONTEXTQA: EXPERIMENTS IN INTERACTIVE RESTRICTED-
DOMAIN QUESTION ANSWERING**

A Thesis

Presented to the

Faculty of

San Diego State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

by

Martin Erik Liljenback

Fall 2007

SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Martin Erik Liljenback:

ContextQA: Experiments in Interactive Restricted-Domain Question Answering

John Donald, Chair
Department of Computer Science

Joseph Lewis
Department of Computer Science

Jean Marc Gawron
Department of Linguistics and Asian/Middle Eastern Languages

Approval Date

Copyright © 2007

by

Martin Erik Liljenback

All Rights Reserved

DEDICATION

I dedicate this work to my wife Wendy. Without her tremendous amount of love and support I doubt this would have been possible.

ABSTRACT OF THE THESIS

ContextQA: Experiments in Interactive Restricted-Domain
Question Answering

by

Martin Erik Liljenback
Master of Science in Computer Science
San Diego State University, 2007

The need for more advanced data mining and search engine technologies has been steadily increasing since the introduction of the Internet. With the exponential growth of information available on the web combined with a public that is becoming more educated in search technology, there exists a great need to quickly and efficiently be able to provide results for a large range of very specific questions. The current natural language processing is still in a primitive state. There is no single solution that will be able to provide quality results to the broad range of potential questions by using indexed data extracted from the web. However there exist several ways to provide more efficient results. One way is to develop more extensive ways to interact with users to target results related to the individual's specific needs.

This thesis focuses on a particular field of research that is called Question Answering Systems. In Question Answering the system provide answers on plain text questions through natural language processing, information retrieval, and data mining on structured or unstructured text data. A summary of the research development in this area is provided and also a description of how the algorithms and techniques have evolved over time until we are left in the current state.

Furthermore, I conclude that there are many compelling reasons to build more refined and targeted knowledge bases. With a targeted knowledgebase and knowledge about an individual specific needs, several algorithms can be applied which provides better results and efficiency than that of an open-domain question answering system. I show that index based search engines are far from providing the same level of accuracy as a restricted-domain QA systems. As part of the thesis a complete restricted-domain QA system is developed named ContextQA. A series of experiments are conducted where ContextQA is configured to use different approaches on restricted-domain question answering algorithms. The results show that high accuracy can be obtained within a restricted-domain with limited resources.

TABLE OF CONTENTS

	PAGE
ABSTRACT	v
LIST OF TABLES	x
LIST OF FIGURES.....	xi
ACKNOWLEDGEMENTS	xiii
CHAPTER	
1 INTRODUCTION.....	1
1.1 Chapter Organization	1
1.1.1 Introduction	2
1.1.2 Background	2
1.1.3 System Design.....	2
1.1.4 Building a Targeted Knowledge Base.....	2
1.1.5 ContextQA System.....	3
1.1.6 System Evaluations and Results.....	3
1.1.7 Future Enhancements	3
1.2 Contributions of This Thesis.....	3
2 BACKGROUND.....	4
2.1 Early Question Answering Utilizing Networked Computers.....	4
2.2 Frequently Asked Questions	6
2.2.1 Expert Inquiry Systems	7
2.2.2 Historical Limitations.....	7
2.3 Question Answering Systems.....	9
2.3.1 Generalized Question Answering Architecture.....	12
2.3.1.1 User Interface	12
2.3.1.2 Question Analyzer.....	13
2.3.1.3 Data Retrieval.....	13
2.3.1.4 Answer Extraction.....	14
2.3.1.5 Ranking	14

2.3.1.6 Answer Verification	14
2.3.2 History of Question Answering.....	15
2.3.3 Question Answering Roadmap.....	18
2.4 Open-Domain QA versus Restricted-Domain QA.....	19
2.4.1 Benefits in Open-Domain.....	21
2.4.1.1 Automatically Updated	22
2.4.1.2 Broad Coverage.....	22
2.4.1.3 Comprehensive Research Available.....	22
2.4.2 Limitations in Open-Domain	22
2.4.2.1 Large.....	23
2.4.2.2 Content Quality	24
2.4.2.3 Inaccurate Conflicting Data	24
2.4.2.4 Maintenance	25
2.4.3 Benefits in Restricted-Domain.....	26
2.4.3.1 Technical Terms.....	26
2.4.3.2 Accuracy in Understanding Content Words.....	26
2.4.3.3 Specialized Vocabulary.....	27
2.4.3.4 Access to Domain Specific Linguistic Resources.....	27
2.4.3.5 Structuring of the Knowledge Base	27
2.4.3.6 Quality and Reliability	28
2.4.3.7 Answer Formatting.....	28
2.4.4 Limitations in Restricted-Domain.....	29
2.4.4.1 Scarcity of Data.....	29
2.4.4.2 Complexity of Questions.....	29
2.4.4.3 Complexity of Answers.....	29
2.4.4.4. Evaluation of Restricted-Domain QA	30
2.4.4.5 Cost of Maintenance.....	30
2.5 The Challenges of Question Answering Systems	31
2.6 Attractive Market	33
3 SYSTEM DESIGN	34
3.1 Limiting the Problem Scope.....	34
3.1.1 Restricted Domain QA.....	34

3.1.2 Shallow Language Understanding	35
3.1.3 Completeness of Answers	35
3.2 Advanced Concepts.....	36
3.2.1 Interactive QA.....	36
3.2.2 Automation.....	38
3.2.3 Responsiveness.....	39
3.3 Previous Work.....	39
4 BUILDING A TARGETED KNOWLEDGE BASE.....	41
4.1 Question Harvesting System	43
4.2 System Design.....	43
4.3 Controller Servlet.....	44
4.4 Wizard Interface.....	46
4.4.1 Wizard Interface Step One	46
4.4.2 Wizard Interface Step Two	47
4.4.3 Wizard Interface Step Three	47
4.5 Document Parser	49
4.5.1 Parser Design.....	49
4.6 QA Database Schema.....	56
4.7 Results Using the System.....	57
4.7.1 Quality of QA Pairs.....	58
4.8 Refining the QA Knowledge Base	59
4.8.1 Spell Checker	60
4.8.2 Domain Specific Dictionary.....	60
4.8.3 Global and Domain Substitutions	61
4.9 Extending the QA Knowledge Base.....	61
4.10 Administration Interface	64
5 CONTEXTQA SYSTEM.....	65
5.1 Client.....	65
5.2 Application Server.....	66
5.3 Database	67
6 SYSTEM EVALUATION AND RESULTS	68
6.1 Measuring Results	68

6.1.1 The Importance of a Good Test Collection.....	70
6.1.2 Automated Test Framework.....	71
6.2 Resolving Question Candidates	71
6.3 Question Selection Agents	72
6.3.1 Agent Resources.....	73
6.3.1.1 Question Index	73
6.3.1.2 Domain Dictionary.....	74
6.3.2 Agent Results	74
6.3.2.1 Agent Homer	75
6.3.2.2 Agent Homer Performance Metrics	75
6.3.2.3 Agent Fry.....	76
6.3.2.4 Agent Fry Performance Metrics.....	77
6.3.2.5 Agent Bender.....	78
6.3.2.6 Agent Bender Performance Metrics.....	79
6.3.2.7 Agent Yoda	83
6.3.2.8 Agent Yoda Performance Metrics.....	84
6.4 Conclusion.....	86
7 FUTURE WORK	87
APPENDIX	
A TECHNICAL SPECIFICATIONS	98
B QA ADMINISTRATION INTERFACE	100
C TEST QUESTIONS	105

LIST OF TABLES

	PAGE
Table 2.1. Example of Answer Extraction from Multiple Sources.....	20
Table 4.1. Distribution of TREC QA Track Question Types.....	43
Table 4.2. Text Parser Features.....	51
Table 4.3. Example of Global Substitutions.....	61
Table 4.4. Rephrasing Questions through Translation.....	62
Table 6.1. Test Collection.....	75
Table 6.2. Agent Homer Performance Metrics.....	75
Table 6.3. Agent Homer Aggregate Performance Metrics.....	76
Table 6.4. Test Collection.....	78
Table 6.5. Agent Fry Performance Metrics.....	78
Table 6.6. Agent Fry Aggregate Performance Metrics.....	78
Table 6.7. Test Collection.....	80
Table 6.8. Agent Bender Performance Metrics.....	80
Table 6.9. Agent Bender Aggregate Performance Metrics.....	80
Table 6.10. Test Collection.....	84
Table 6.11. Agent Yoda Performance Metrics.....	85
Table 6.12. Agent Yoda Aggregate Performance Metrics.....	86
Table 7.1. Example of Query Template Rephrasing.....	89
Table 7.2. Conceptual Question Categories with Examples.....	91

LIST OF FIGURES

	PAGE
Figure 2.1. Networked computers sharing bulletin board systems.	5
Figure 2.2. Different questions leading to the same answer.	8
Figure 2.3. Information retrieval system.	10
Figure 2.4. Generalized question answering architecture.	12
Figure 2.5. Intersecting QA research fields.	32
Figure 3.1. A.L.I.C.E tree structure.	37
Figure 4.1. MVC software architecture.	44
Figure 4.2. Controller Servlet UML diagram.	45
Figure 4.3. Step one of the FAQ parser wizard.	47
Figure 4.4. Step two of the FAQ parser wizard.	48
Figure 4.5. Step three of the FAQ parser wizard.	48
Figure 4.6. FAQ parser classes.	50
Figure 4.7. Program flow diagram for the FAQ parser.	52
Figure 4.8. Program flow chart for the extract-question method.	54
Figure 4.9. Question and answer container classes.	55
Figure 4.10. QA database ER diagram.	56
Figure 4.11. Question word count distribution.	58
Figure 4.12. Distribution of initial unigrams.	58
Figure 4.13. Results for knowledge base n-gram verification.	63
Figure 4.14. N-gram verification results across different languages.	64
Figure 5.1. ContextQA system.	65
Figure 5.2. ContextQA client interface.	66
Figure 5.3. High level process flow of the ContextQA system.	67
Figure 6.1. Precision and recall.	68
Figure 6.2. Agent Homer correct answers per index.	76
Figure 6.3. Agent Fry correct answers per index.	79
Figure 6.4. Agent Bender correct answers per index.	81

Figure 6.5. Confidence levels versus query results.....	82
Figure 6.6. Neural network.	84
Figure 6.7. Agent Yoda correct answers per index.....	85

ACKNOWLEDGEMENTS

I want to acknowledge my thesis chair Dr. Donald. Dr. Donald has always encouraged me in my work. I would also like to thank Dr. Gawron, and Dr. Lewis for supporting me in this work and being part of my thesis committee. I want to acknowledge the Google team behind Google Scholar. Great job guys, now there is no excuse for anyone not to stand on the shoulders of giants.

CHAPTER 1

INTRODUCTION

The history of question answering goes far beyond computer software systems. One of the most powerful ways to learn is by asking questions of someone that has an intimate knowledge in the field that you are interested in learning about. There is a period while growing up when we frequently ask our parents questions about everything in our surroundings so that we can build up knowledge about the world we live in. In many schools teachers now practice inquiry based learning. This has many times proven itself to be a more powerful way to learn as indicated by Harlen (2004). Inquiry based learning is a different approach rather than older style teaching where students mostly listen and document what the teacher is saying. One reason why inquiry based learning has not previously been practiced that widely is because it requires a lot of interaction with individual students. With the introduction of the internet students can now practice inquiry based learning by using online resources. Question answering systems would provide a very powerful way to practice inquiry based learning.

Computers most common task is to quickly process large amounts of information to solve different types of logical problems. Another way to word this is that we use computers to provide us with answers to problems that are considered too hard or time consuming for us to process by hand. We interact with computers through graphical and hardware user interfaces such as using a keyboard and a mouse to control a software application while reading results on a monitor. These types of interfaces have evolved to become rather sophisticated throughout the years, but they still do not compare to how we interact with other people while asking questions. Question answering systems are a step in that direction.

1.1 CHAPTER ORGANIZATION

This section contains a brief summary on the contents of each chapter. These summaries give an overview on how the thesis document is organized.

1.1.1 Introduction

In this chapter I describe what my goals are and what I plan to accomplish. The chapter gives a short description of Question Answering systems. Furthermore, I describe the need and value these systems provide to our community. In addition, this chapter includes an outline of the entire document.

1.1.2 Background

There is a comprehensive description on the history of question answering and how it has evolved since the inception of computers and computer networking. A roadmap of future work in this field is presented and discussed. The need of question answering systems is described throughout this chapter. This chapter holds some detailed information on how a generalized question answering system architecture can be constructed. The different sub-systems in a question answering system are described. There is an in depth analysis of open-domain question answering systems versus restricted-domain question answering. In this section I show some of the clear benefits of using restricted domain question answering systems. This chapter also contains a section describing the complexities of question answering and what makes it one of the hardest problems to solve. The chapter is completed with a section where I describe the potential market of question answering and that the reasons why companies keep on funding projects to develop question answering systems.

1.1.3 System Design

This chapter describes how I choose to tackle some of the problems so that I could obtain the goals I had set out to reach with this thesis. I list some of the previous work within restricted question answering and give an idea on what needs to be tackled next.

1.1.4 Building a Targeted Knowledge Base

In this chapter I cover the work I did when trying to automate the knowledge building phase by using open-domain resources in obtaining a restricted domain question answering knowledge base. By extracting QA-pairs from online frequently asked questions pages a knowledge base is constructed. The work refining this knowledge base is described in some detail and various metrics on the final knowledge repository are also provided. A novel

approach of extending an existing knowledge repository within question answering using translation to other languages is discussed and results are provided.

1.1.5 ContextQA System

The system design of the question answering system that I constructed for this thesis is described in this chapter. The framework, resources, and the way algorithms were implemented is covered.

1.1.6 System Evaluations and Results

This chapter provides results and findings from several different question answering algorithms that were run through a test collection within the ContextQA system.

1.1.7 Future Enhancements

In this chapter I write about topics that were not covered in my work but would serve as a natural continuation of it. I write about what I believe is the correct way of implementing an open-domain question answering system using the features and advantages you obtain within restricted domain question answering.

1.2 CONTRIBUTIONS OF THIS THESIS

The main objectives and contributions of this thesis are listed below:

- Provide an overview on the past and present state of Question Answering and different approaches on how to solve the problem.
- Discuss and show the benefits of an interactive restricted-domain Question Answering System.
- Build a refined knowledge base that can be used as a resource for future research work on Question Answering at San Diego State University.
- Research, design and implement a fully working restricted-domain Question Answering System.
- Present the performance of different algorithms in a QA system implementation.

CHAPTER 2

BACKGROUND

This chapter describes the history of Question Answering. A common Question Answering architecture is presented. Open domains versus restricted domain question answering systems are described in depth.

2.1 EARLY QUESTION ANSWERING UTILIZING NETWORKED COMPUTERS

Providing answers to questions across networked computers was first introduced by the use of bulletin board systems (BBS). These types of systems were hosted on computers connected to the phone network. Bulletin board systems were popular during the 1970s and 1980s especially in Europe. The servers hosting the bulletin boards used different methods and protocols to synchronize messages such as FidoNet (Bush, 1995). By utilizing these types of protocols messages entered on one server could be distributed to all the other servers that were part of the network. These servers were accessible through local phone numbers using modems as seen in Figure 2.1. By synchronizing servers questions got a lot more exposure than if only hosted on one single machine. The questions and knowledge exchange in these early networks were mostly computer science related. At the point when the internet started to become more widely adopted a similar system was developed named UseNet. UseNet was a set of news servers connected in a somewhat organized network. Today UseNet spans most of the globe and is still very actively utilized. The connected servers communicate using NNTP (Network News Transfer Protocol) (Kantor, & Lapsley, 1986). This system gained popularity very quickly after it was introduced. The amount of articles, questions and answers grew exponentially and today news servers are considered a major knowledge resource for almost any type of information. Various companies such as Google among others have realized that indexing this type of information can be a very powerful resource. The information available in these archives consists mostly of people submitting questions that other users get a chance to answer.

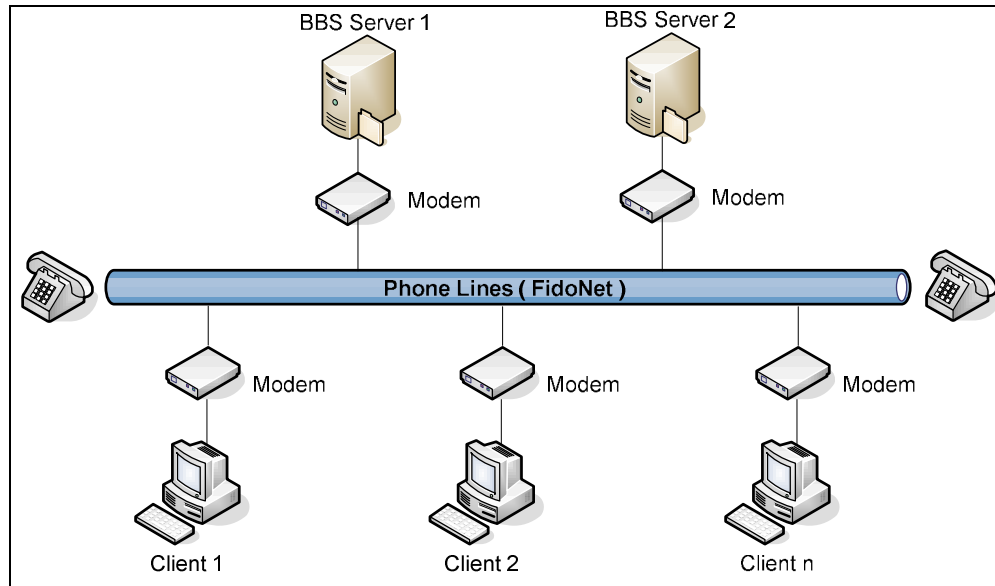


Figure 2.1. Networked computers sharing bulletin board systems.

This way a user can perform keyword searches to quickly locate postings containing answers to common questions. Searching for answers for more specific questions this way usually takes some work by analyzing resulting articles. It also takes some practice knowing what keywords to use to get quality results. Answers to questions that are not that common are much harder to find using this method and the result will usually not guarantee a correct answer. Ideally a user would be presented with one single answer instead of a list of documents. The resulting relevant documents might not even contain the answer, or might contain false answers. When viewing logs from popular search engines approximately 15% of all queries posted are fully formulated questions. The following might be some examples of the type of questions people submit to search engines:

- What are the top rated colleges in the UK?
- How do I lose weight?
- Where can I buy quality headsets?
- How many kilograms are five pounds?
- What does bilingual mean?

In a related work, Radev, Libner, and Fan (2002) concludes that when posting complete questions to search engines correct answers are only found three fourths of the time within the first forty results returned. This shows that there is a great need to provide more

precise answers instead of a list of links to documents that potentially hold the answer. However, the quality of the results for plain text questions is likely to degrade even further if QA systems do not become an integral part of regular search engine technology any time soon. The reason for this is that it is usually better to type in several well thought out keywords that correspond to the question context. Plain text questions tend to generate poor results and are even discouraged by certain search engines.

2.2 FREQUENTLY ASKED QUESTIONS

One popular way of sharing inquiry based information on the internet is by listing frequently asked questions (FAQ) and their answers on web pages or in text documents. The FAQ originated as a text medium on ARPAnet mailing lists and later FAQs became available on UseNet in the form of "*.answers" moderated newsgroups. Many internet based organizations and corporations will first direct clients to read through a publicly available FAQ before being able to submit questions to support personnel. This limits the amount of redundant questions and reduces the resources spent on support personnel.

A FAQ has several drawbacks when compared to an interactive QA system. If the FAQ is too small, the questions will most likely not cover enough of the topic. The positive aspect of this is that it becomes easier to quickly read through the FAQ to determine if the question is covered or not. A large FAQ on the other hand can be intimidating to read through and is not always guaranteed to provide the correct answer either. Sometimes larger FAQs will be covered on several documents. This can provide some structural advantages but makes it harder to search for specific keywords relating to a question. Another problem with a static FAQ is that there is no straightforward way for an administrator to find out what types of questions clients have. This can be solved by providing some sort of feedback such as email, but usually a client will want the question answered right away and will search elsewhere for an answer before submitting any feedback. If feedback were to be provided an administrator could theoretically refine the FAQ based on empirical knowledge on what types of questions are submitted the most. The FAQ is still today the most common way to provide answers to common questions online. However one of the greatest drawbacks with static FAQs is that the client is not given the option to ask free form questions but is forced to re-phrase the question so that it matches what might be available.

2.2.1 Expert Inquiry Systems

Another common way to provide answers online is by having a staff of experts whose job it is to answer client's questions. These types of systems are geared more towards knowledge bases covering specific topics. An example would be questions on how to invest your money or how to write programs for computers. Many of the leading search engine companies now provide experts that can answer or research more complex questions. This is another indication that the public many times searches for specific answers that cannot be found or are hard to find given the indexed content available through search engines. These services usually come with a small fee as can be seen on Google Answers. There also exist several free systems such as Expert Exchange where authors instead of charging for their services get awarded virtual points for answering questions. These free systems usually cover common topics and have a growing community of people giving free advice.

2.2.2 Historical Limitations

The systems described up until now are greatly limited by either their interface or their cost. They also require constant maintenance by people or have a limited number of statically defined questions and answers. The flexibility of these types of systems is therefore limited. When questions are submitted to a system that is maintained by people, one or more might answer or none at all. If an answer is provided, it will not be provided instantly. Usually it takes one or more days before an answer becomes available. In the systems described so far it is also tedious to find existing answers even if the question has previously been submitted and an answer already exists. The reason for this as stated previously is that there are so many different ways to formulate questions that lead to the same answer. An example of this is depicted in Figure 2.2. One good reason to create a system that can automatically answer questions is due to the massive amount of online information that is publicly accessible today. There is however no solid standard on how to organize this information or how to organize systems that are designed to provide answers to questions. The information available online is usually not organized very well which makes it hard to manually search for answers with specific contexts. In an automatic QA system manual searching is not required. All the complexity that is required to provide an answer is instead

implemented in the system layer. In a QA system the answer is (or at least should be) provided in real-time.

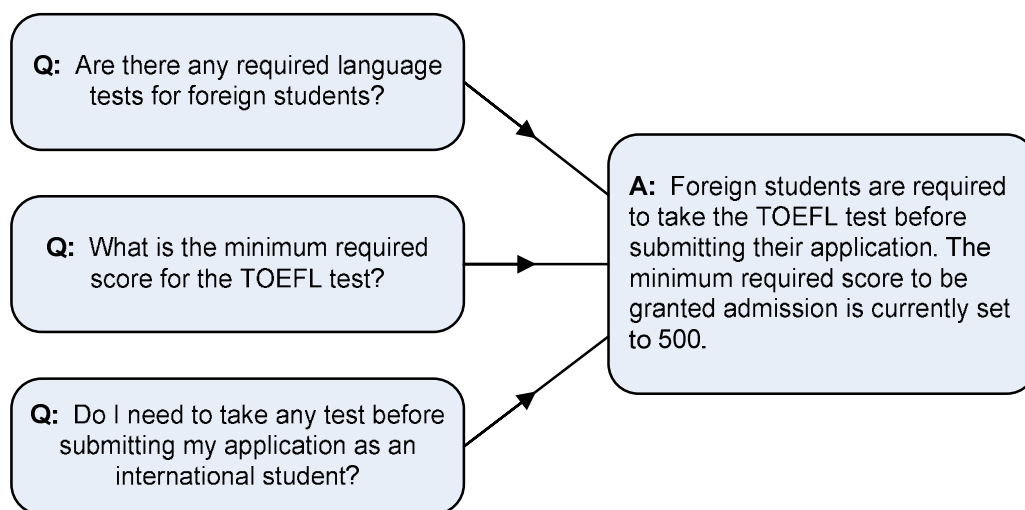


Figure 2.2. Different questions leading to the same answer.

A massive amount of resources has been spent and is actively being spent on research and development in this area. Designing an automated QA system is not in any way a novel idea, but it is not until recently that it has started to become more commercially feasible to do. One reason is the cheap storage and processing power that is now available. There has also been a lot of progress in developing parallel computing clusters using networked computers especially within the open source community. Some examples of systems like these are the NASA developed Beowulf design, and Sun Microsystems Inc.'s N1 Grid Engine. Any decent QA system tends to be resource intensive and can benefit from these types of algorithms.

The government has historically contributed a large amount of resources trying to develop IR and QA related systems. One example of such a system is CYC (Lenat, 1995) which is an attempt to create a natural language processing system that can process and understand plain text documents such as the New York Times. CYC does this by creating ontologies which are basically context driven relationships between words in a sentence. This works to a certain degree and allows CYC to build a vast knowledge base of simple relationships and word meanings. These relationships are not much use when it comes to providing answers to more complex questions. More logical questions such as: *is a dog a*

cat? could be analyzed by using the knowledge base of CYC to be determined to be false. The knowledge base itself can be used as part of a QA system. Using CYC in a QA system has only recently been considered (Curtis, Matthews, & Baxter, 2005). This became possible when part of CYC was released under an open source license. CYC is a good example on how much work is required to just get a basic understanding of written or spoken language.

Given lessons learned from previous approaches today's QA systems mostly rely on hybrid solutions where shallow language understanding is used combined with some aspects of natural language processing.

2.3 QUESTION ANSWERING SYSTEMS

Question Answering Systems have evolved from the field of Information Retrieval. Information Retrieval traditionally takes as input a set of keywords that constitutes a query. The query is then processed by various algorithms. Usually the query is broken up by separating all the individual keywords which are then used to search for relevant documents. During the search the keywords will be matched up against an index that references the different documents. Many times a keyword query will not contain sufficient information to produce quality results. One reason can be due to the lack of keywords or the lack of descriptive keywords. This is a very likely scenario when clients type in the queries manually. These types of problems can be solved to a certain degree by applying query rewriting, or query expansion prior to searching. There are several different algorithms that are involved in this process such as finding keyword synonyms, and possibly apply stemming algorithms. Stemming was first introduced by Lovins (1968) and is used to obtain a word morphological root, thereby reducing the granularity of words that need to be indexed. Stemming is usually an integral part in any modern IR system. When a resulting set of documents has been retrieved these documents are sorted based on relevance and then presented to the user as seen in Figure 2.3. The most common ways of doing this is to use a Boolean Model or a Vector Model. The Boolean model suffers from the fact that it is binary and will blindly exclude documents whose index terms (keywords) do not qualify the Boolean expression. Let t_i be index terms in the following Boolean expression: $q = t_1 - (t_2 / t_3)$. For the query q to be true index term t_1 is required, and either t_2 or t_3 needs to be present as index terms for the document. This works well when controlling exactly what document

subset is returned. Many search engines use the Boolean model. The main disadvantage of the Boolean model in IR and QA is that index terms can only be given Boolean weights i.e. $w_i \in \{0,1\}$. This means that with too restrictive expressions no documents will qualify. On the other hand a very general expression will result in too many documents being returned. A more popular model that allows for non-binary weights is the vector space model (VSM). In this model the terms of the query are given weights and so are terms within documents.

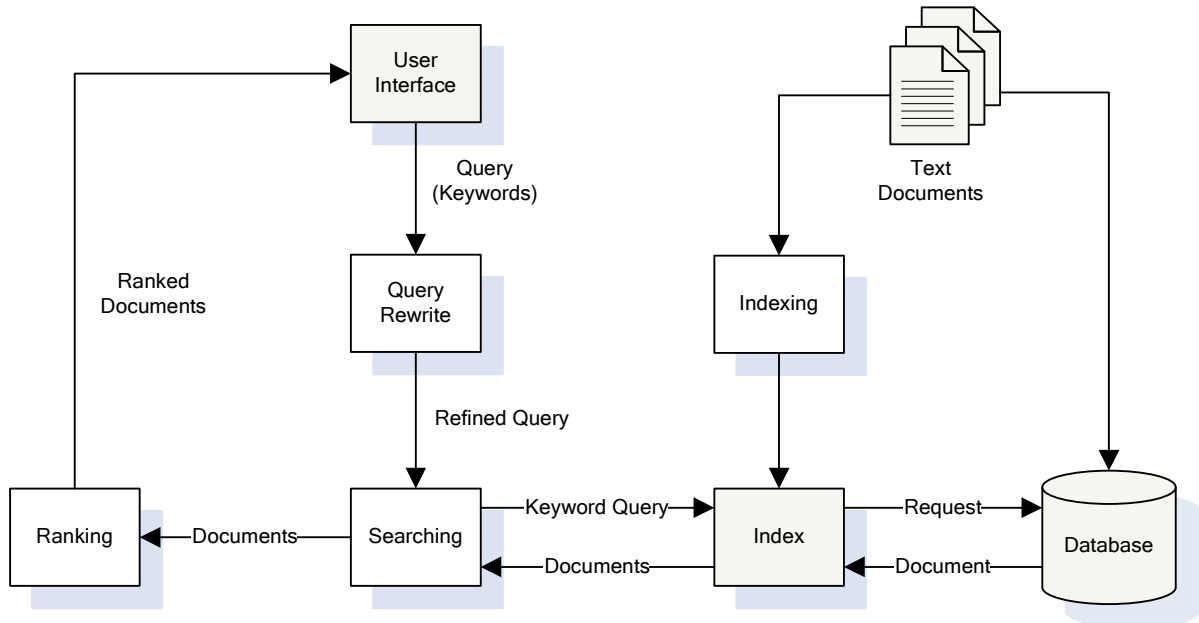


Figure 2.3. Information retrieval system.

These term weights are represented by two n -dimensional vectors, one vector \vec{d} for the document, and one vector \vec{q} for the query. A document relevancy number can be calculated by taking the cosine angle between these two vectors. This is done by taking the vector dot product divided by the vector cross product.

$$rel(d, q) = \frac{\vec{d} \cdot \vec{q}}{|\vec{d}| \times |\vec{q}|}$$

The VSM can qualify documents that are only similar to a certain degree to the query. The amount of documents retrieved can be adjusted by setting a cut-off value to the

relevance score. The term weights can be calculated in many different ways. One of the more popular ways is the tf-idf (term-frequency and inverse document frequency) weighting scheme developed by Salton and Buckley (1988). The term frequency tf of a term t_i is determined by dividing the frequency of the term in a document by the frequency of all terms in that document.

$$tf = \frac{n_i}{\sum_k n_k} \quad idf = \log \frac{|D|}{|(d_i \subset t_i)|}$$

The inverse document frequency idf is obtained by dividing the total number of documents with the number of documents containing the term. Finally the $tf-idf$ weight is calculated by taking the term frequency times the inverse document frequency. The $tf-idf$ weighing scheme has received a lot of attention in the IR research community, and is one of the more popular models.

The main problem with information retrieval is that when provided with the resulting full text documents finding the requested answer can many times be difficult. *An information retrieval system does not inform (i.e. change the knowledge of) the user on the subject of his inquiry. It merely informs on the existence (or non-existence) and whereabouts of documents relating to his request* (Lancaster, 1968). To simplify this analysis, a system that can perform Information Extraction (IE) to extract specific content data from resulting documents would be needed. That way a user could more quickly determine whether the information is available or not. Providing this feature is one step closer to Question Answering.

The task of a Question Answering System is when given a plain text question to extract information from its knowledge base and serve up an intelligent answer back to the client. A simple keyword query in Information Retrieval is trivial when compared to being able to understand a question written in natural language. Many have said that Question Answering is in fact the holy grail of Information Retrieval. The scientific reason for Question Answering is that the ability to provide an intelligent answer given a structured or unstructured source of information can be considered the essence of understanding.

2.3.1 Generalized Question Answering Architecture

Sometimes when asked a question it can be difficult to give an answer even when possessing the knowledge required. Programming a machine to answer questions becomes much more difficult. However several approaches on how to design such a system have been suggested and implemented. In this section the general architecture of a Question Answering System is presented, and all the different parts are explained. Each section directly refers to the different parts depicted in Figure 2.4.

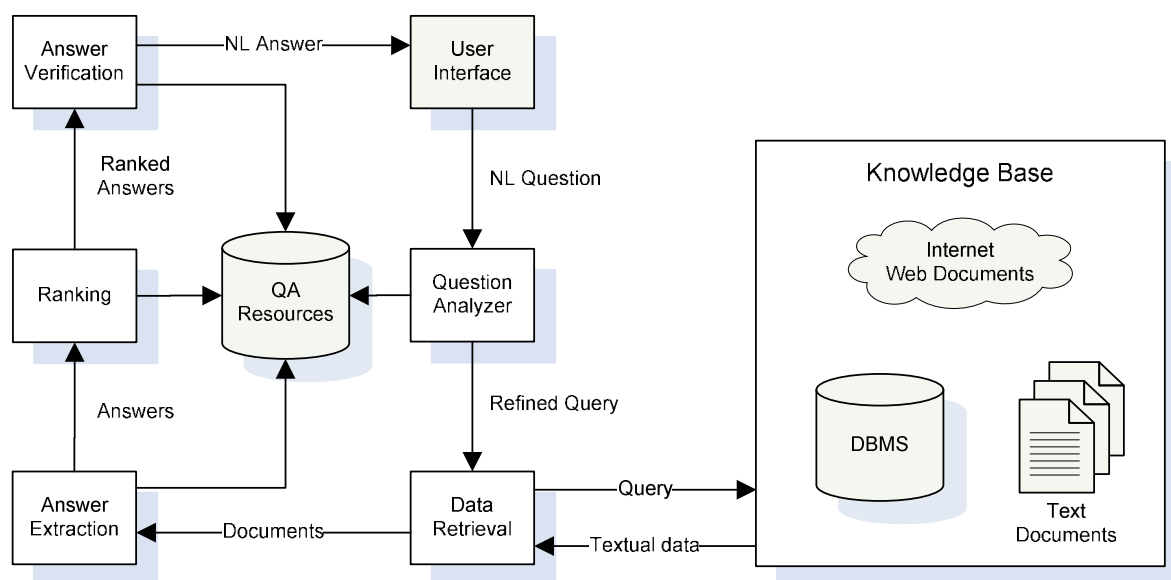


Figure 2.4. Generalized question answering architecture.

2.3.1.1 USER INTERFACE

Most question answering systems that have recently been implemented provide the user with a web based form where questions can be entered. The question is then submitted and the system interprets the question and responds by returning a formatted answer similar to what you would expect from a person. The user interface could also be constructed by a speech recognition and speech synthesis interface. The user interface is an important part of a QA system but not that much research has been devoted to it as of yet. Many interfaces mirror the look and feel of a search engine such that not only the answer is presented but also other answers that received a high confidence score. Future QA interfaces will most likely

blend transparently into our everyday life such as telephone support, household products, and car navigation systems.

2.3.1.2 QUESTION ANALYZER

The question analyzer plays an important role in any type of question answering system. In this module the question is analyzed and processed to extract as much information as possible that can be used later in the data retrieval phase. Different implementations will differ in the depth of analysis at this step. For example such analysis may involve breaking up the words in the question and using everything but the stop words in the search compared to complete syntactic parsing of the sentence. Stop words are words that will not increase the performance in the retrieval phase. These stop words can be words such as *is, are, he, which*, etc. In open-domain systems the question is commonly rephrased into what is thought to be part of the answer.

- Where is the Eiffel tower located? (original question)
- the Eiffel tower is located (part of answer)
- the Eiffel tower is close to (part of answer)
- the Eiffel tower can be found in (part of answer)

That way several parts of the possible answer are used when searching through the document collection. Query expansion is another way to increase the chances of finding a document containing the answer. Question terms can be expanded into several terms using synonyms. Other features about the question can also be extracted such as the question type.

2.3.1.3 DATA RETRIEVAL

Some of the information that has been extracted in the question analyzer will be utilized to query the knowledge base for information. This can be done in several different ways. An open-domain QA system would use either a custom search engine or a third party search engine to search documents distributed over the internet. A closed-domain system can search unstructured, semi-structured, or structured data sources such as a database. Within this part of the system other pre-processed information is many times used to improve performance. These types of resources can be anything from a comprehensive index to pre-processed parts of the document collection. The data-retrieval part of a QA system is many

times similar to the data-retrieval that is done in an IR system which would utilize methods such as Boolean keyword search or term weighing. In the data-retrieval phase of a QA system it is important to retrieve as much relevant information as possible. The quality is less important because the result will not necessarily be presented to the user. What is more important is that the information asked for be found. That means that if the system has enough performance to process the information the larger amount of information the better.

2.3.1.4 ANSWER EXTRACTION

Answer extraction falls under IE (Information Extraction). In this part of the QA system the information has been retrieved. The information can be either documents or text resulting from a database query. This information is used to extract passages that relate to the question asked. At this part QA systems really start to deviate. Some systems will consider a passage containing the answer a valid response. Other systems will try to create a properly worded answer. An open-domain system can differ from a closed-domain system in this step where an open-domain system will always have a set of documents at this stage or a set of passage summaries as result of a search engine query. The most basic way to generate an answer is to extract parts or chunks of the information that relate most closely to the question. These chunks can then be put together to form an answer or several answers. There are many different ways IE algorithms rate what passages to extract (Tellex, Katz, Lin, Fernandes, & Marton, 2003).

2.3.1.5 RANKING

If the answer extraction results in more than one answer these answers are ranked based on relevance. Again, there are many different approaches on how to properly weight the answers and this is closely connected to how the answer is extracted from the information processed in the answer extraction phase.

2.3.1.6 ANSWER VERIFICATION

Some more advanced systems will further improve the accuracy by analyzing the resulting answer using deeper NLP methods to justify it against the question. The question and answer are parsed and converted to their logical forms. The question and answer are then compared by trying to logically prove the correctness of the answer by using methods such as

abductive justification. This is done by using information and facts contained within the documents, world knowledge which is many times extracted from WordNet, and domain specific ontologies.

2.3.2 History of Question Answering

Designing a Question Answering system is no novel concept. Several systems have been produced since the 1960's. The first systems were restricted-domain QA systems that interfaced against databases. One example of such a system is BASEBALL developed 1961 by Green, Chomsky, and Laughery (1961). This system was designed to provide factoid type information about the American baseball league statistics. This was done by using shallow language parsing techniques. Another system similar to BASEBALL was developed by Woods (1973) and was named LUNAR. LUNAR was able to answer questions regarding the rock samples returned from the Apollo lunar exploration. Wood system translated the questions to one or more database queries. The TEAM system developed by Grosz (1983) had some basic features such as semantic representation routines, and a schema translator that made it more modular than the previous two systems. The similarities between all three systems are that they were all using databases to store their knowledge base. The design of these databases and the structured data they contained was all created manually by experts in their respective fields. Having a very structured knowledge base made the systems perform very well against their targeted domain of expertise. Any questions relating to topics outside the targeted domain would generate poor results.

Dialog systems started to appear around the 1960s. These systems were mostly influenced by the test suggested by Alan Turing which he called "Imitation Game" (Turing, 1950). The theory behind the test was that a machine is to be considered intelligent if a person communicating with the machine via teletype could not distinguish it from a real person. Conversational systems such as Joseph Weizenbaum's ELIZA (1966), and the "Conversation Machine" (Green, Berkley, & Gotlieb, 1959) were the first systems that could be verified against a Turing Test. These systems were designed to provide ways to carry on a basic conversation by recognizing certain word patterns. The result could be quite convincing but even a novice user could quite quickly find flaws. In 1991 Hugh Loebner started the Loebner Prize Competition, offering a \$100,000 prize to the author of the first computer

program to pass an unrestricted Turing Test. This has not yet been accomplished, but several comprehensive rule based systems have been competing each year.

Using natural language processing for story comprehension received a lot of focus during the 1970s. One system like this was MARGIE (Schank, Goldman, Riesbeck, & Rieger, 1975). This system was able to process documents which it later could answer basic questions about. This was done by parsing and organizing the document's information in a similar pattern as human memory is thought to work. This was further improved by Lenhart, Dyer, Johnson, Yang, and Harley (1983) in the BORIS system. BORIS explored story comprehension by introducing elements such as emotions and themes. BORIS and MARGIE are systems that more closely mirror the way open-domain question answering systems work today. The likelihood comes from the way the systems are designed to extract and process information from unstructured text.

During the 1980s Natural Language systems that interfaced against databases started to get a higher commercial adoption rate. Among the companies that started to offer these types of solutions were EasyAsk who now back enterprise customers such as Forbes, BASF, and FedEx with certain QA solutions.

Some of the first web-based QA systems appeared during the 1990s. MIT's START, appeared in 1993, (Katz, 1997). START used annotation to break up sentences in something called a ternary expression <subject relation object>. These ternary expressions were later used to more efficiently issue queries against the system's knowledge base. START was followed by Ask Jeeves in 1996. Ask Jeeves supports NL queries and is thought to mainly use query templates to match questions (Sneiders, 1999).

The focus on Question Answering systems within the research community got a major boost during the last decade. This was due to a combination of things such as large amounts of indexed documents becoming available with the introduction of the internet. Also advances within Information Extraction, and more commercial interests for QA solutions caused the field of Question Answering to get more attention. Other things that have made it easier to develop more complex QA systems are resources such as WordNet (Fellbaum, 1998), and OpenCyc (Curtis, Matthews, & Baxter, 2005). WordNet provides access to semantic information, and semantic relationships. CYC provides access to ontology

information and everyday common sense knowledge. Both these systems have now been made freely available to the public.

The U.S. government's Text Retrieval and Evaluation Conference (TREC) Question Answering track (Voorhees, 1999) also made a big contribution. The question answering track was initiated at TREC-8 in 1999 with 20 participants. The Question Answering track is a competition to evaluate systems of question answering in open-domains. This event enabled researchers to start sharing their experiences, and also compare their results using common metrics. Every year the TREC competition has gotten more comprehensive in its QA tasks. In the first QA track the participants were required to return a 50 to 100 character long string as a response to a question. This string did not need to be formulated correctly but was supposed to hold the answer to the question. Most of the questions lead to short factual answers such as the name of a person or a date. These types of questions are called factoid questions. Since then another question category has been introduced when a question can produce several answers. An example of a list question could be, "Name three car manufacturers". Aggregating a list of answers makes the task a lot more complicated because the list of answers often needs to be extracted from several different documents. Throughout the last question answering tracks Language Computer Corporation (LCC) has been the best performing team by far, averaging close to 70% accuracy on factoid type questions. In the 2004 competition the team in fourth place had 34.3% accuracy (Voorhees, 2004). That score is less than half of what LCC got, and shows that the average QA system performance is still quite low. For the list questions the top ten teams in the 2004 competition had an average accuracy of 24%.

In the last TREC QA competitions the approaches used by different teams can be summarized as statistical, rule-based, and mixed. Most advanced Question Answering systems are starting to become exceedingly complex, often using several different modules such as information retrieval, sentence parsing, pinpointing question-types, semantic analysis, and even reasoning (Moldovan et al., 2002) to evaluate, and rank answer candidates. With the increased complexity of QA systems it many times becomes harder to pinpoint where the increased performance comes from.

2.3.3 Question Answering Roadmap

Given the success of the TREC competition a committee was put together by some of the key people involved in the competition to produce a roadmap for question answering. This roadmap was intended to cover many aspects of QA and what type of enhancements would be expected over the next years. The result was a roadmap document (Burger et al., 2000) describing a vision of future QA. Questions range from simple facts to complex scenarios such as producing answers based on the context of a discussion. The committee determined that users of a QA system would want the following key features from a QA system:

- *Timeliness*. Answers should be provided in real-time. Knowledge bases should include recent and complete information.
- *Accuracy*. The precision of the QA system should be flawless. Not responding to questions that are not known is important. QA systems should mimic common sense inference.
- *Usability*. More domain specific knowledge must be incorporated in QA systems.
- *Completeness*. Complete answers need to be provided. Answer should be fused between several different data-sources. Open-domain and restricted-domain knowledge might need to be combined to provide an accurate answer.
- *Relevance*. Answers to a question must be relevant within the current context. This relates to when a user issues follow up questions, and the system considers the context of previous questions to determine the answer.

Given the above goals, systems have not really improved that much since the roadmap was released. Many systems still struggle with factoid type questions. Open-domain systems still do not formulate answers in a very readable way. The state of the art open-domain systems are not close to being real-time, especially not if you consider that a QA system should support serving thousands of clients at the same time.

Some focus in the roadmap document was put on different types of questioners and that different answers need to be produced based on their sophistication.

- Level 1. Casual questioner.
- Level 2. Template questioner.
- Level 3. Cub reporter.
- Level 4. Professional information analyst.

Several advanced QA research areas were presented with examples how a QA system should reason to provide answers that fit the questioner based on their sophistication level. Question classes was the first research topic. One important aspect of determining a question class is to determine the focus of the question. This can often only be done by knowing additional information about question context, domain knowledge, and also general world knowledge. The second research topic was to be able to determine question ambiguities and implications. An example question could be: “what recent drugs has Pfizer introduced on the market?”. The ambiguity here would be drugs developed by Pfizer or marketed by Pfizer. The third research topic was context. Supporting context based QA would mean that the system could potentially answer the same type of question differently based on the current context. An example could be if a user first asked a question about Las Vegas and then asked on what street the Eiffel Tower was located. The context topic could have been set to Las Vegas and the system would be able to determine that the user wants to know where the Paris hotel is located. One other research topic was somewhat related to the list question in the TREC question answer track competition where the systems need to extract the answer from multiple sources. This research topic was more complicated though because it introduced answer verification (see Table 2.1). The QA system would have to be able to resolve several different types of complex relations to render a proper answer.

2.4 OPEN-DOMAIN QA VERSUS RESTRICTED-DOMAIN QA

In this section the benefits and limitations of open versus restricted domain QA systems are analyzed in more detail. Open-domain QA systems can be defined as tools capable of extracting the answer to user queries directly from unrestricted-domain documents. Restricted-domain QA systems are geared more towards providing answers from knowledge bases that cover a specific domain such as student advising or computer repairs. In over a decade, the open-domain QA research has dwarfed the workload that has been devoted to developing restricted-domain QA systems. There are several reasons for this significant difference in focus. One of the main reasons for this difference in focus is the introduction of the internet. By using the internet, billions of indexed documents have become easily accessible through search engines. These documents serve as the main

knowledge base in an open domain QA system. Another strong reason is the addressable market that becomes available with an open domain system versus a restricted domain. An open domain system can facilitate almost anyone searching for simple factoid type questions.

Table 2.1. Example of Answer Extraction from Multiple Sources

<p>Level 1 “<i>Casual Questioner</i>”</p>	<p>Q: When was Queen Victoria born?</p>	<p>Text 1: Queen Victoria (1854, 1889) ruled Britain with an iron fist</p> <p>Text 2: British monarchs: Victoria 1832-1889 Edward 1874-1946 Elizabeth 1923-</p> <p>Answer: 1832</p>
<p>Level 2 “<i>Template Questioner</i>”</p>	<p>Q: How many casualties were reported last week in Fredonia?</p>	<p>Text 1: Last Monday two people were killed on the streets of Beautiville, Fredonia, after a bomb exploded</p> <p>Text 2: The terrorists murdered a family with a small child in Fredonia last Friday, near its border with Evilonia. The father just returned home the day before.</p> <p>Answer: five people</p>
<p>Level 3 “<i>Cub reporter</i>”</p>	<p>Q: How many U.S. households have a computer?</p>	<p>Text 1: Two families in three are connected to the Internet in the U.S.</p> <p>Text 2: Last year, IRS has received 150 million individual return forms.</p> <p>Answer: 90 million</p>
<p>Level 4 “<i>Professional Information Analyst</i>”</p>	<p>Q: Why were there hacker attacks on the computers at University of California, Santa Barbara?</p>	<p>Text 1: U.S. colleges have powerful computing facilities.</p> <p>Text 2: Computer hackers need speedy processors to break security passwords.</p> <p>Answer: To use their computers for password cracking</p>

A large portion of the revenue sources online originates from advertisement. Enterprise companies such as Google, Yahoo!, and Microsoft want to create products that address a wide user segment. None of these companies currently have a strong open domain QA solution, but they are all working hard on different QA solutions. Yet another strong reason that open domain QA systems have received more focus is the U.S. government's QA track competition. This competition has introduced several organizations and universities to open domain QA research.

Open domain QA systems perform well in facilitating the basic need of the casual internet user. However, these systems are not sufficient to provide answers to more complex questions that would make QA become really useful.

2.4.1 Benefits in Open-Domain

Open domain QA systems benefit greatly from the abundance of data available on the internet. The more information an open domain QA system can access and process, the more efficient it becomes (Banko et al., 2002). That the QA system accuracy would increase with more data is based on the theory that if several phrases are found with the supposed answer the likelihood that this is the correct answer increases. Many open domain systems in their initial phase will rewrite a question as an answer, or a part of an answer. The result of this transformation will then be used as a search query. Several open domain systems will also post the entire question as a search. In many cases these simple approaches will produce quality results when finding the correct answers to simple factoid type questions. The reason for this simplicity is that almost any type of trivial question will have a certain amount of coverage online. An example of this would be the question: *Where is the Eiffel Tower located?* One way to rewrite this question as part of the answer would be: *the Eiffel tower is located.* When using this partial answer string as an exact search in Google the first document returned contains the string: *The Eiffel Tower is located in the St Germain district, Paris, France.* The second document contains the string: *The Eiffel Tower is located in Paris, France.* Both these results would have been sufficient in answering the question.

Most of today's open domain question answering systems uses the large quantities of similar or redundant data available online to statistically justify the correctness of an answer. This is done based on the occurrences of the same answer in several documents. Small

answer chunks are extracted from documents. These documents have been retrieved as the result of the initial information retrieval phase following the query expansion. These answer chunks are then co referenced and combined in such a way that an answer is formulated (Morton, 1999).

Several researchers have found that when increasing the amounts of data that are found covering the topic of the question it can reduce the complexity of the algorithms needed to produce a correct answer. Banko et al. (2002) states that “*The more training data that is used, the greater the chance that a new sample being processed can be trivially related to samples appearing in the training data, thereby lessening the need for any complex reasoning that may be beneficial in cases of sparse training data*”.

2.4.1.1 AUTOMATICALLY UPDATED

Many internet resources will receive frequent updates. If designed correctly, an open domain system can benefit from these updates. Automatic updates will limit or completely eliminate the work that would normally be required whenever new or updated information becomes available. This enables the open-domain system to quite effortlessly facilitate questions regarding highly variable content such as news resources. The system maintenance is reduced to frequently scanning existing content resources for updates and additions.

2.4.1.2 BROAD COVERAGE

If the open domain system does not constrain its web search to specific domains there is a high probability that almost any type of topic will be covered to at least some degree.

2.4.1.3 COMPREHENSIVE RESEARCH AVAILABLE

As stated earlier, the effort that has been put into the field of open domain QA research is significant. This will benefit anyone who plans on investing resources in open domain QA system development or research.

2.4.2 Limitations in Open-Domain

This section describes limitations that can be found in Open-domain question answering systems.

2.4.2.1 LARGE

The amount of data covered in most open-domain QA systems is significant, many times exceeding several terabytes. Because of the large amount of data, there are challenges when determining the set of documents to include when searching for an answer. When the query or queries are formed during the initial IR phase of the QA system, it is critical that they are constructed just right. If the queries are too general, too many documents will be retrieved, and the system will not have enough resources to process the resulting documents. This will cause the system performance to suffer and result in unacceptable delays. On the other hand, if the queries are too specific there is a high chance that the correct answer will not be part of the returned document collection. Determining how many documents will be required to answer the question depends on the complexity of the question. If the question is very common, it will result in large amounts of documents being returned. Restricting simple queries will still have a high likelihood that the correct answer is found. Determining the complexity of a question is very hard. *“We do not yet understand how to predict what makes some questions harder than others”* (Kukich, 2000). An example would be the question, *what courses would you recommend me to take?* This question would require knowledge about all courses available, and also possibly information on what focus the student has who is asking the question. The challenges in determining the complexity of a question leaves an open-domain system in a dilemma. The open-domain system works to construct queries in a way that will generally provide good results for a large number of queries. The amount of searchable content available through search engines will likely continue to grow exponentially for some time. This will further impose a more stringent way to construct open-domain queries to target correct answers. The hardware performance of computers is steadily increasing. To improve QA, the algorithms need to become more complex. More complex queries require better query resources that are tied to the searchable content. Additional complexity requires additional storage and processing power; this will not be a commercially viable solution for most open domain QA systems. When the document collection is increasing at the rate of today, open-domain systems will have to devote most focus on efficiently managing very large amounts of data. The focus should instead be directed toward extending the functionality and efficiency of the QA system itself. An open-domain system is limited on its query resources. These resources facilitate simple searches

through the document collection. Features such as named entity taggers (Greenwood, & Gaizauskas, 2003), predictive annotation (Prager, Brown, Coden, & Radev, 2000), and comprehensive indexes are very useful to improve the performance of a QA system. These features rely on synonym extraction and deeper NLP analysis which increases the system storage requirements. This means that an open-domain system cannot construct an extensive feature rich repository of resources. The open-domain system will have to rely on the most efficient features that will have limited storage requirements.

Another problem with a very large knowledge base is related to the way most people ask questions. The first question is usually not that complex, but is then followed by more detailed questions relating to the same topic. With a large knowledge base, an open domain QA system will be limited to the initial questions and will usually not be able to provide answers to more complex follow-up questions.

Almost all open-domain QA systems require that a very large amount of documents refer to the same type of information. This makes it very hard to transition an open-domain system to cover restricted domains. Applying an Open Domain question answering system on a restricted domain it will not have a large document base covering the same information. This will result in the quality of answers decreasing significantly.

2.4.2.2 CONTENT QUALITY

The inability to control the content of the open-domain knowledge base results in several drawbacks. An open-domain system will suffer from misspellings, badly formatted web-pages and text. The documents can also hold malicious or deliberately misleading information. The information can be biased, to present one among competing views. All these scenarios will hinder the open-domain system's ability to find and construct correct answers.

2.4.2.3 INACCURATE CONFLICTING DATA

The main benefit of an open-domain QA system also has its downfalls. Relying on the abundance of data to statistically prove accuracy may result in a flawed approach for several types of questions. The Language Computer Corporation (LCC) has an online version of their NL QA system. When presented with the following question: *What will be the price*

of the Playstation 3? The LCC system responds with candidate answers containing several different prices. The prices are \$399, \$800, \$599, \$499, \$499, and \$600 respectively. The correct answer at the time of the query was \$499. A user would not be able to determine this unless visiting the documents where the prices were extracted from. A user might not be able to determine the correct answer even by reading through the documents the information originated from. This is a good example of an open-domain QA system trying to answer questions about rapidly changing facts. Using multiple sources of unstructured text, and weighing extracted snippets by the number of occurrences and various other criteria, older data might be considered more relevant than new. This makes sense because the Playstation 3 game console was initially thought to be priced around \$800. There was also discussion about trying to match the price against Microsoft Xbox 360 which was priced at \$399. When most web content is not time stamped, or can not be trusted to be time stamped correctly, an open-domain system cannot efficiently include time related information in the answer ranking process. In a restricted-domain system there will usually only be one answer provided. A restricted-domain system is backed by the structured or semi structured knowledge base. If the knowledge base is properly maintained, or provided by a trusted knowledge provider the problem mentioned above would less likely occur. Therefore, the system could easily be fixed.

Open-domain systems that are heavily based on the redundancy of information, will work better for fixed factoid type questions. These factoid type questions are part of the TREC competition (Voorhees, 1999). However, these questions are less likely to be asked by regular users. When the question is directed towards a restricted domain, the open domain QA system will most likely mislabel it as a general question and provide an inaccurate answer.

2.4.2.4 MAINTENANCE

Open domain systems that rely on a knowledge source that is not controlled by the staff administering the system, will always suffer from maintenance problems. This model fits most open-domain QA systems. If a resource needs to be extended or updated but resides outside the direct control of the system, it will end up causing problems. Modifying an open-domain system to handle these types of scenarios is hard. It becomes problematic adding

additional knowledge to the system. There is no easy way to make sure the system learns about new facts. The system will not be able to provide accurate answers if the only way to learn new facts is when third party providers update their content.

2.4.3 Benefits in Restricted-Domain

In a restricted-domain QA system there are a lot of specific properties that can be exploited by means not possible in open-domain QA. A question answering system requires understanding of natural language text and the QA system requires much linguistic and common knowledge for answering correctly. One way to improve the accuracy of a question answering system is to restrict the domain it covers. By restricting the question domain, the size of the knowledge base shrinks and several different methods to efficiently process questions becomes available. Many of these methods would be too process intensive to apply in an open-domain QA system.

2.4.3.1 TECHNICAL TERMS

In some restricted domains, many of the questions can only be answered correctly by experts. These types of restricted domains are usually associated with a large amount of technical terms. These technical terms might exist only within that particular domain. Within open-domain QA systems these terms are usually discarded as misspelled words or words where no type or word-sense can be associated. Within restricted-domain QA systems these technical terms play an integral part of determining the answer to a question. Technical terms can be included in domain specific word-lists and lexicons. These resources hold information on what the term means and how it is associated with other terms within the restricted-domain. In this manner the results from a query can be greatly improved compared to those terms being discarded. A restricted-domain QA system can use this information to generalize terms or find alternative terms to use in the query expansion and answer selection processes.

2.4.3.2 ACCURACY IN UNDERSTANDING CONTENT WORDS

Restricted domains will naturally impose a restriction on the meaning of polysemous words. Polysemous words are words that can have several different meanings. There is a much higher chance that a basic bag-of-word approach will render quality results.

Polysemous words such as the word *interest* can have several meanings in an open-domain QA system. This word could mean interest in taking some course work, or the interest on your bank account. This requires an open-domain to more carefully control the query expansion to limit the search results based on what was asked. Polysemous words are quite common, “*approximately 20% of the words in WordNet are polysemous*” (Hung, Wang, Yang, Chiu, & Yee, 2005). Within the restricted student-advisor system, the word interest is highly unlikely to represent bank account interest which increases the chances of finding more relative information. This will limit the complexity needed for analyzing the questions. Therefore, additional processing resources can be used elsewhere. This fits well with the shallow parsing methods that are common for question answering algorithms.

2.4.3.3 SPECIALIZED VOCABULARY

In a restricted-domain QA system, it is easier to create a lexicon or an ontology that covers the knowledge base and the domain. This is due to the limited vocabulary that needs to be covered. If system is able to quickly access information about domain specific words, it is possible to use inference to determine the meaning of certain words within the context of the question.

2.4.3.4 ACCESS TO DOMAIN SPECIFIC LINGUISTIC RESOURCES

In an open-domain QA system any type of specialized resources need to be general. With general resources the system can provide enhancements across any domain. In a restricted domain, it is possible to build lexicons, dictionaries and ontologies that target a specific domain. There have been many successful restricted-domain systems developed in various fields such as biomedicine (Zweigenbaum, 2003). These resources can help both with analyzing questions and constructing answers.

2.4.3.5 STRUCTURING OF THE KNOWLEDGE BASE

When documents or resources that constitute the knowledge base of a QA system are stored locally, it can benefit the system. The access and structural layout of the documents can be changed to better facilitate the design of the QA system. This is much harder to do in

an open domain system where almost no control can be imposed on the knowledge base; this is due to the knowledge base being provided by third parties. An open-domain system can write an intermediate tier where this structure could be created programmatically. Creating an additional tier for this purpose will hinder performance. In a restricted-domain QA system, there are special components that are designed to parse specific documents within that domain. Completely structured data in the form of databases are also common. This is the case of the student-advisor system, where most of the knowledge base resides within a database.

2.4.3.6 QUALITY AND RELIABILITY

Quality and reliability are the greatest advantages of restricted domain QA systems. When the system can control its knowledge base, the quality and reliability of the answers will increase. Resources that contain content that is constantly updated is treated with less confidence and subject to more frequent verifications. If the data in the knowledge base is formalized this will improve reliability even further. A restricted-domain QA system can be designed to certify that the information provided is accurate and correct. This would be close to impossible in an open-domain QA system.

2.4.3.7 ANSWER FORMATTING

Restricted-domain QA systems allows for better control on how answers get formatted, especially in the case of the student-advisor, where answers are completely custom formatted based on the question. Custom formatted answers are preferred over programmatically generated answers because the answer can include some context. Users are found to prefer some context compared to an exact answer. That users prefer more verbose answers was concluded in a research effort by Lin, Quan, Sinha, Bakshi, Huynh, Katz, and Karger (2003) "*users prefer paragraph-sized chunks of text over just an exact phrase as the answer to their questions*". From the same study it was also found that only 3.33% wanted an exact answer, 53% wanted a paragraph, 20% wanted a sentence, and 23% wanted a whole document. These numbers would indicate that open-domain QA systems most likely leave the user wanting more information. Their very limited ability to provide more than just factual answers is unfavorable to most users.

2.4.4 Limitations in Restricted-Domain

This section describes limitations that can be found in restricted-domain question answering systems.

2.4.4.1 SCARCITY OF DATA

The abundance of data is something all open-domain QA systems rely on. Several passages that relate to the question are extracted and non frequent passages are discarded when constructing the final answer. *Experimental results show that question answering accuracy can be greatly improved by analyzing more and more matching passages* (Dumais, Banko, Brill, Lin, & Ng, 2002). In a restricted-domain, that luxury is usually not available. The scarcity of data makes it hard to write IE algorithms that determine which information extracted relates to the question. However, restricted-domain QA systems such as ContextQA will present fewer problems due to matching questions to existing QA pairs. It is not of great importance if the answer is represented multiple times or not, especially if there are multiple questions leading to the same answer.

2.4.4.2 COMPLEXITY OF QUESTIONS

In many restricted domains questions tend to be more complex than the questions handled by open domain. Therefore the questions are in many cases more verbose and longer. With this follows that the process of providing answers also becomes more complicated.

2.4.4.3 COMPLEXITY OF ANSWERS

More complex questions usually require more in depth answers. Generating a properly formatted answer is one of the major problems in both open-domain and restricted-domain QA systems. A complex question can be a why or a how type question. These types of questions might need to compare different properties or provide an in depth explanation on how to do something. The problem of automatically generating an in-depth answer in the ContextQA system is not much of a problem. If the question is correctly matched with one of the existing questions in the knowledge base, a well-formed answer is guaranteed to be available.

2.4.4.4. EVALUATION OF RESTRICTED-DOMAIN QA

The limited amount of analysis using restricted-domain QA systems makes it challenging to accurately evaluate its performance. Open-domain systems have the advantage of having several common metrics and question sets that can be analyzed and compared against. Some researchers developing restricted-domain QA systems have found this problematic and have suggested several enhancements. These enhancements would separate restricted-domain QA evaluation from open-domain, *“Simply applying the open domain QA evaluation paradigm to a restricted-domain system poses problems in the areas of test question development, answer key creation, and test collection construction.”* (Diekema, Yilmazel, & Liddy, 2004). Any restricted domain QA system will be targeted towards a specific audience. Thereby the system might require different evaluation methods based on the target audience. The restricted domain evaluation extends beyond the domain specific questions and should incorporate tests that show how well it fits its target audience. To standardize this test process becomes difficult.

2.4.4.5 COST OF MAINTENANCE

By not having automatic updates done by third party resources, the maintenance of a restricted-domain QA system is left to the administrators. The problem of maintenance will become more significant in an environment where there is a high rate of change. The knowledge base can be complex, covering proprietary information that requires knowledgeable administrators. Some systems might require constant maintenance to conform to new data sources. All these aspects will increase the maintenance cost. Keeping the QA system small enough to be maintainable but still useful is a delicate balance which requires experience. A smaller company or organization may have minimal resources to hold a large knowledge repository. In many cases the repository would then cover areas that are known not to change very often. Smaller companies usually require outsourcing to properly implement a restricted-domain QA system. Implementing a restricted-domain QA system is a significant project with large up front expenses. A project like this can also yield an uncertain return which acts as a deterrent to small businesses.

2.5 THE CHALLENGES OF QUESTION ANSWERING SYSTEMS

We do not yet understand how to predict what makes some questions harder than others (Kukich, 2000). One of the hardest aspects of question answering is complexity of the language the questions are formulated in. The English language by itself has hundreds of thousands of words and that is just when using a common dictionary. Then you have to add domain specific terms in a restricted-domain question answering system. Spoken languages and communication is mainly designed for humans and not computers. Computers could more easily communicate using a much more optimized language to better suit its architecture. Given that in today's world humans have daily interaction with machines creates a need for better and more innovative interfaces between the two. There is a reason why this communication hasn't gone very much further than simple instructions. The problems of making a computer understand spoken language are not trivial. Different words mean different things dependent on what the context is. Things that make it more complex are slang, bad grammar, and spelling errors. Humans can quite easily get past these types of problems given the great ability of understanding patterns. We can also apply our world knowledge to these patterns to be even more efficient.

The hardest types of questions have been determined to be Why and How type questions. To provide an answer to these types of questions you usually need knowledge about everything the question relates to, and not only the direct context of the question. You might also need experience from several other topics which have to be combined and analyzed. A good example would be: *Why is Question Answering Hard?* To properly be able to provide an explanation on why writing question answering systems are hard you need to know almost everything there is to know about question answering. In the ContextQA system more than 25% of the questions are How and Why type questions. This is possible partly due to the design of the knowledge repository and limiting the problem scope.

Another aspect that makes Question Answering hard is that the problem space intersects with several of the more complex research fields (see Figure 2.5) in computer science.

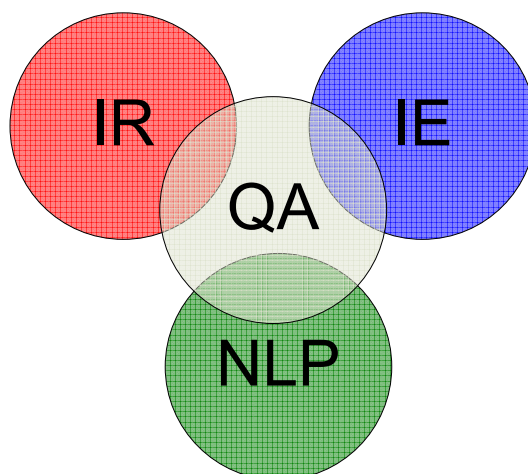


Figure 2.5. Intersecting QA research fields.

These fields are mostly Natural Language Processing (NLP), Information Retrieval (IR), and Information Extraction (IE). The positive aspect to this is that QA systems will benefit from advances in any of these fields.

The complexity of QA can easily be underestimated. The TREC competition shows this where new more advanced QA methods that extend past retrieving factoid type answers have not made much progress. There have not been any significant new results for quite some time. Because QA can be seen as part of NLP it will to a certain degree have the similar problems as NLP such as being able to understand world concepts. The IR portion of QA also introduces a whole slew of complex problems that need to be addressed. When a question answering system retrieves its initial set of documents to extract answers, the noise ratio can be quite high. The noise ration will be more limited in a restricted-domain system than in an open-domain one. However, in both types of systems there is a significant chance of getting irrelevant results even with the right set of search keywords. The QA system needs to be constructed in such a way that it can quickly sort out data that is not relevant to the user queries. Regular IR usually removes all stop words and only considers what is left. Stop words are words such as *is, are, he, which*, etc. Using only the approach of removing stop words is not sufficient in QA because just subtle changes in a sentence can completely change its meaning.

1. Why should I have an advisor? (repository question)
2. When should I have an advisor? (client question)

3. Where should I have an advisor? (client question)

After removing stop words we are left with the words should and advisor, this qualifies the repository question for both alternative two and three but if the question only address why the student should have an advisor the answer would not be correct. Ideally the word When or the word Where should automatically invalidate alternative two and three.

Finally, when information is extracted and an answer is formulated one of the most difficult tasks for a QA system is to determine if that answer is in fact correct. In a restricted-domain QA system this problem is usually limited since the chance of erroneous data decreases with the size of the domain. An open domain usually tries to tackle false facts by correlating several documents against each other. This way decisions can be made on what is correct and what is wrong.

2.6 ATTRACTIVE MARKET

Because there are so many different ways of applying question answering to various markets, several companies are developing a wide range of different QA systems. Spoken dialog systems are getting more advanced where a person can call in and ask questions over the phone. These systems can be anything from a bank teller system, or a directory service. The potential savings for companies solely focusing on these types of tasks can be significant. In 2001 Forrester Research estimates that the cost of manually answering a customer inquiry by phone to be \$33, e-mail \$9.99, and a web self-help system \$1.17.

Online QA systems still remain more collaborative in nature. Users submit questions and other users answer their questions. Some companies like Ask.com have integrated support for natural language questions in their search engine. Ask.com has also sold this technology to companies like E*Trade and Toshiba. A large percentage of people are using search engines when looking for answers online. A significant portion of online revenue is made based on search engine advertisement. It is natural that QA systems have become a very attractive problem to solve.

CHAPTER 3

SYSTEM DESIGN

This chapter gives a design overview of the ContextQA system. The QA roadmap research (Burger et al., 2000) was part of the inspiration when designing the structure of the QA system, and what problems to solve. The roadmap research document covers more or less how a flawless QA system should work. The document authors describe the many tasks, problems and techniques that need to be addressed to produce such a system. The roadmap document led to several of the decisions made in the process of compiling the final set of research topics included in the ContextQA system. Currently the QA research community is stagnant and the roadmap is already failing. The complexity of producing quality QA systems exceeds the commercial ability to create such systems.

3.1 LIMITING THE PROBLEM SCOPE

The field of Question Answering Systems has proven itself to be one of the hardest fields in Computer Science to solve. Many projects have failed due to the inability to properly understand the magnitude of the problem. In a project like this the scope needs to be carefully planned so that the results produced can still contribute to the research community. Without the backing resources available to large corporations this becomes that much harder. Given these facts I have tried to come up with a unique approach in designing a complete question answering system that uses some interesting approaches to solve some of the hardest problems.

3.1.1 Restricted Domain QA

The ContextQA system is focused on restricted-domain QA. The decision to target the restricted-domain is based on the earlier evaluation where I compare open-domain to restricted-domain QA system. Restricted domain QA has many of the same problems open domain QA has to tackle, but it requires a significantly smaller knowledge base. This decision limits the resources required to produce quality results.

Running a restricted domain QA system you do not have to rely completely on third party solutions such as online search engines but rather maintain the knowledge base locally. The process used to collect and build the ContextQA system knowledge base is mostly automated. To be able to automate building the knowledge base semi-structured web documents containing QA pairs are used to kick start the information gathering process. This provides a rapid way to produce a restricted domain knowledge base in a short amount of time. The information gathering to build the knowledge base is described in more detail in the question harvesting section.

3.1.2 Shallow Language Understanding

The main knowledge base of the ContextQA system is based on a semi-structured repository of QA pairs. The ContextQA system uses various shallow language parsing algorithms to match incoming questions against the repository of questions in its knowledge base. Shallow language parsing is different from complete parsing in that it usually only considers portions of a text, or chunks. Not having to break down the entire sentence into a syntactic parse tree or similar structure increases performance. A sentence such as *Where is the cashiers office located*, could be broken up in the following way, *[Where is] the [cashiers office] [located]*. With those chunks the system can understand that *[Where is]* and *[located]* hint that the answer is most likely a location. If the second chunk *[cashiers office]* also is included, shallow language algorithms can determine matching questions based on that information. Any question that is of type location and has the chunk *[cashiers office]* in it would be a likely candidate to try and match with.

3.1.3 Completeness of Answers

With completely formulated answers in the knowledge base the system can guarantee that a well formed answer exists if a question qualifies as a match to an existing question in the repository. Having well formed answers greatly reduces the complexity of the QA system, and enables the answers to exceed the quality of most existing QA system. Matching questions against other questions separates the system from common QA systems where information extraction, and answer formulation is done. A question answering system that constructs the answer programmatically will many times suffer from badly worded

responses. Given that most QA systems at the time can only produce simple factoid type answers complex answers that require a large body of text are not even remotely feasible. Not having to perform information extraction and answer formulation limits the problem.

Another benefit with having QA pairs is that anyone could quite easily extend or modify the knowledge base of the system without being required to have any particular knowledge about question answering system design. This way several departments in an organization which is implementing the QA system can easily work in parallel to extend the system.

3.2 ADVANCED CONCEPTS

This section lists some of the more advanced concepts that the ContextQA system utilizes from that of a standard QA system.

3.2.1 Interactive QA

Some of the more advanced topics that have not effectively been tackled in QA systems to date include interactive QA. Interactive QA means that the system can converse with the user and keep track of what has been said earlier. One of the best conversational systems in the world is A.L.I.C.E (Artificial Linguistic Internet Computer Entity). The A.L.I.C.E system uses a markup language to describe its knowledge base. This markup language is named AIML (Artificial Intelligence Markup Language). AIML is a derivative of XML and describes units that hold topics which contain categories. Every category has pattern elements which are used to match against a user's input. The patterns can support wildcards. The system converts the patterns into a tree structure (see Figure 3.1) that is used for matching incoming sentences. As the figure shows, a random response can be used if several responses have been specified. The algorithm is a restrictive version of the depth first search. Dr. Wallace states that the branching factor for the first node is about 2000 for 20,000 categories. Then the average branching factor goes down to 2 and for each new branch it decreases further. An incoming question is broken up into words and then matched against the tree structure until an answer is found. The algorithm can be described as follows:

1. Given:
an input starting with word X, and a graph:

2. Does the current node contain the wildcard key `_`? If so, search the sub graph rooted at the child node linked by `_`. Try all remaining suffixes of the input following `X` to see if one matches. If no match was found, try:
3. Does the current node contain the key `X`? If so, search the sub graph rooted at the child node linked by `X`, using the tail of the input (the suffix of the input with `X` removed). If no match was found, try:
4. Does the current node contain the wildcard key `*?` If so, search the sub graph rooted at the child node linked by `*`. Try all remaining suffixes of the input following `X` to see if one matches. If no match was found, go back up the graph to the parent of this node, and put `X` back on the head of the input.
5. If the input is null (no more words) and the current node contains the `<template>` key, then a match was found. Halt the search and return the matching node.

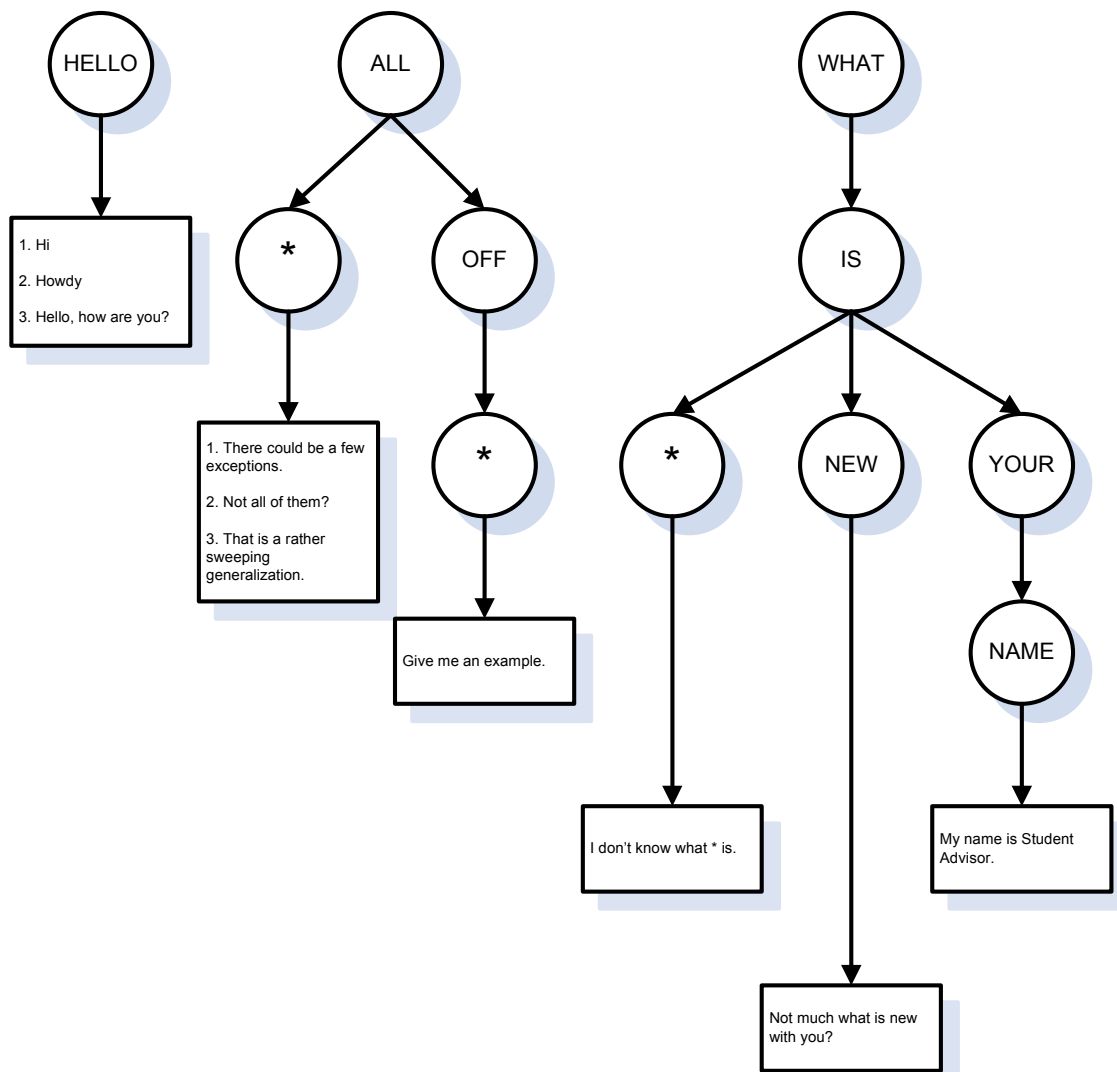


Figure 3.1. A.L.I.C.E tree structure.

The wildcard _ means that the sentence is starting with the text replaced with _. The wildcard * can be replaced with any phrase from the sentence. Given A.L.I.C.E's straightforward design it is still more advanced than any commercial conversational software (Bush, 2001).

My first intention was to completely base the ContextQA system on an enhanced version of A.L.I.C.E. However, using A.L.I.C.E requires an extensive use of wildcard and manual updates to produce quality results. Taking a base set of questions and converting them to AIML without wildcards results in a close to useless system because it is the same as performing exact matches directly against the questions themselves. To be able to increase the quality each question would have to be analyzed, grouped and modified to utilize the more advanced functionality provided by A.L.I.C.E. These additional features are things like context specific attributes set by particular questions and answers. My second thought was to expand questions based on synonyms and different wording to the maximum number of permutations possible into the tree. Designing a QA system this way with completely static questions might work well in trivial scenarios with a very restrictive domain. When the QA domain is expanded the memory requirements of the tree will increase exponentially. Because of the limited performance of this approach it was abandoned early in the research. In a more complex environment where the system should cover a larger knowledge base the requirement for a more dynamic algorithm design becomes a necessity.

Instead I decided to use A.L.I.C.E as a fallback if the QA algorithm confidence was too low. What this means is that if the confidence is low the answer produced is unlikely to be the correct one. This way a user will still get an answer or a continued discussion. The base conversational AIML repository included with the A.L.I.C.E distribution is used for this purpose. The repository has been slightly modified to fit that of a student advisor. It can be argued how useful this is to an end user, but given that most people want to speak with an actual person when requesting support, QA systems will have to improve in this area to become commercially viable.

3.2.2 Automation

When designing the ContextQA system a great deal went into thinking about automation. With the complexity of QA you want to limit the amount of work initially setting

up a system that covers a completely new topic. Using resources to bootstrap almost any domain knowledge is described in more detail under the section “building a targeted knowledge base”.

3.2.3 Responsiveness

System responsiveness was one of the topics in the QA roadmap. A QA system needs to be close to real-time to be useful. This was also considered when designing the ContextQA system. Having a real-time system will many times limit the type of algorithms that can be used. This includes fast models of retrieval in the information retrieval phase. Answer extraction must be fast and the reasoning mechanism must also execute quickly. That the answer extraction in the ContextQA system is fast comes naturally because of its design. The system should also be able to scale easily. All of these different performance sub tasks are addressed in the research and system design.

3.3 PREVIOUS WORK

There has not been a significant amount of work previously done that is similar to the QA system developed as part of this thesis. Much more work has been devoted to open domain question answering. There has been work done previously using frequently asked questions and answers as the knowledge source. One such system is FAQ Finder (Hammond, Burke, Martin, Lytinen, 1995). The FAQ Finder system bases its knowledge on UseNet which I described earlier under the section Early Question Answering utilizing Networked Computers. The system should be classified as an Open Domain system but it can still be compared to the work in this research for its use of question and answer pairs. FAQ Finder will use START in its information retrieval phase. START is used to retrieve FAQ documents that contain terms used in the user’s question phrase. The system uses a couple of different algorithms; these algorithms will determine if the user entered question matches any of the QA pairs in the FAQ documents returned. Statistical similarity, Semantical similarity, and Coverage are used. For statistical similarity tf-idf (term frequency – inverse document frequency) is used. Semantic similarity is done by analyzing each word in the user’s question and its semantic relationship with other words in the FAQ. The coverage is the percentage of query terms that intersect with the user’s query. AutoFAQ (Whitehead,

1995) is another QA system that uses FAQ documents. In the case of AutoFAQ, the FAQ documents are also retrieved from UseNet, but maintained locally instead of searching external resources. There is not much coverage on how AutoFAQ extracts answers. The system compares the question a user inputs against the local repository of QA pairs to extract an answer.

Other work using frequently asked questions is Automated FAQ Answering (Sneiders, 1999). The Sneiders QA system uses a method he calls Prioritized Keyword Matching. Three different types of keywords are used in this process. These three keyword types are required, optional, and forbidden keywords. Each of these keyword types is associated with an answer. The primary keywords have to be present in the input question to further match it against a FAQ entry with those primary keywords. The optional keyword does not have to be present. If several optional keywords do not intersect between the FAQ entry and the input question, it will not be considered a match. The number of optional keywords that cause a match to fail is configurable, but usually set between 0 and 1. Irrelevant keywords are removed from the question. Irrelevant keywords would be the same as stop-words. If the input question has any of the forbidden keywords listed in a FAQ entry, it would automatically fail the comparison. The keywords are associated with FAQ entries in a manual fashion by the administrator of the QA system. This means that the administrator of the QA system needs to be very familiar with the system domain. The administrator also needs to be very familiar with linguistics. The administrator will only then be able to determine what keywords are relevant to certain FAQ entries. Sneiders also brings the restricted domain dictionaries to an extreme. Each FAQ entry can have its own dictionary. This includes things like singular, plural form, different spelling, split, merged form, and synonyms. This information is also set up manually, but it is not required. The Prioritized Keyword Matching algorithm analyzes the incoming question against features that are associated with an answer. The algorithm does not analyze the answers, and does not have any questions coupled with the answers in its knowledge base.

CHAPTER 4

BUILDING A TARGETED KNOWLEDGE BASE

A QA system designed to provide the same information that is usually obtained from a Computer Science Student Advisor should be categorized as a targeted knowledge base. Such a system would pose a challenge to an administrator implementing the QA system if he is not familiar with this particular knowledge field. This unfamiliarity would make it hard to determine what the most frequently asked questions are. Given this targeted area of expertise, the knowledge base would best be derived from an actual repository of the most commonly asked questions that students actually pose to their advisor. To have actual questions would be preferable compared to building a knowledge base from questions that are just thought to be common ones.

One benefit when modeling a student advisor is that there are thousands of student advisors around the world. Today many Universities have started to use the Internet as a resource to alleviate the number of questions asked to advisors. This is done by listing frequently asked questions on University web pages.

Given the many lists of frequently asked questions available online, I determined that this would be a very good start of my knowledge base for the system in my research. However, the vast amount of questions to harvest tends to be a tedious task. This task becomes difficult given the wide variety of different formats that schools use on their web pages. Many times a question can be listed within a larger body of text where big portions of the text need to be excluded from the final result. After researching several websites, I determined that a system that could automatically extract questions would be the ideal solution. This system would also provide a natural building block of a complete QA system solution. The system would take a URL as input and proceed to parse the document that URL leads to. It would extract and harvest questions and answers with the supervision of an administrator. This way it is easy even for a novice administrator to build a targeted knowledge base covering some common field. This cannot be done for every single topic, but as of Jan 1, 2004 there were 194 million registered domain hosts on the internet. Google had

indexed 8 billion pages as of 2005. With the exponential growth pattern of the internet there is a high likelihood of finding these types of resources for almost any topic.

An open-domain QA system does not have the luxury of being able to process web-documents to the degree of the student-advisor system. Open-domain question answering systems usually perform a certain amount of work of indexing into existing documents and extracting certain data points. This process will facilitate quicker searches which will reduce the time to present that answer. It will also increase the quality of the answer. However, in an open-domain system this has to be done in a very shallow fashion due to the storage and processing requirements. To generate a more comprehensive knowledge base to cover all specific fields is not feasible. This is one reason why current open-domain systems mostly deal with factoid type answers. A good example of a system like this is the predictive annotation QA system developed by Prager, Brown, Coden, and Radev (2000). This system processes a very large document base to extract various data points referring to factual information. It also indexes into these documents so further processing can be done during the answer retrieval. This system has several different ways to classify and rank the collected information but suffers from the same flaws as all open-domain systems suffer from. The documents might not be formatted sufficiently to describe certain facts that are critical to correctly describe the content of the document. The facts could contradict each other. If the document contains longer more descriptive information these cannot be extracted correctly with this method. Most open-domain systems will not have a way of building and indexing more descriptive answers. One reason that many open-domain systems are designed to handle factoid type questions such as who, what, when, and where questions are that it is much easier than dealing with how, and why type questions. Another reason is that many researchers have been motivated by the question answering track of the Text REtrieval Conference (TREC). The types of questions in the TREC question answering track are a misrepresentation of questions that people usually pose. Table 4.1 shows the question type distribution in TREC as described by Moldovan, Pasca, Harabagiu, and Surdeanu (2002). However, when looking at 200 questions posed to the collaborative answers service provided by Yahoo!, the distribution of how type questions are closer to 50%. Building a system that automatically collects verbose answers for complicated questions would eliminate the need of having to combine chunks of information to construct an answer.

Table 4.1. Distribution of TREC QA Track Question Types

Type	Number (%)
Class 1 (Factual)	985 (67.5%)
Class 2 (simple-reasoning)	408 (27.9%)
Class 3 (fusion - list)	25 (1.7%)
Class 4 (interactive – context)	42 (2.9%)
Class 5 (speculative)	0

4.1 QUESTION HARVESTING SYSTEM

The remainder of this chapter will cover the design and implementation details of the question harvesting system. A system that automatically retrieves questions and answers from web based content needs to have certain features to be able to control what is being collected. If there is no control, certain questions might be extracted that have no meaning in the context of the target knowledge base. The system could address this problem by having prior knowledge about what to collect. In this case the system is building the knowledge base that would be required to make these kinds of decisions. Thus, these types of decisions are left up to an administrator.

4.2 SYSTEM DESIGN

The question harvesting system is implemented as a web application with the Model View Controller (MVC) software architecture in mind (see Figure 4.1). This architecture is a good way to implement web applications. By design it separates business rules and data handling from the logic that is responsible for rendering the user interface. This separation also promotes code reuse which is preferable in any larger project. I picked Java as the programming language to implement this system because it works really well when interacting with web resources. Java is usually a good choice if the system is not required to handle any significant processing load. With Java you can quickly develop a working prototype without having to devote too much time to details. The system runs within a Servlet container and can therefore be installed on almost any standard web server.

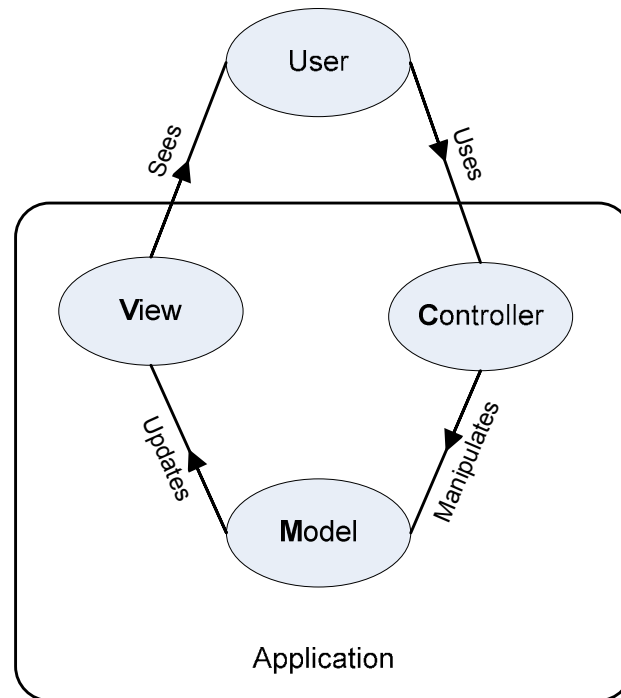


Figure 4.1. MVC software architecture.

The user interface for the FAQ parser is a wizard interface that guides the administrator through a set of pre-defined steps. In each step he is prompted for various decisions and formatting tasks. In the final step the resulting question and answer pairs are permanently stored in a relational database. The resulting questions and answers can later be accessed by administrative tools, and the main question answering application.

4.3 CONTROLLER SERVLET

The controller is implemented as a Servlet. This Servlet controls what content should get produced based on the current state of the application. The application state is in turn controlled by attributes that are passed back from the client web browser to the controller. The controller Servlet class is named UI, and its UML representation can be viewed in Figure 4.2. The UI Servlet controls a set of classes that are responsible for producing the content that will be fed back to the client web browser. Each page or application state is implemented by extending the Servlet-page class (also depicted in Figure 4.2) which is an abstract class. Each Servlet-page implementation needs to at least implement the execute method. This method should in turn generate content to be pushed back to the client.

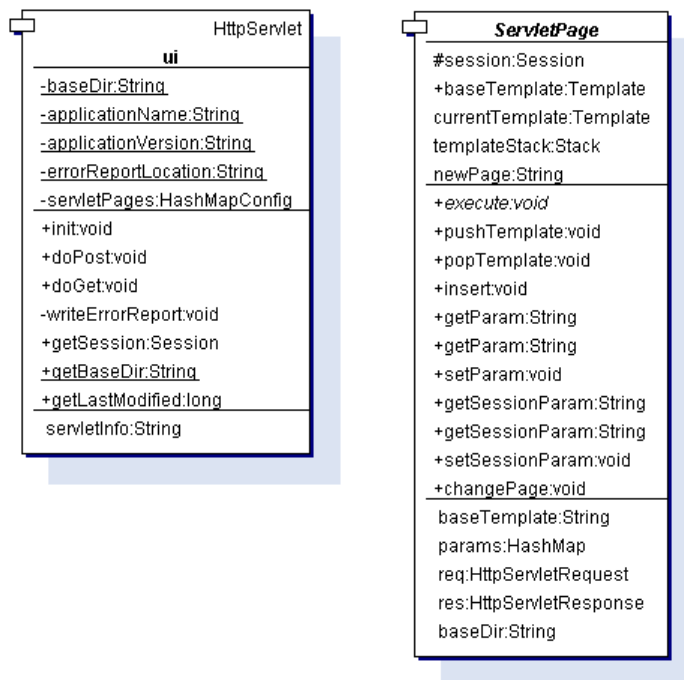


Figure 4.2. Controller Servlet UML diagram.

The UI Servlet will load and instantiate one or several Servlet-page implementations through a dynamic class-loader for each client request. The UI Servlet also provides the Servlet-page implementation with session state for each client which is kept server-side. The session state is where application specific parameters are stored and maintained. The session is tied to each unique client by setting a cookie in the client web browser. The Servlet-page implementation can read and modify session parameter names and their values during its execution.

The Servlet-page class makes the process of generating content more streamlined for its implementation classes. Simplicity is accomplished by providing a suite of methods that makes it easy to generate complex text documents. This is done by the use of text templates. A text template is a text document that can represent HTML, SQL, XML, or any type of document that can be formatted as plain text. The templates are stored on disk and the Servlet-page class utilizes a template factory class to gain access to these template files. A Servlet-page implementation class can issue a push template method call with the name of a template file. This method will cause that file to be loaded and set as the current active template. This type of call would be done in the Servlet-page execute method call. If this

template is a static HTML file that was chosen because of a certain application state the execute call can end here. The content of that template will then be pushed back to the client.

The template files also support a text-tag that can be replaced by calling the Servlet-page insert method call. The insert method call takes a tag-name and an object in its method signature and will replace the template tag with the resulting text of that object. This can be done multiple times for each template tag and each template can have multiple insert tags. What makes it more dynamic is that the Servlet-page supports to push a template into another template and set that new template as the current active template. Complex hierarchies of templates can be constructed by issuing push and pop template calls. The Servlet-page also supports switching to another Servlet-page implementation by calling the change-page method. An example of this would be when a user tries to login to an application and the login is successful. Instead of performing a redirect roundtrip to the client the system can automatically switch to its new state. The Servlet-page implementation also has access to all URL parameters. These parameters are passed to the UI servlet by the get-parameter methods.

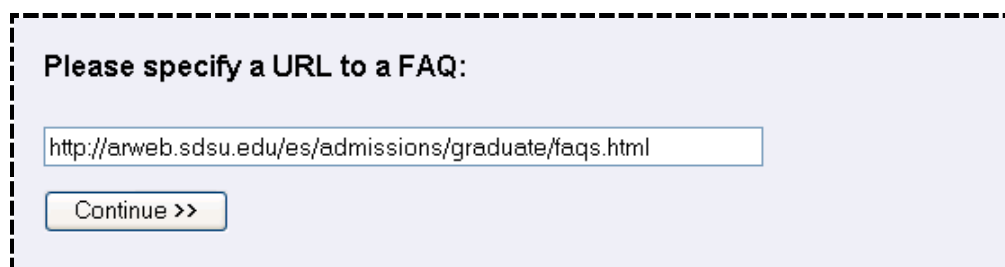
4.4 WIZARD INTERFACE

This section describes the wizard interface that guides the administrator through the process of building the initial knowledge base using URLs to FAQ documents found online. The wizard interface is implemented by extending the servlet-page class described in the previous chapter. The wizard is setup as a three step process which is described in more detail below.

4.4.1 Wizard Interface Step One

In the first step of the wizard the administrator is presented with a screen where he is prompted to enter a URL that leads to a FAQ document. Before starting the wizard, the administrator would have collected several URLs that lead to documents that ideally have a comprehensive list of questions and answers. These questions and answers should match the context of the knowledge base that the administrator is trying to build. The URL can lead to either HTML formatted pages or text files. The parser supports both these document types. If the URL is not formatted correctly or leads to a page where no question can be extracted, an

error message will appear. This error message will describe the problem and the administrator will have to adjust the URL or try a different one. The URL that is shown in Figure 4.3 is one that I used when building the knowledge base for the ContextQA system. In this process I ended up using more than one hundred FAQ documents from all the major colleges in USA and other English speaking countries.



Please specify a URL to a FAQ:

Figure 4.3. Step one of the FAQ parser wizard.

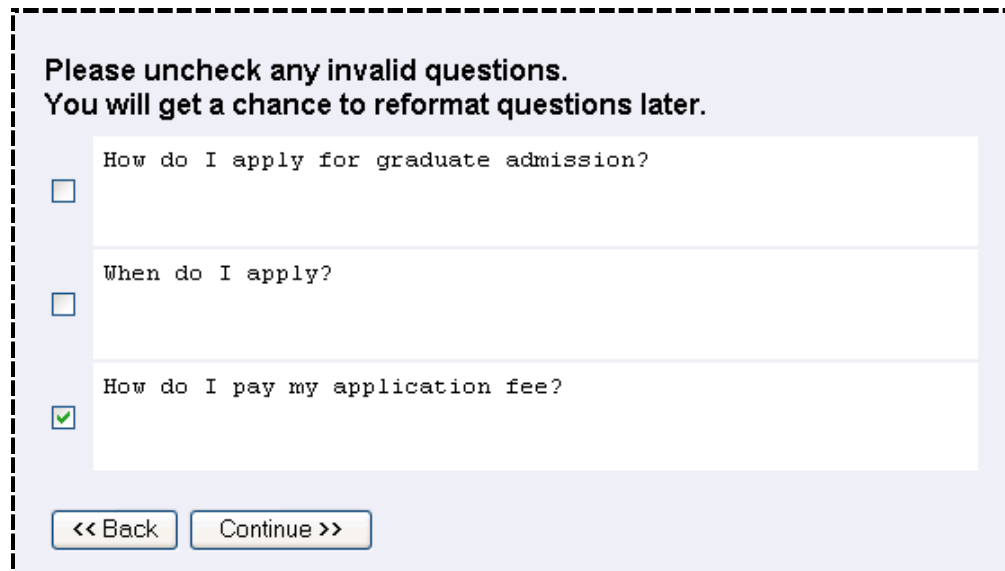
4.4.2 Wizard Interface Step Two

In the second step (Figure 4.4) of the wizard all questions that have been extracted from the FAQ document will be listed on the screen. All questions are displayed with check boxes next to them which are all initially checked. On this screen the administrator is able to determine how well the questions were extracted from the document that he specified in step one. If the results are not satisfactory the administrator has the option to go back to the initial step of the wizard. If any of the questions listed on the screen are not applicable to the knowledge base that the administrator is trying to build, he should uncheck those. This will cause them to be excluded from the final set of questions that are permanently stored in the knowledge base. When a complete set of questions have been selected the administrator can click continue to go to the final screen.

4.4.3 Wizard Interface Step Three

In the third and final step (Figure 4.5), the questions with associated answers will be listed on the screen. In this step the questions and answers can be edited for correctness. This feature is made available so that the administrator can adjust the questions and answers to produce exactly what he wants to store in his knowledge base. In this final step the

administrator will have one last option to exclude question and answer pairs. This is done by un-checking the check box next to the question.



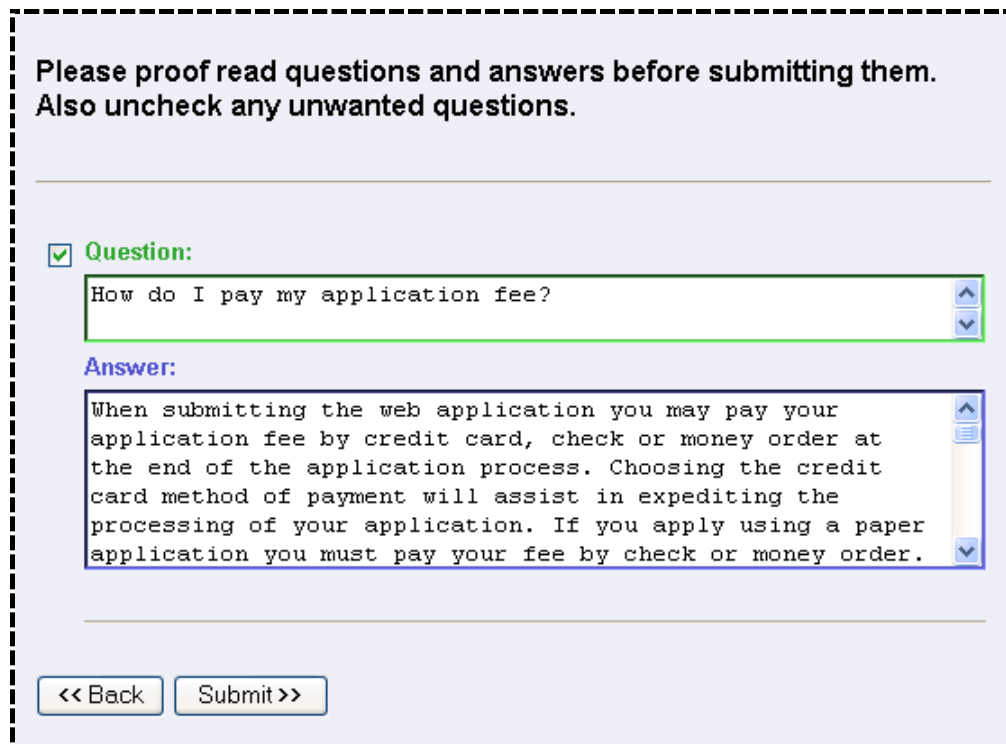
**Please uncheck any invalid questions.
You will get a chance to reformat questions later.**

How do I apply for graduate admission?

When do I apply?

How do I pay my application fee?

Figure 4.4. Step two of the FAQ parser wizard.



**Please proof read questions and answers before submitting them.
Also uncheck any unwanted questions.**

Question:

How do I pay my application fee?

Answer:

When submitting the web application you may pay your application fee by credit card, check or money order at the end of the application process. Choosing the credit card method of payment will assist in expediting the processing of your application. If you apply using a paper application you must pay your fee by check or money order.

Figure 4.5. Step three of the FAQ parser wizard.

When finished the administrator would click submit or back if he is not satisfied with the result. Submitting the questions and answers will cause them to be permanently stored in a database. The administrator will be brought back to step one of the wizard where he can enter a new URL or leave the interface.

4.5 DOCUMENT PARSER

The document parser is responsible for locating and extracting questions and answers from the FAQ documents. The parser needs to be able to determine when a question starts and when it ends. This might not seem too difficult at first; however the large amount of badly formatted web pages, different designs, document types, and layout of these documents, complicates the situations. The collection algorithm to adopt for these stochastic environments can slowly evolve to be able to handle almost any type of document. Dealing with documents containing errors and documents that are constantly changing does not affect this particular system. Having a controlled collection mechanism will let the administrator sort out any flawed information early on in the collection process. This way the final semi-structured knowledge base will have a very low error rate. However, when querying an open-domain QA system documents many times have to be re-processed to extract additional information for complex queries. This increases the risk of trying to extract information from documents that now have changed or include errors.

4.5.1 Parser Design

The first thing the parser does is to connect to the database where the QA knowledge base is stored. This connection is used to determine if the URL has been parsed previously. If the URL has already been parsed, this will trigger a message to be presented to the administrator of the date when the URL was last processed. In any case the parser will retrieve the document from the URL and the administrator will have the option to process or re-process the document. The document can be either in plain text or HTML format. The document type is usually specified in the HTTP header returned from the web server. The parser does not determine the format of the document by analyzing HTTP headers because many times these headers are not setup correctly.

The Java class that is responsible for parsing the retrieved document is the `FaqParser` (see Figure 4.6). To be able to efficiently parse text documents the FAQ parser uses a more generic text parser which is also depicted in Figure 4.6. The FAQ parser needs to have several more complex text parsing options than the standard Java API text utilities provide. The two main parser features are the option to parse the document forward or backwards and to be able to save the state of the parser at any point. These two features make it possible to implement more complex parse logic. Being able to parse forward or backwards becomes useful when parsing forward to find the end of a question, and then trying to find the beginning of the question. The parser would first parse forward and then backwards.



Figure 4.6. FAQ parser classes.

Some of the features of the text parser are described in more detail in Table 4.2. The main part of the FAQ parser that is called by the wizard Servlet-page implementation takes two attributes. These two attributes are the content of the document and an index where to

start parsing. If an index is not specified it will start parsing from the beginning of the document. If the document is determined to be HTML it will start at the starting body tag of the document to avoid any JavaScript syntax.

Table 4.2. Text Parser Features

Parse method	There are several ways to parse the document text using the parsers parse-to methods. All these methods will search the document until a certain criterion is met. The supported criteria are the following: parse to a single token, an index, the closest match of an array of a set of supplied strings, the next uppercase character, or a regular expression. Each parse-to method will return the string from the current index to the index which matches the criteria.
Parse direction	The parser can be set to parse either forward or backwards. This is useful if the end of a question is determined by a question mark and the start of the question needs to be determined. The parser can then be set to parse the document backwards from the current index.
Save-points	In some scenarios more than one algorithm or search needs to be executed before information is extracted correctly. If some algorithms fail it is usually desirable to start over from a previous state and try something else. For these types of scenarios the parser supports save-points. A save-point works in a similar way as the save-points in Oracle where a rollback will exclude queries that happened after the save-point was set. The parser has a rollback method that brings the parser back to the state of the save-point. Multiple save-points are supported.
Case-insensitive parsing	For some algorithms case-insensitive parsing is preferable. In these scenarios the parser can be set to ignore case.

The JavaScript syntax is usually written within the HTML header tag. JavaScript and other scripting languages tend to confuse the parser because of all the special characters. The parser will then extract questions one by one until all questions have been extracted from the

document. The program flow of the main part of the parser is presented in Figure 4.7. The FAQ parser has three different steps it goes through before it determines that the parse process has failed.

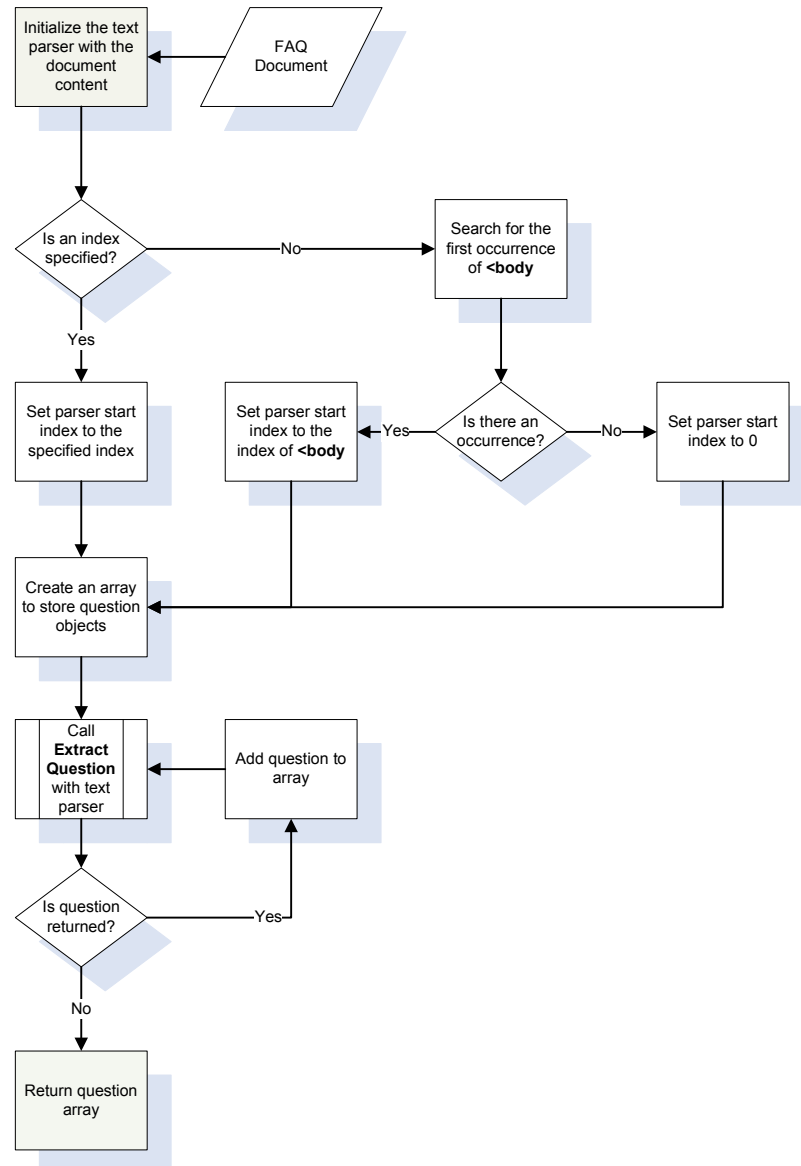


Figure 4.7. Program flow diagram for the FAQ parser.

In the first step it tries to parse the document as an HTML formatted text document. When implementing the text parser, I discovered that FAQ documents that were formatted using HTML had one common feature. This feature made them easier to parse than regular text documents. I discovered that due to the fact that the markup tags usually surround the

questions and answers in different ways to format the text. This fact can be adopted by the parse algorithm to more efficiently parse HTML FAQ documents. The question mark (?) will identify a question when written using the Latin alphabet. In the English language the question mark will be positioned at the end of a written question. When the end of a question is found in an HTML formatted document, the parser will continue parsing until it finds the outer most closing HTML tag. After this is done, if a matching starting tag can be determined, the content between the starting tag and the ending tag will be the question. This is a somewhat simplified description of the process and the complete program flow for parsing the HTML FAQ document is shown in Figure 4.8.

If the HTML parsing fails, the parser will resort to other means of extracting the questions. This is shown in Figure 4.7 as the non-HTML text parsing logic. It will use two different methods. The first method is using common question prefix strings used in FAQ documents to determine when a question starts. Questions are commonly prefixed with things like:

- Q: or Q. or Q)
- Question:
- Ask:

If this does not work the parser will try to use common words that usually indicate the start of a question. Words such as ARE, CAN, DO, DOES, HOW, IF, IS, WHAT, WHEN, WHERE, WHICH, WHO, WHY, and WOULD are some of the common words. Both these methods will require that a question mark be available prior to the word or symbol (except for the first question extracted). In this manner the parser can quite efficiently tackle both HTML and plain text FAQ documents. Most FAQ documents that are found through web searches will be formatted using HTML. With HTML formatted document the parser success rate is much higher than plain text documents.

If the parser succeeds at extracting a set of questions from the document it will store the question start index inside a question container object (see Figure 4.9). The parser will use this index to revisit the website and retrieve the answers after the administrator has made his final choice what questions to keep. The parser will also check for duplicates within the set of questions. Many web sites will put a table of contents listing all the questions on top of the document. This will cause the questions to be repeated with their associated answer

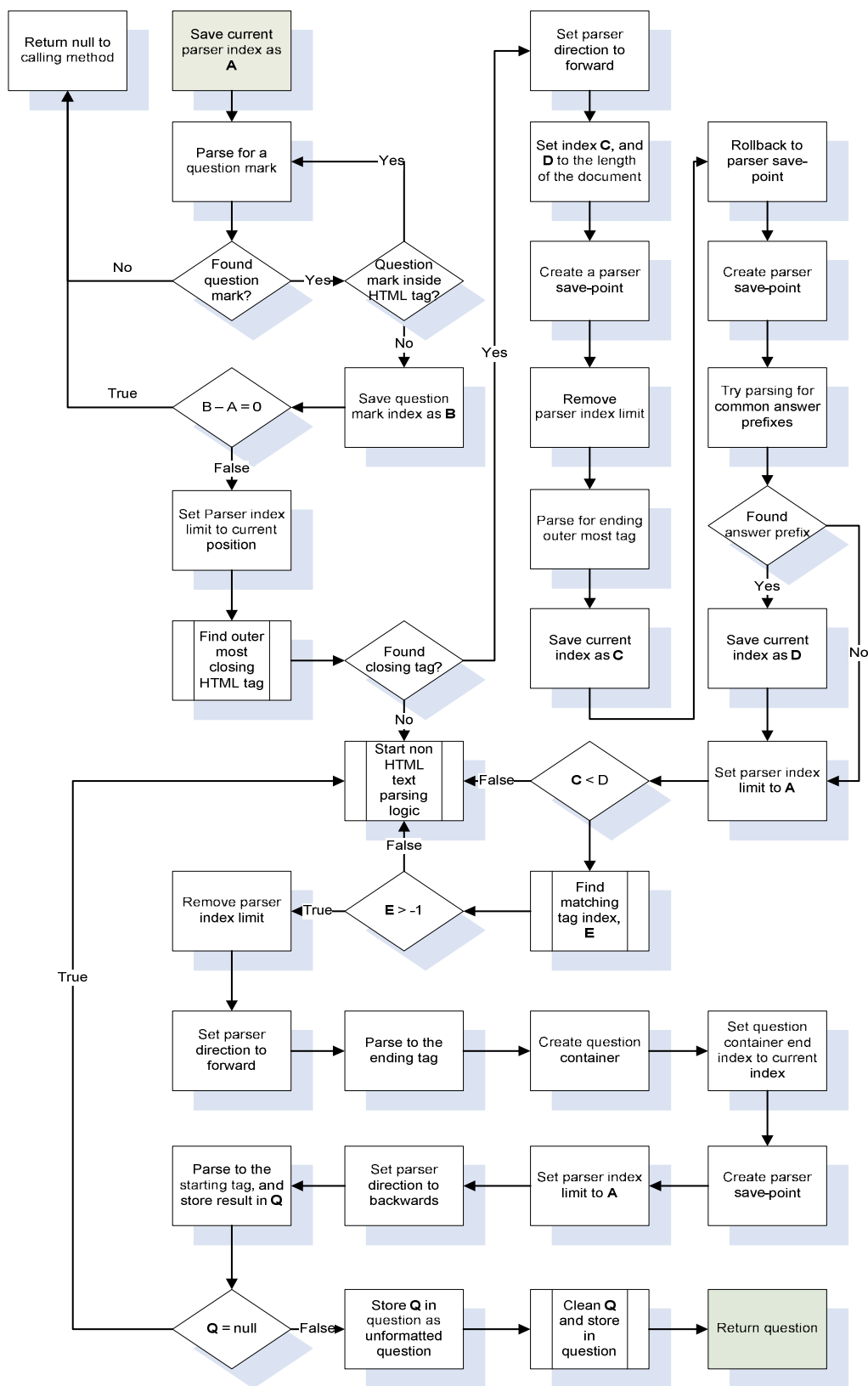


Figure 4.8. Program flow chart for the extract-question method.

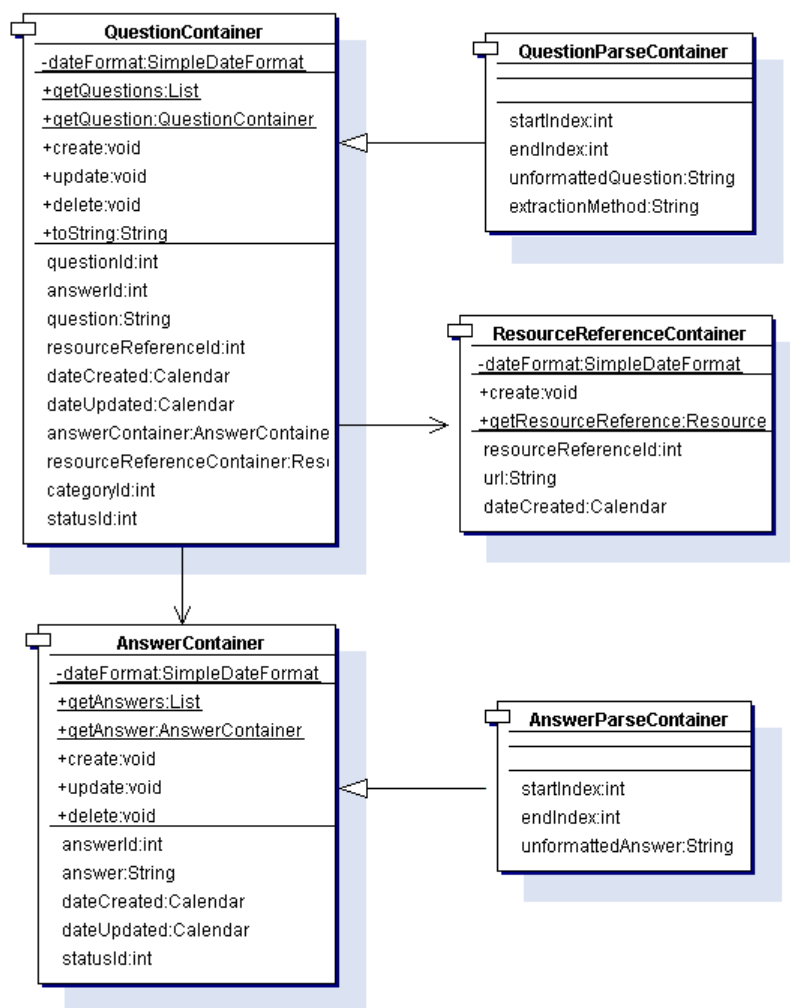


Figure 4.9. Question and answer container classes.

further down on the page. If the first duplicate to the first question can be determined, all the questions up until that point can be discarded. Then the table of contents will not affect the result.

At the last step when a set of questions and answers have been determined they are inserted into the database with the associated URL reference. The reference will indicate where the question and answer were initially located. This can be useful for administrative purposes. During the insertion phase if a question already exist the new qa-pair will be discarded. If an answer already exists but the question is different, the new question is linked to the existing answer. This way there exist a many-to-one relationship between questions

and answers. To increase the success rate of a question matching algorithm you would want to map as many differently formulated questions to the same answer as possible.

During the development of the parser I found it hard to keep it backwards compatible with documents it historically was able to parse. To solve this problem I implemented a JUnit test which automatically could verify a new parser against a set of older documents. These documents are stored with the distribution and can therefore be tested for accuracy any time the parser is extended with additional logic.

4.6 QA DATABASE SCHEMA

The database role in the QA system is to store the questions and answers representing the knowledge base. MySQL was chosen as the underlying database software (Appendix A). This relational database supports all the necessary features needed for the knowledge base.

Features such as SQL query language to insert and extract information, and constraints to maintain referential integrity. The schema consists of three tables which can be viewed in the ER diagram in Figure 4.10. The database tables hold all the information that has been extracted from the FAQ documents. The question table includes the question text which is the question without a question mark. The resource reference ID in the question table refers to the URL where the question was initially retrieved from. There URLs are

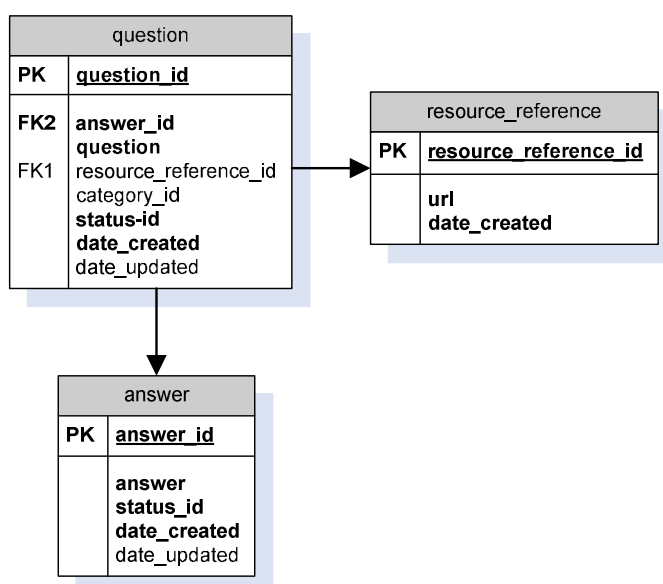


Figure 4.10. QA database ER diagram.

stored in the resource-reference table. The URL reference is not required because questions can be entered manually into the knowledge base. The question table has a category ID which is also not required. The status-ID field indicates if the question is active or deleted. That way a question is never completely removed from the database.

Questions and answers have a many-to-one relationship. The natural extension would be to support a many-to-many relationship that would be controlled by the current state of an ongoing conversation between the QA system and the client.

4.7 RESULTS USING THE SYSTEM

I used search engines to collect URLs leading to university pages with frequently asked questions. These searches resulted in a collection of a little more than one hundred URLs. After running these URLs through the wizard interface, my knowledge base was comprised of a little more than one thousand questions with answers. During this process I made minor adjustments so that the questions fit the domain. The resulting knowledge base is mainly targeted towards prospective CS majors, and not existing CS students seeking answers to CS related problems.

From initial analysis it seemed that more than 30% of the questions were comprised of How type questions. This is as expected. Students will most likely find factoid type answers faster elsewhere. The average length of the collected questions was 9.2 words. The full word distribution range can be viewed in Figure 4.11. The distribution shows that very few questions have less than five words. In the TREC competition questions were collected from various search engines such as MSNSearch and AOL (Voorhees, 2003). The average length of questions was 10.2 words (Zaanen, Pizzato, & Moll'a, 2005). The number of words in the TREC collection could mean that an ideal knowledge base will benefit from having an average question length that is close to ten words. However, the TREC competition questions length will be affected to a certain degree by the competition rules of that year.

If the number of words in a question is too small, the resulting information entropy of the question will also be low. Low information entropy will reduce the chance to match the question against existing questions in the repository. A very large amount of words in a question does not necessarily increase the performance of matching algorithms either. Words that differ will usually be penalized in the matching process in such a way that the matching

confidence score will be low. This balance is important when performing shallow language parsing without complete syntactic understanding.

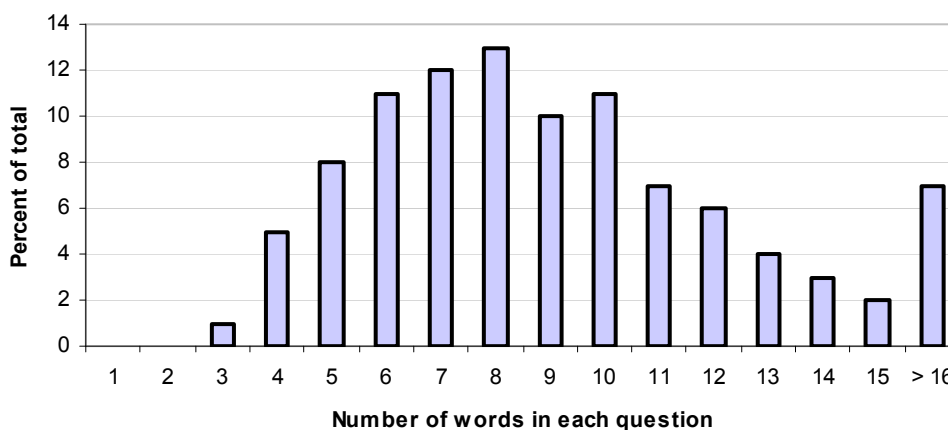


Figure 4.11. Question word count distribution.

The graph depicted in Figure 4.12 shows the initial unigram distribution of all questions in the knowledge base. The distribution shows that how type questions are very common. Factoid questions such as when or who type question are not that frequent.

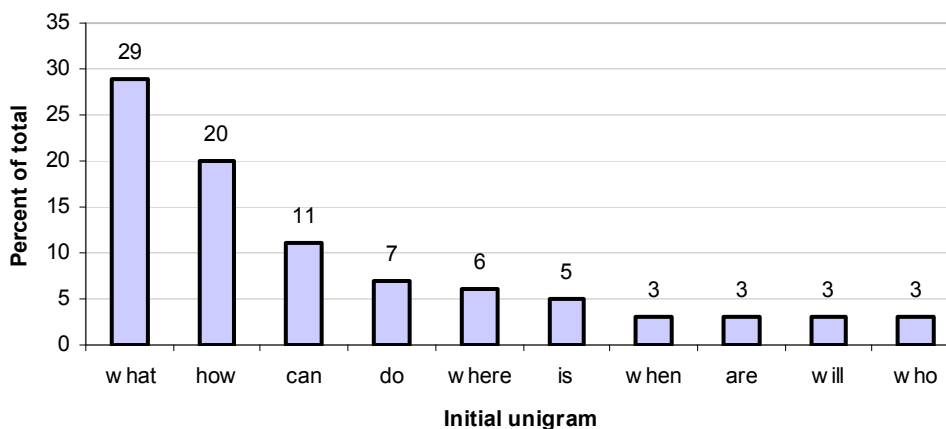


Figure 4.12. Distribution of initial unigrams.

4.7.1 Quality of QA Pairs

The quality of the collected QA pairs will heavily impact the performance of retrieval algorithms later on. I found that the quality of QA pairs varies quite significantly depending on what documents they were collected from. Some questions are composed of really long sentences that will make them difficult to match against. Some questions were just too short

to contain sufficient information to determine a match. Sneiders (1999) writes about the quality of question phrases.

In this paper he describes three features that characterize a good QA pair:

1. Thorough selection of required and optional keywords.
2. Good context controlled vocabulary.
3. Sufficient number of auxiliary entries.

Sneiders defines required keywords as keywords that have to be part of a question before even trying to match. Optional keywords can be part of the question but are not required for qualifying a question. Sneiders also states that the vocabulary should tie into the context of the question. There should also be a sufficient number of auxiliary entries. Auxiliary entries mean that multiple questions lead to the same answer. The QA collection system I developed is designed to support a large set of auxiliary entries. The number of auxiliary entries will in fact grow as more documents are parsed and the system matures. Multiple questions leading to the same answer will suppress any bad influence of questions that are badly formatted. Having multiple questions linked to the same answer will also suppress the lack of performance a restricted-domain QA system can be exposed to due to the lack of co-reference that is widely used in most open-domain QA systems (Morton, 1999).

The most noticeable drawback with Sneider's system is that the required and optional keywords are selected manually. This implies that an administrator is initially required to invest a lot of time before deploying the system. Based on empirical knowledge derived from system logs, the administrator will most likely also have to further adjust the required words.

4.8 REFINING THE QA KNOWLEDGE BASE

After examining the initial set of QA pairs in the knowledge base I found a lot of spelling errors. There were also many domain specific words that would not be part of a common dictionary. These domain specific words were many times spelled differently, e.g. MSC, and MS-CS. To keep these inconsistencies would complicate the process of matching incoming questions against questions in the knowledge base. To resolve these inconsistencies and to normalize the database I implemented the following features to be part of the QA system:

1. Spell checker.
2. Global dictionary.
3. Domain specific dictionary.
4. Global substitutions.
5. Domain specific substitutions.

4.8.1 Spell Checker

Although spell checkers are uncommon in QA systems they can greatly benefit from this feature. In a restricted domain the spell checker can be used to refine the knowledge base so that all different terms are correctly spelled. With a spell checker the system can also ensure that incoming sentence terms are correctly spelled which in turn increases the possibilities of finding correct answers. The spell checker I ended up using is an open source project named Jazzy. I wrapped the spell checker in a tool that guides an administrator through the entire set of questions. Any time a misspelled word was encountered it would be presented to the administrator with associated suggestions on how to correct it. The spell checker itself can be instantiated with several different dictionaries. For this initial task I wanted a very large dictionary that would cover a big portion if not all of the English language. There are several dictionaries and word lists freely available to the public. I used the 12dicts which holds approximately 80.000 English words. The spell check process works as follows:

1. Tokenize sentence words.
2. Remove stop words.
3. Check each word for misspellings.

4.8.2 Domain Specific Dictionary

The domain specific dictionary was also extracted by using the spell checker. All domain specific words were flagged as misspellings. After the domain specific dictionary had been created it got incorporated into the spell checker so that these words would not be flagged in future spell checks. A domain specific dictionary or ontology is a very strong feature for restricted QA systems.

4.8.3 Global and Domain Substitutions

To be able to automatically resolve domain specific substitution a substitution table was created. With this table substitutions such as MSC and MS-SC can automatically be resolved to one common term. A global substitution table (see Table 4.3) was also created. The global substitutions mostly consist of truncated words such as:

Table 4.3. Example of Global Substitutions

Term	Substitution
what's	what is
who's	who is
how's	how is
couldn't	could not

The normalization of the existing QA knowledge base is also implemented as an automated process where the administrator will be presented with suggested substitutions if a question is encountered that is not fully normalized.

4.9 EXTENDING THE QA KNOWLEDGE BASE

When trying to match questions against existing QA pairs in the knowledge base, it is important to have many auxiliary questions. These auxiliary questions are linked to the same answer but should all be formulated somewhat differently.

- Why do I need a master's degree?
- Can you give me a reason why I should pursue a master in computer science?
- Why get a graduate degree in computer science?

In the examples above we can see that not only is the MS degree formulated differently but the questions are not similar at all. This will increase the chances of a possible match. Extending the knowledge base this way can either be done manually by the people maintaining the system, or by some automated way that rephrases existing questions. It is difficult to rewrite questions efficiently without a complete comprehension of the English language. The approach I took was to write an automated system to translate questions from English to an intermittent language, and then back to English again. An example on how

sentences get rephrased can be seen in Table 4.4. The idea is that for most translation services this process will render a slightly different sentence than the original.

Table 4.4. Rephrasing Questions through Translation

Language	Question
English	How do I contact my advisor?
German	How do I approach my advisor?
Chinese Simpl.	How do I relate with mine consultant?
Dutch	How do I contact my consultant?
French	How do I contact my adviser?
Italian	How I put myself in contact with my councilman?
Japanese	Do I how communicate to my advisor?
Russian	How I will be connected my adviser?

In this manner, I can obtain a training set with the size of the knowledge base times the number of languages used. The translation services I used for this task is provided by Systrans, and Google. Most Anglo-Saxon languages provide quality translation when questions are rephrased. However, some other languages might actually degrade system performance because of badly formatted sentences. To avoid badly formatted sentences, the system needs to verify the syntactic accuracy of a sentence. This can be done in several different ways. I decided to post n-grams to Google to determine if the word combination exists. This is far from a scientific approach but it does provide a crude way to determine accuracy. To determine how well this would work I tried this approach on the existing knowledge base. Bigrams, trigrams, and quadrigrams were used for the test. The results can be viewed in Figure 4.13.

When performing this type of verification one will rely on the notion that web documents are always formatted correctly. This of course is not true. If the sentence being tested is worded slightly wrong, there is a high possibility that someone else made the same mistake. If an n-gram is worded incorrectly within any content that has been indexed by the search engine it will be flagged as valid. Any domain specific words included in an n-gram will also increase the chances of failing verification. When analyzing the results, the bigram

test qualifies the most of the questions as valid (98%). The Trigram test produces almost the same result as the bigram test, but when using quadrigrams the results dropped significantly. When looking at the quadrigrams that failed, I discovered that most of them fail due to domain specific words. The other portion of quadrigrams that failed was due to badly formatted questions in the existing knowledge base. Quadrigrams failing due to domain specific words could be solved by constructing a domain specific ontology. This would mirror WordNet where the hypernym would replace the domain specific term. This way SDSU (San Diego State University) would be replaced with college. However, 84% is still sufficient to produce a large enough amount of auxiliary questions. Another way to limit the number of badly formatted questions would be to limit the search to use trusted sources such as dictionaries and lexicons.

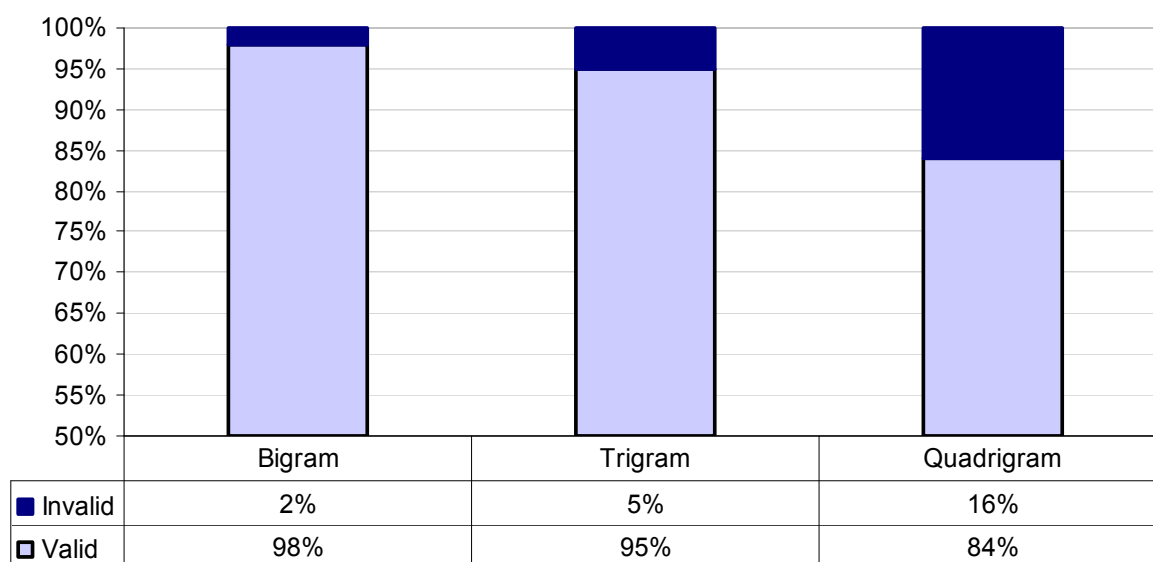


Figure 4.13. Results for knowledge base n-gram verification.

When analyzing the quadrigrams that passed the verification, most of them were correctly worded. This shows that quadrigrams will limit the search results sufficiently to avoid most of the badly formatted phrases. As the amount of indexed pages grow the n-gram might have to be increased to limit the amount of badly formatted sentences. When analyzing the results, it is apparent that the search misses has an exponential trend. This would limit the problem of badly formatted sentences when more searchable content is indexed.

After using five different languages with n-gram verification, I was able to extend the knowledge base 2.29 times its original size. The number of n-grams that failed was highly dependent on what language was being used. This can be seen in Figure 4.14.

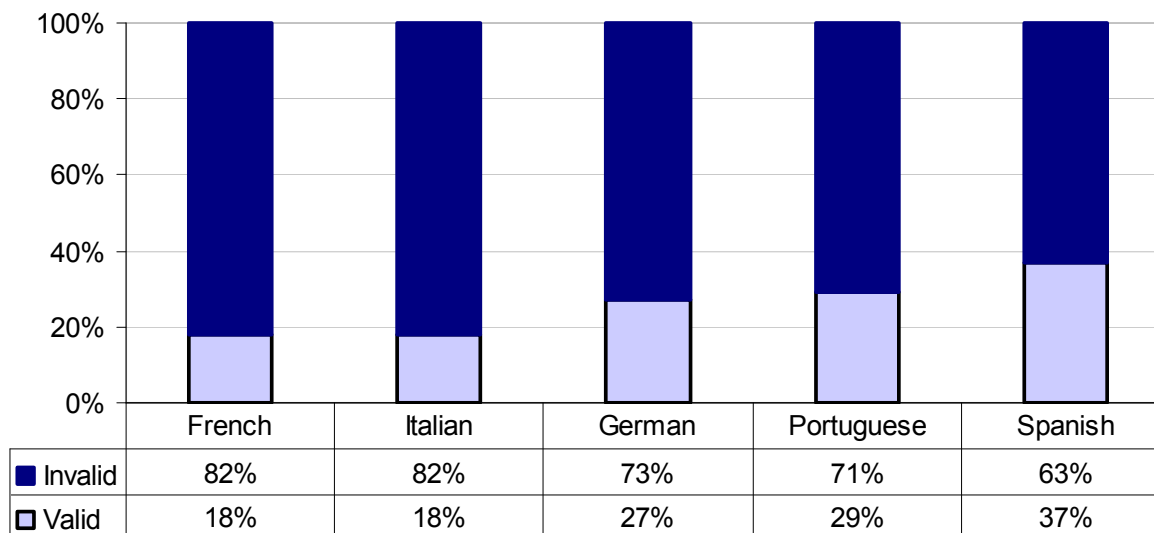


Figure 4.14. N-gram verification results across different languages.

Shorter questions tend to have a much greater chance of generating a correct rephrased sentence. Almost none of the longer questions were translated correctly.

4.10 ADMINISTRATION INTERFACE

To make it easier for an Administrator to manage the ContextQA system there is also an administrative interface available. This interface is described in more detail in Appendix B. Having a useful administration interface is very important. It is especially important to have search capabilities that closely match that of the main QA system. This way an Administrator can analyze logs and quickly perform the same type of searches to find answers that might require additional auxiliary questions.

CHAPTER 5

CONTEXTQA SYSTEM

From a high level view the ContextQA system is built up by three different parts which are depicted in Figure 5.1. These three parts are the client software, the web server, and the database. This design is comparable with most web applications. The client software and the database system are both insignificant in comparison with the amount of logic implemented in the application server.

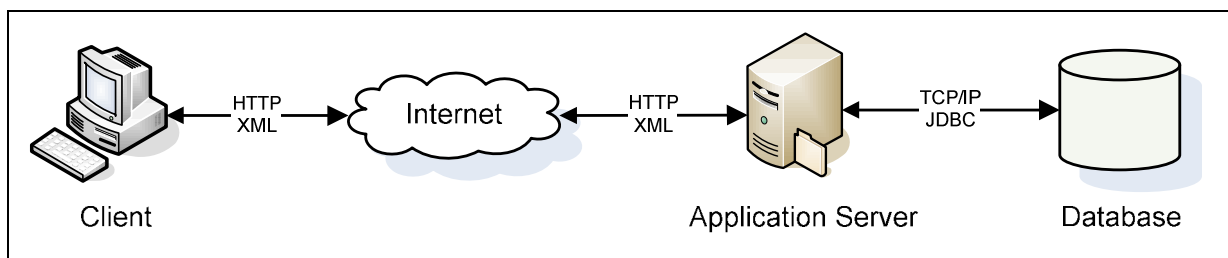


Figure 5.1. ContextQA system.

5.1 CLIENT

The QA client is implemented as a flash application. The flash client communicates with QA services running on the application server by using XML requests that are sent over HTTP. This way the client only needs to be retrieved once from the application server. The interface never needs to be refreshed as regular HTML applications. Light weight queries and responses are transmitted between the client and the services residing on the application server to make the application quick and responsive. When the client is first instantiated the user is prompted to input a name. The name is recorded so that more interactive and personal communication can be done throughout the session. The main client interface is then shown which includes an input row where questions can be entered and a main window where responses to questions are presented; see Figure 5.2. In addition to the response a set of related questions will also be presented in a window at the bottom of the screen. Related questions are determined by the questions that receive a high confidence when the QA

system matches the original question against questions in the repository. Any of these related questions can be selected and sent back to the system as a new question. This is an efficient way to make sure the QA system always presents something related to the user question.

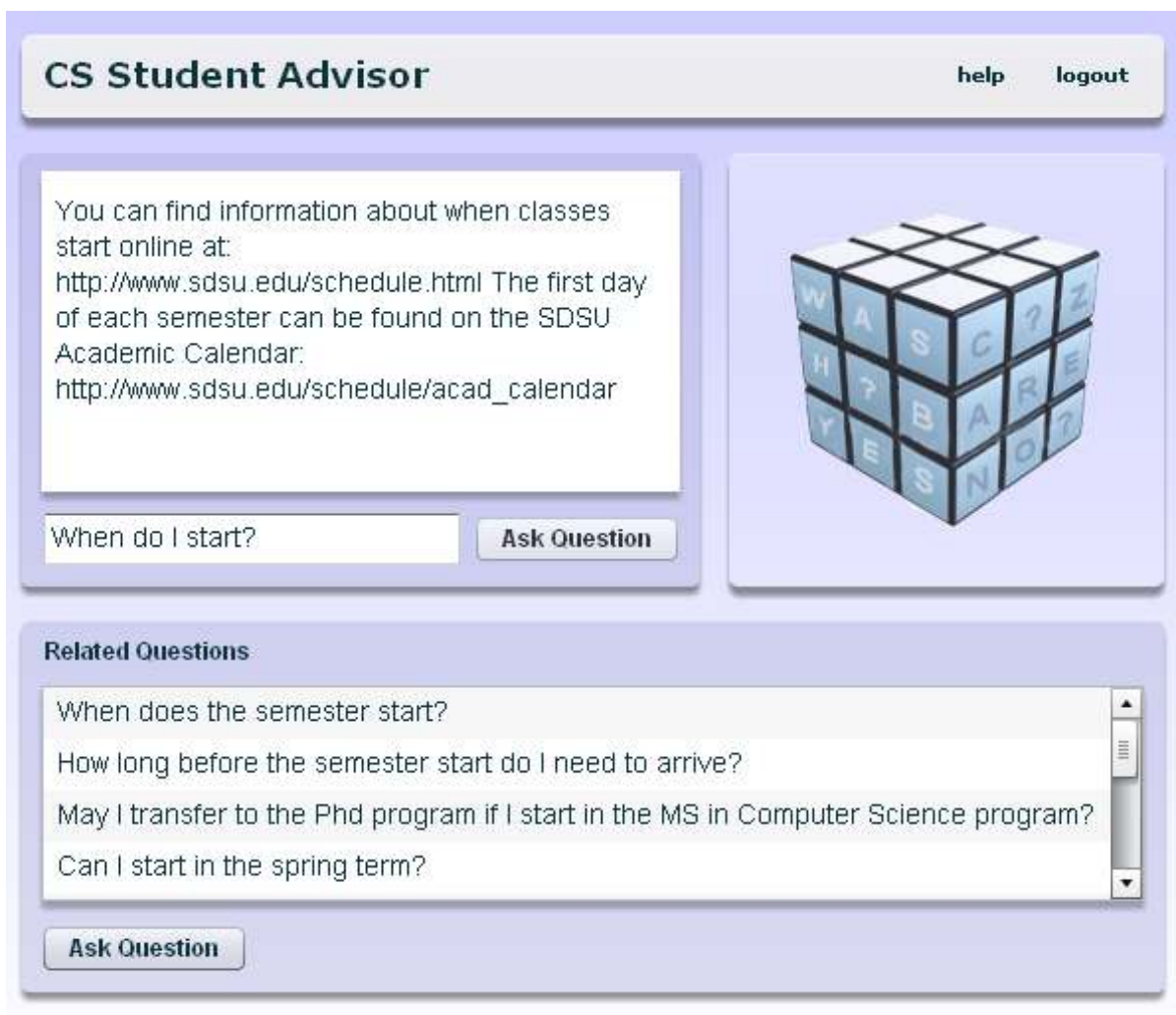


Figure 5.2. ContextQA client interface.

5.2 APPLICATION SERVER

The application server contains various services that can respond to client requests. These services share the client session. There are also several sub-systems that are instantiated at the point the application server is brought online. One of these sub-systems is A.L.I.C.E which is used as the fallback conversational system when no matching questions are found by the QA system. The main service that is called by the client continuously is the

service that handles incoming questions. This service will refine incoming questions and pass them to the QA agent implementation. The QA agent implementation then returns a set of QA pairs with associated confidence scores. Each agent also supplies its own confidence cut-off score that will indicate whether or not to present the answer has sufficient confidence to be presented to the user. The process is described in Figure 5.3.

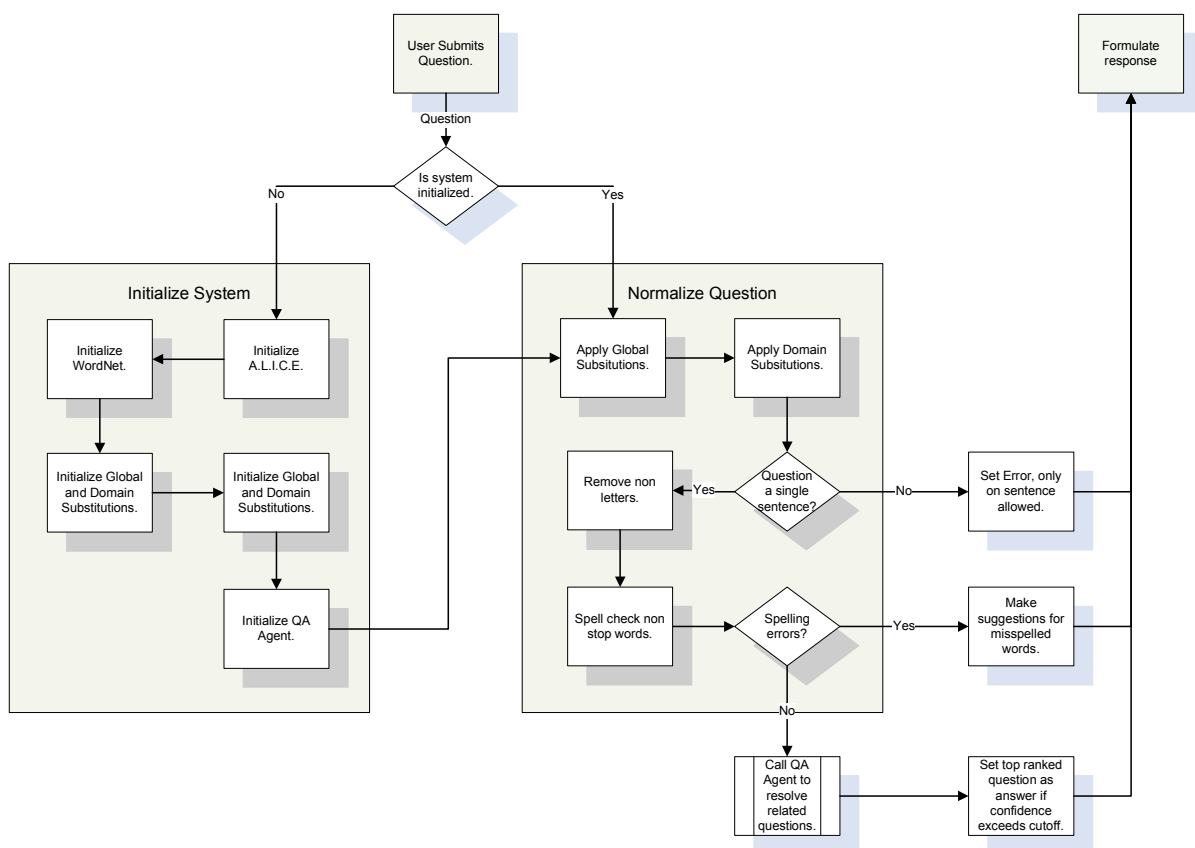


Figure 5.3. High level process flow of the ContextQA system.

5.3 DATABASE

The database holds several resources that are mainly used by the QA agent implementations. The database holds global and domain dictionaries, substitution tables, stop word lists, etc. Common task for all agents are to use refined database indexes for performing fast lookups to get candidate QA pairs. Resolving candidate questions is described in more detail in the system evaluation section.

CHAPTER 6

SYSTEM EVALUATION AND RESULTS

This chapter provides an in depth analysis using different QA algorithms for the restricted-domain QA system developed for the ContextQA system.

6.1 MEASURING RESULTS

To accurately measure the performance of a QA system you need metrics that can provide a good indication on how the system would perform in a real world scenario. In QA it is important to retrieve documents that contain the answers or part of the answer that will satisfy the user question. In the ContextQA system where the knowledge base is constructed by QA pairs, it is important to retrieve questions that match what the user writes. The precision and recall parameters are the most commonly used indicators to measure IR retrieval quality Salton and McGill (1983), and Rijsbergen (1979). Rijsbergen defines precision as *the proportion of relevant material actually retrieved in answer to a search request*. The precision parameter describes the relation between the number of relevant documents and the total number of documents returned from a user query (see Figure 6.1).

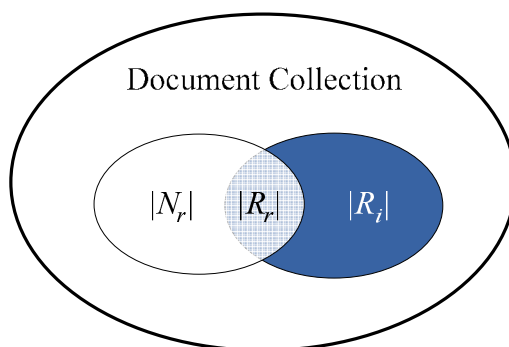


Figure 6.1. Precision and recall.

Given that $|R_r|$ are the number of relevant documents retrieved, and $|R_i|$ are the number of irrelevant documents retrieved, the precision is given by the following equation:

$$precision = \frac{|R_r|}{|R_r| + |R_i|}$$

Rijsbergen (1979) defines recall as *the proportion of retrieved material that is actually relevant*. Recall describes the relation between the number of relevant documents that were retrieved $|R_r|$, and the number of relevant document that was not retrieved $|N_r|$.

$$recall = \frac{|R_r|}{|R_r| + |N_r|}$$

The recall and precision parameters are usually inversely related. A high recall number will result in a low precision number and vice versa. A traditional IR system will try to maximize both recall and precision. In my scenario, I am trying to retrieve questions that match the user question. In this manner, the goal is slightly different. I need to maximize the recall parameter in the first IR step of the QA pipe line. This is done so that the question matching algorithm has as much relevant data to work with as possible. In my scenario, I want to have a recall number close to 100% when retrieving the first set of questions to evaluate. If the recall value is low, this will hinder any type of matching algorithm, especially those relying on auxiliary questions leading to the same answer. The bad results will then propagate through the entire QA pipeline. An easy way to improve the recall rate would be to retrieve as many QA pairs as possible. Some restricted-domain QA systems evaluate all documents in the knowledge base. Retrieving all documents is not a good idea because it hinders the system to scale efficiently when adding more complexity. An example would be a system intensive matching algorithm.

One algorithm used in the TREC QA competition is the Mean Reciprocal Ranking (MRR) algorithm. The MRR algorithm considers the rank of the first correct answer in a list of possible answers. If a system returns the correct answer first in the list it retrieves a 100% score. If the correct answer is in the fifth slot it retrieves a 20% score. *The reciprocal rank has several advantages as a scoring metric. It is closely related to the average precision*

measure used extensively in document retrieval. It is bounded between 0 and 1, inclusive, and averages well (Voorhees, 1999).

Another important aspect when measuring QA systems performance is the ability to determine if an answer does not exist. It is always more important to be able to determine the non existence of an answer than to provide an answer that is faulty.

There have also been several additional methods developed on how to measure the performance of QA systems (Breck et al., 2001; Radev, Qi, Wu, & Fan, 2002). These methods are more geared toward the specifics of a QA system than those that are used to measure standard IR performance. Things like a systems response should also play a role in determining the overall system performance. In a QA system the response time should be instant but many times it is not.

6.1.1 The Importance of a Good Test Collection

When developing a document classification system, the test collection can usually be obtained by extracting a portion of the already classified documents. Given the QA knowledge base a certain amount of the auxiliary questions that are linked to the unique answers could be extracted to build such a test collection. However, this would provide a bias towards the content of the knowledge base. Even a trivial restricted-domain requires a significant number of questions to not become completely useless. This is why it is important to test the system using domain related questions obtained elsewhere. An unbiased test collection would be a set of NL questions that relates to the restricted-domain but were formulated completely independently of any knowledge of the QA system. To obtain such a set of questions I ran the same question collection process that I used when building the initial knowledge base excluding the last step where the questions and answers are incorporated in the knowledge base. This way I was able to collect a complete set of unrefined questions that you would expect actual clients would write. Each of these questions has one of two distinct features. Either an answer exists, or does not exist in the knowledge base. This way metrics can be provided on how good the QA system is able to determine if an answer exists or not. This is one of the metrics to measure QA performance included in the TREC competition (Voorhees, 1999). Excerpts from the test collection is provided in Appendix C.

6.1.2 Automated Test Framework

To be able to quickly determine the quality of an algorithm the ContextQA system was equipped with an automated test framework where a new algorithm can be run against the test collection and a complete set of metrics are produced.

6.2 RESOLVING QUESTION CANDIDATES

When examining the generalized QA system in Figure 2.4, it shows a streamlined process where a QA system applies certain logic in each step which will either expand or limit the data set that is supplied for the next step. After the question has been analyzed the query to retrieve relevant QA pairs is constructed. If any relevant question or answer remains unresolved after this step, the error will propagate through the entire system reducing its performance. This is why it is important to be able to retrieve as much information as possible at this point in the process. The information extracted at this stage will not be visible to the user in any way so the precision is not as important as the recall. To maximize recall the ContextQA system utilizes an answer index where a large amount of question terms are linked to existing answers. This index is constructed by a separate process which iterates through each answer in the knowledge base. The following steps are executed for each answer:

1. Resolve all questions leading to the answer.
2. Remove stop words.
3. Resolve all term synonyms using WordNet.
4. Apply Porter stemmer (Porter, 1980) to each term.
5. Insert resulting terms in index table.

Given the WordNet synonym term expansion this index becomes very comprehensive. Just using unigrams to resolve synonyms can limit the number of synonyms when using WordNet because WordNet include n-grams. To accommodate for this, step three also includes bigrams and trigrams extracted from the original question to resolve synonyms. In step three, the average number of synonyms resolved per question was 27.65. Any resulting synonym is broken down into unigrams, and stemmed in step four.

Because polysemous words can be part of multiple synsets I extract synonyms from each synset for all senses. I do this because the system currently does not have any way of

determining the word sense (word sense disambiguation) for the word within a sentence. This approach could cause problems or decrease performance if the sense is in fact different between the questions being compared. However, in restricted domain question answering this problem will be limited because the restricted domain will increase the chances that the word sense is the same. An example could be the word interest. This word could mean interest in taking some course work, or the interest on your bank account. The bank interest word sense is unlikely to be part of a restricted domain that deals with giving advice to prospective students. Adding additional shallow language parsing logic to the QA system will further limit the impact of this potential problem.

The sole purpose of this index is to retrieve as many potential question candidates as possible while still limiting the result to only a subset of all questions. A question answering system which would rely only on this mechanism to produce a final set of answers would produce poor results from excluding stop words. The value of stemming versus morphological query expansion was analyzed in detail by Bilotti, Katz, and Lin (2004). The conclusion was the morphological expansion provided better results. The approach taken here uses both expansion and stemming. This will increase the recall even further which is the main purpose.

There can also be a drawback when using an index based on stemmed words. Using stemming in the process of scoring question candidates can hurt performance. Stemming words can change the meaning of a sentence. An example could be the sentence, Name the fastest runners. Using the Porter stemmer runners is reduced to runner and the question that should produce a list of the fastest runners is reduced to just providing the fastest runner. The same goes with stop words. When removing stop words it can also hurt performance. If stop words such as Who or When are removed it can limit the meaning of the question. It is important to understand when to expand a question phrase and when to reduce it.

6.3 QUESTION SELECTION AGENTS

The ContextQA system is implemented in a modular way which allows it to easily replace the core set of algorithms that match incoming questions against questions in the knowledge base. These algorithms are controlled by an agent. The agent receives a plain text question and responds with a list of QA pairs ordered by relevance. Each qa-pair has a

confidence score. This confidence score is compared against a cut-off score to determine if an answer exists. The cut-off score for the confidence number can be different for different agents. If the top rated QA pair falls below the cut-off level this is an indication that the agent was unable to find an answer. This is called a NIL response. If the confidence score ranks above the cut-off confidence score the answer portion of the QA pair will be presented to the user.

6.3.1 Agent Resources

Each QA agent has access to various base resources that exist within the knowledge base of the ContextQA system. If any other resources are needed the agent can set those up in the initialization phase of the system. This section lists the basic resources available to all agents.

6.3.1.1 QUESTION INDEX

To limit the number of questions that will be evaluated during the matching phase and to maximize question recall a lookup table was created. This table maps question words to answer ID's. This table is generated when the QA knowledge base has been normalized. Before the process is executed the knowledge base is verified to be completely normalized. For every question any word or word combination (up to tri-grams) is included if it exists in the domain dictionary. The following logical sequence builds the lookup table.

Iterate across each word

- Remove stop words

- Extract all possible synonyms from WordNet

Iterate across each bigram

- Extract all possible synonyms from WordNet

Iterate across each trigram

- Extract all possible synonyms from WordNet

Create a unique set of synonyms and insert in the synonym answer mapping table.

The maximum number of word combinations that could qualify in a sentence of n words is expressed below:

$$F(0) = 0$$

$$F(1) = 1$$

$$F(2) = 3$$

$$F(n) = 3n - 3 \text{ (for } n > 2)$$

However, this situation will be highly unlikely when using free-form questions which will almost always include several stop words the longer the sentence is. I evaluated the approach to use a stemmer to index the questions but determined that this led to worse results if not retrieving a very large collection of potential candidates. Using the raw words and adding morphological variations extracted through synsets in WordNet was more efficient. This follows in line with what Bilotti and Katz (2004) found.

6.3.1.2 DOMAIN DICTIONARY

In a restricted-domain QA system it is very important to have a domain specific dictionary. In an open-domain QA system you can usually rely on co-occurrence to determine domain specific words. In a restricted-domain system you will not have that luxury and especially not when the knowledge base is generating from QA pairs. Each agent has access to a domain specific dictionary that can be used for spelling correction or other domain specific logic. This domain dictionary was generated by the spelling correction process that is part of the ContextQA system.

6.3.2 Agent Results

This section presents the results from different agents that all implement different QA algorithms. Each agent follows the same model where it receives an NL question and returns a set of potential matches. The last agents also provide a confidence level which indicates if it was able to find an answer at all. For each agent the same set of performance metrics are presented.

6.3.2.1 AGENT HOMER

The Homer agent tokenizes the question entered by the user into a bag of words. The resulting bag of words is then used as a query constraint against the existing repository of questions in the database. The resulting list of questions is then sorted based on relevance. The relevance is determined from the number of words that match words in the original question. Having the query be unconstrained result in a significant amount of questions to match against for each new question that the agent process.

6.3.2.2 AGENT HOMER PERFORMANCE METRICS

Table 6.1 lists the total number of questions, the number of questions where an answer exists, and the number of questions that does not have an answer in the repository.

Table 6.1. Test Collection

Total test questions	44
Total test questions where answer exists.	34
Total test questions where no answer exist (NIL)	10

The following metrics (see Table 6.2) are based on the first answer the agent returns. A NIL-answer means that the agent does not think there is an answer available. The Homer agent does not have the capability to determine if a question results in a NIL-answer and only returns the highest scoring question each time.

Table 6.2. Agent Homer Performance Metrics

Correct answers	13
Incorrect answers	31
Correct NIL answers	0
Incorrect NIL answers	0
Total correct answers	13
Total incorrect answers	31
Total correct answers / total questions	0.30
Average answer response time	290 ms

Figure 6.2 shows that this agent has thirteen correct answers at index one. This is the most important index because that will be the answer that is produced. The correct answers should have as low index as possible

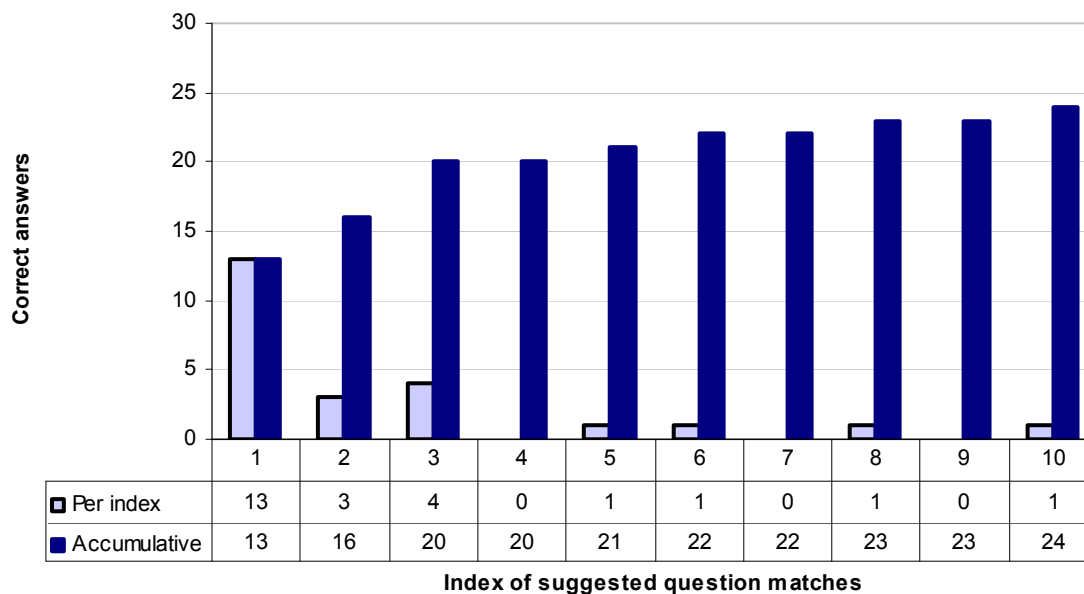


Figure 6.2. Agent Homer correct answers per index.

The MRR score listed in Table 6.3 is good compared to most open-domain qa systems.

Table 6.3. Agent Homer Aggregate Performance Metrics

Mean Reciprocal Ranking (MRR)	0.60
Total run time	12.74 seconds

This agent benefits from the fact that database queries are executed on the entire knowledge base. This would not be a valid approach in a real-world environment unless the system has access to significant resources. The results from this agent show that even with limited complexity a restricted domain system can produce results comparable to some of the better systems that attend the TREC competition.

6.3.2.3 AGENT FRY

The Fry QA agent is similar to the Homer agent except it is utilizing a Porter stemmer on each non-stop-word to build the database query. The query is then executed on a WordNet

synonym answer mapping table that has been generated previously. The question entered by the user goes through several steps of refinement. The question is first tokenized into a list of words. The list of words is then normalized based on global substitutions and domain specific substitutions. Stop words are removed from the list. Each word is stripped of suffix and prefix using the Porter stemmer. The resulting terms are then used as a query constraint against the WordNet synonym answer mapping table in the database. This query is limited to a maximum of 50 rows. This limitation is introduced to make sure the system is realistic in a real world scenario where queries could not be executed against the entire repository of questions each time. The Fry agent will rely on this initial IR phase being as good as possible. If the sub-set of questions retrieved during this phase does not contain the correct answer it does not matter how good the analysis is after that point. That means that any question that is not part of the result will not be considered in the subsequent logic. The previous agent Homer executed queries against the entire repository which resulted in far larger data-sets to consider for possible matches. The Fry agent process the resulting questions based on maximum term matching frequency. The results show that even though this agent only considers a sub-set of the questions it scores a higher MRR. This shows the benefit of the synonym answer mapping table. Due to the limited number of questions to compare against the total runtime is also greatly improved compared to the Homer agent. All the agents except the Homer agent utilizes the synonym answer mapping table for their initial IR phase.

6.3.2.4 AGENT FRY PERFORMANCE METRICS

Table 6.4 lists the total number of questions, the number of questions where an answer exists, and the number of questions that does not have an answer in the repository.

The following metrics (see Table 6.5) are based on the first answer the agent returns. A NIL-answer means that the agent does not think there is an answer available. The Fry agent does not have the capability to determine if a result should be labeled as a NIL-answer.

Figure 6.3 (p.79) shows that this agent has eighteen correct answers at index one. This is the most important index because that will be the answer that is produced. The correct answers should have as low index as possible.

Table 6.4. Test Collection

Total test questions	44
Total test questions where answer exists.	34
Total test questions where no answer exist (NIL)	10

Table 6.5. Agent Fry Performance Metrics

Correct answers	18
Incorrect answers	26
Correct NIL answers	0
Incorrect NIL answers	0
Total correct answers	18
Total incorrect answers	26
Total correct answers / total questions	0.41
Average answer response time	150 ms

The MRR score in Table 6.6 show that the agent scores well compared to most open-domain qa systems.

With this agent there has been a significant increase in the number of correct answers compared to the previous one. We can also see that the correct answers sharply drop off after the first suggested answer which is the ideal behavior.

Table 6.6. Agent Fry Aggregate Performance Metrics

Mean Reciprocal Ranking (MRR)	0.71
Total run time	6.59 seconds

6.3.2.5 AGENT BENDER

The Bender QA agent is similar to the Fry QA agent except it extends the portion that qualifies the answer by utilizing Levenshtein word edit distance. A confidence limit is also set to determine if a NIL answer should be returned. As described earlier being able to

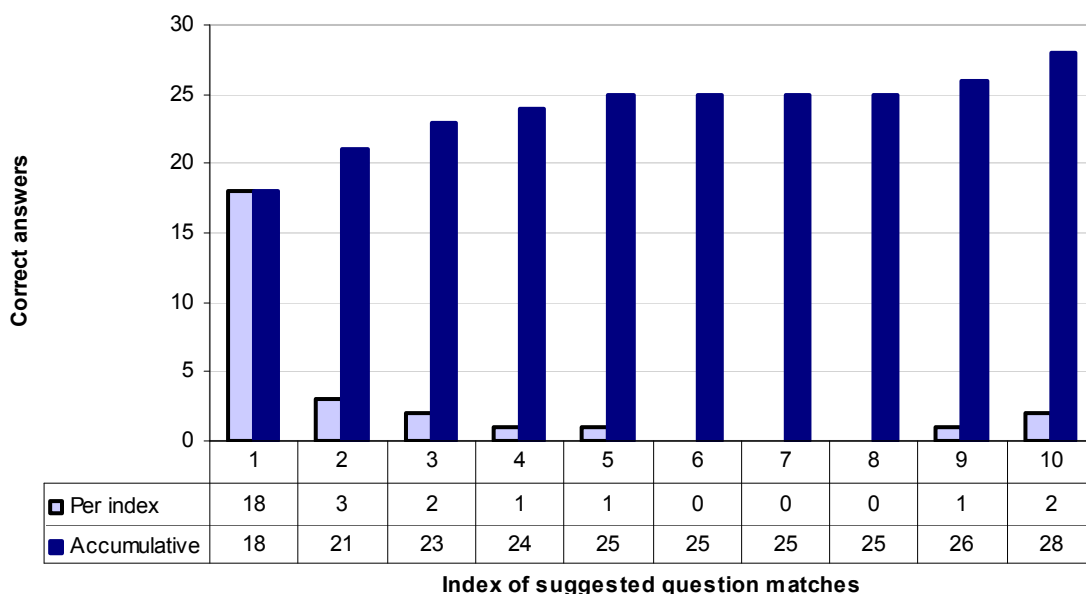


Figure 6.3. Agent Fry correct answers per index.

determine if an answer is not available is very important for a QA system. The system should be able to let the user know that it does not know the answer to the question. If the system replies with a faulty answer it can damage the user's confidence in the system or even worse the user can damage something else because of the answer. The minimum edit distance is measured on the questions after they have gone through a process where they are modified for maximum synonym match. WordNet is used to retrieve a list of synonyms for all non stop-words in each question. This list is analyzed and both questions are adjusted to match as closely as possible based on synonyms or terms that intersect both questions. The confidence score is then determined by taking the total number of words minus the edit distance, and then divided by the total number of words. The result is a number between 0, and 1.

What I found with this agent is that if stop words are not included in the edit distance calculation the performance is reduced significantly. This is because stop words can have a great importance especially in a QA system. If words like Where, and Who are removed the meaning of the question is likely to be lost.

6.3.2.6 AGENT BENDER PERFORMANCE METRICS

Table 6.7 lists the total number of questions, the number of questions where an answer exists, and the number of questions that does not have an answer in the repository.

Table 6.7. Test Collection

Total test questions	44
Total test questions where answer exists.	34
Total test questions where no answer exist (NIL)	10

The following metrics (see Table 6.8) are based on the first answer the agent returns. With a NIL-answer means that the agent does not think there is an answer available.

Table 6.8. Agent Bender Performance Metrics

Correct answers	11
Incorrect answers	7
Correct NIL answers	9
Incorrect NIL answers	17
Total correct answers	20
Total incorrect answers	24
Total correct answers / total questions	0.45
Average answer response time	3.01 seconds

Figure 6.4 (p. 81) shows that this agent has seventeen correct answers at index one. This is the most important index because that will be the answer that is produced. The correct answers should have as low index as possible. The MRR score in Table 6.9 shows a slight increase since the previous agent.

Table 6.9. Agent Bender Aggregate Performance Metrics

Mean Reciprocal Ranking (MRR)	0.72
Total run time	132.37 seconds

This agent provides a slight step forward in performance compared to the previous agent. Given the introduction of a confidence level the agent removes some of the valid answers but compensates the total correct answers with its capability to determine if an

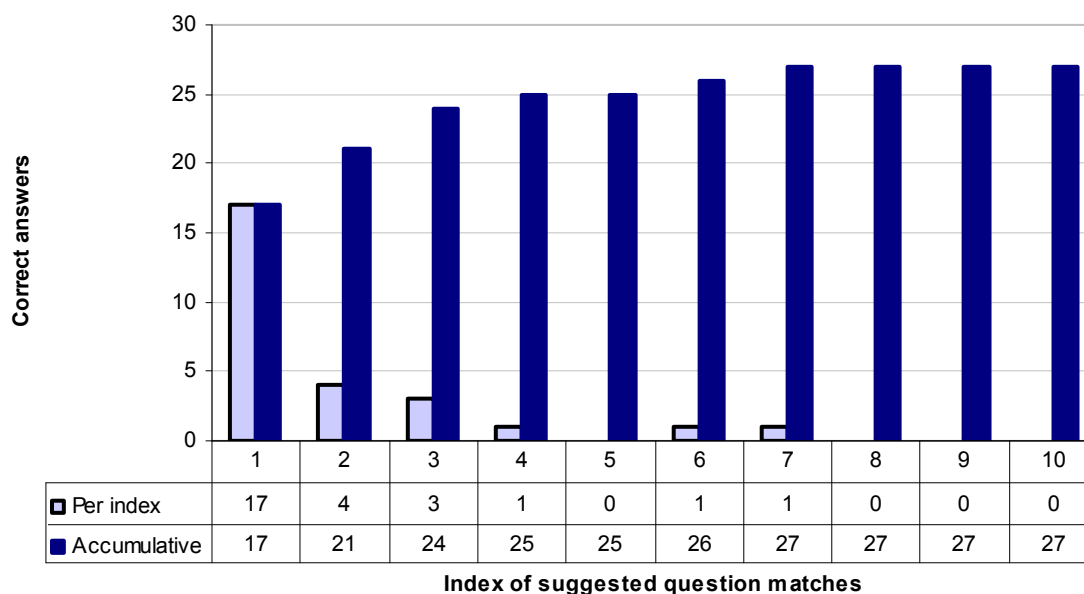


Figure 6.4. Agent Bender correct answers per index.

answer is available or not. The MRR score has also been slightly improved because of a higher density towards index one in the answers per index table. The limited test collection of 44 questions will limit the noticeable impact of modifying the algorithm. The impact of a small change can easily be misrepresented. Figure 6.5 show a graph which depicts this QA-agent running with different confidence cut-off levels against the test questions. The chart shows many interesting aspects of question answering. The following are the different labels associated with the query results:

- Correct - A correctly answered question.
- Incorrect - An incorrectly answered question.
- Correct NIL - A correct answer that states that an answer does not exist in the knowledge base.
- Incorrect NIL - An incorrect answer that states that an answer does not exist in the knowledge base.

The X-axis of the graph represents the confidence cut-off value where the agent considers the question to match the question the user inputs. For a very small confidence cut-off such as 1 the number of questions that match is relatively high ~40%. However, the number of questions that are considered to match but in fact are not correct is close to 60%. This is not

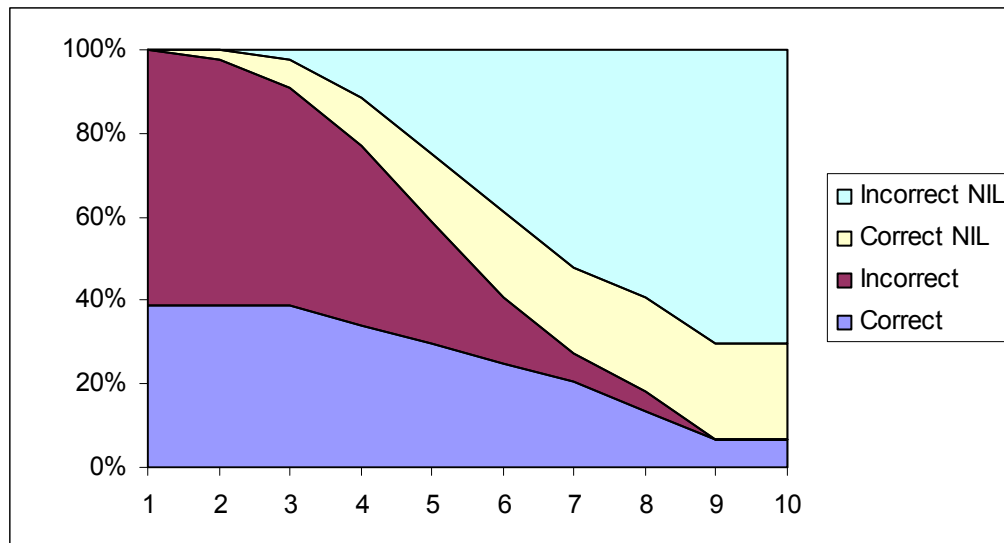


Figure 6.5. Confidence levels versus query results.

good performance for a QA-system because 6 out of 10 questions will result in the wrong answer. By increasing the confidence cut-off, more questions will start to qualify as NIL. This means that the qa-agent does not think there is a matching question and will respond with nothing. In the case of the ContextQA system this means that the system will fall back to the A.L.I.C.E conversational system. The system will respond with something that attempts to continue the conversation with the client. The ContextQA system also show the questions that the agent scored with a high confidence score as a list under the main interface.

Another interesting detail is the so called "sweet spot" for the confidence cut-off. This cut-off will be different for the type of system that is developed. For example a medical system you would never accept an incorrect response. In the above case that cut-off would correspond to a 9 or 10 setting. With this setting you would render the system quite unusable given all the NIL responses but no patients would be hurt in the process. The above agent is set to operate on a level 5 confidence. This level corresponds to a combined NIL and regular correct answer rate of 45%. If you include the faulty NIL responses which are adding another 23% of A.L.I.C.E conversations leaves us with only 20% of faulty responses. These results would probably not beat a real student advisor but the results are still good.

Another thing that would improve the above chart for the ContextQA system is the initial search algorithm that is used to determine the candidate set of questions that are used

for matching. The recall rate of this algorithm is not considered in the above chart. Only the total correct answers that exist in the knowledge base are considered. This means that the quality could in fact be improved if more questions were considered. However, the responsiveness of the system would degrade linearly with the amount of new questions added. I did not want to consider all questions, because I wanted to mirror more of a real-world scenario where you do not have an infinite amount of resources.

6.3.2.7 AGENT YODA

Many of today QA systems have several modules that extract certain features from the question and the answer candidates. Given the usually non-linearly separable solutions I determined to utilize a set of feature extracting algorithms that would each provide either confidence related information or a set of numerical features. These results are then fed into a Neural Network which in turn provides a confidence number which determines the final rankings of the questions. Limiting the number of hierarchical levels in a set of machine learning algorithms is usually a good idea if the training data is not very large. Otherwise the lower level algorithms will not receive sufficient data to be able to provide correct classifications. The complexity of measuring the effects of combining different question matching algorithms grows exponentially harder with the number of techniques used. The complexity eliminates the possibility of trying to manually determine the value of combining different techniques. The last agent implementation demonstrates that slightly altering some values can have a significant impact on the results. Most of today's QA systems, both open-domain, and restricted domain are becoming more complex by adding more parts. It is difficult to determine the impact of a new algorithm when used in combination with other algorithms. This is because the result of a certain algorithm is many times fed as input to the next algorithm. Many times when introducing a new algorithm, it affects other details of the system based on where they are positioned in the QA system's pipeline. A widely accepted approach when creating a function from a series of observations is using an artificial neural network (see Figure 6.6).

Learning a non-linear multi dimensional function can be taught through supervised learning. This is a useful feature given all descriptive metrics can be extracted from all previous agents. A neural network also fits the problem because it can model non

deterministic and stochastic problems. The input to the neural network within the Yoda agent is the same confidence number that drives the decision of the Bender agent. In addition the Yoda agent has two more inputs. The full word count difference and the word count difference with stop words removed between the two questions that are compared.

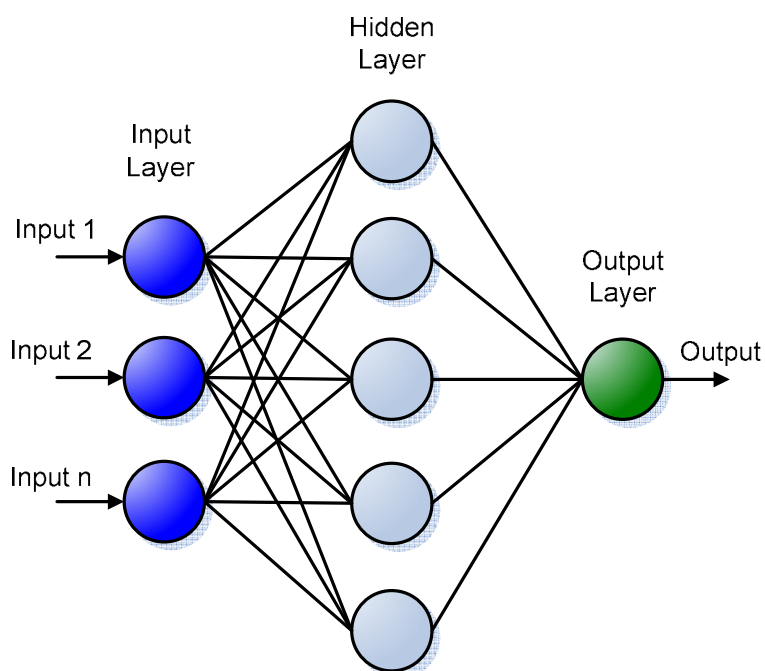


Figure 6.6. Neural network.

6.3.2.8 AGENT YODA PERFORMANCE METRICS

Table 6.10 lists the total number of questions, the number of questions where an answer exists, and the number of questions that do not have an answer in the repository.

The following metrics (see Table 6.11) are based on the first answer the agent returns. With a NIL-answer means that the agent does not think there is an answer available. The confidence level for the Yoda agent is the direct output from the neural network. The confidence level for the Yoda agent had to be set really high to get any type of performance.

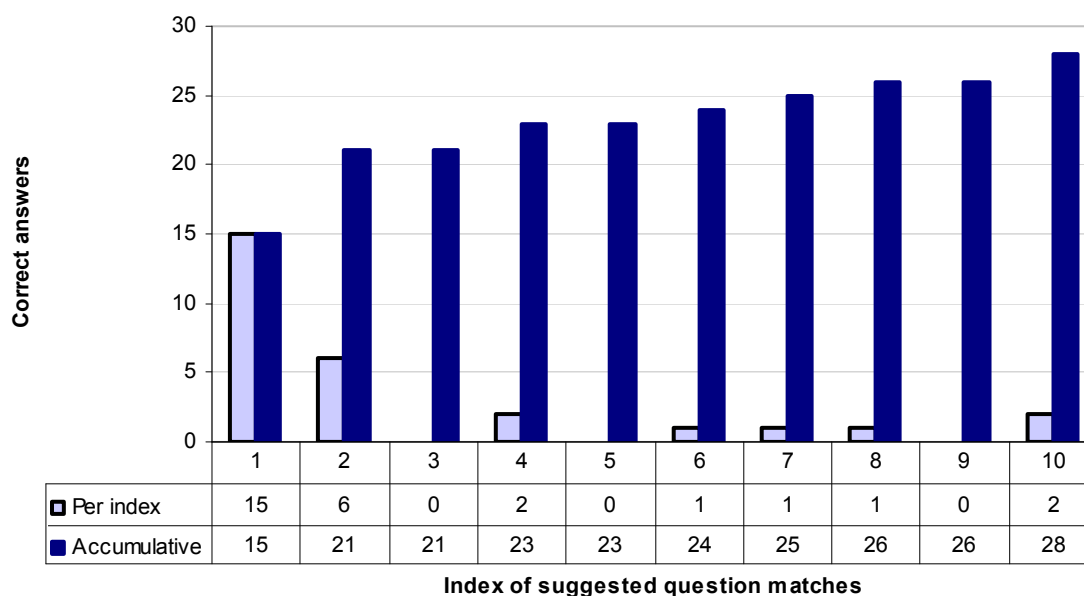
Table 6.10. Test Collection

Total test questions	44
Total test questions where answer exists.	34
Total test questions where no answer exist (NIL)	10

Table 6.11. Agent Yoda Performance Metrics

Correct answers	12
Incorrect answers	15
Correct NIL answers	6
Incorrect NIL answers	11
Total correct answers	18
Total incorrect answers	26
Total correct answers / total questions	0.41
Average answer response time	3.94 seconds

Figure 6.7 shows that this agent has fifteen correct answers at index one. This is the most important index because that will be the answer that is produced. The correct answers should have as low index as possible. The MRR shows in Table 6.12 (p. 86) shows a slight degradation since the previous agent.

**Figure 6.7. Agent Yoda correct answers per index.**

The results show that there was degradation in performance compared to the previous agent. This shows that adding the additional two indicators used as input to the network did

not provide additional data to segment the questions correctly. I still believe that using a neural network will be beneficial when adding more ways to classify questions.

Table 6.12. Agent Yoda Aggregate Performance Metrics

Mean Reciprocal Ranking (MRR)	0.68
Total run time	173.51 seconds

6.4 CONCLUSION

The restricted domain system constructed as part of this thesis fits best in a static environment where the answers do not change frequently. This way the system can slowly evolve to be more efficient on providing correct answers based on the knowledge base. The system would need certain enhancements to better fit in a more dynamic environment where the knowledge source changes each day. Such as a quickly expanding source of data or data sources from news feeds that are constantly updated. In these scenarios you would need to be able to tie answers to external resources that are not statically contained within the system itself. An example of this could be results from a test which is stored in a database. If an answer node had logic to retrieve data from this database it would be able to construct a query based on the question that led to the answer and extract information from that database.

With the ContextQA system I have shown an efficient way on how to bring open domain QA resources to use within a restricted-domain QA system. I have also shown throughout the document that you almost certainly will need to develop targeted and trusted systems for QA to ensure quality and reliability. The ContextQA system performs equally as well as some of the top performing systems in the TREC competition.

CHAPTER 7

FUTURE WORK

While writing this thesis I have evaluated several possible approaches and topics about question answering systems. In this chapter I present some ideas and topics that were not researched in detail but could serve as a natural continuation of the work I have started.

To simplify the ContextQA system even further it should become completely automated. This should be a relatively easy task given that the system is already designed towards automation. With some minor changes some of the existing manual steps could be eliminated. It should be possible to perform web searches within the system. The search should be expanded to target FAQ pages that include the terms within the search. When qualifying newly found questions the system should show existing repository questions that correlate with the new question. By showing similar questions an administrator could automatically link new questions to existing answers. When new questions are found they should automatically be normalized to fit that of the repository's domain and common dictionaries. When an administrator has collected or entered new questions and answers the indexes and resources used by QA Agents should automatically be regenerated. Spell checks should be done automatically on new questions. Domain specific words and terms should automatically be detected during the collection phase. Extracting these words and terms would be done by spell checking and analyzing the existing domain dictionary. Ideally the domain dictionary would be extended to cover relationships and include complete restricted domain ontology information. The same format as WordNet could be used to expose such a feature.

The system should be extended to support question templates. With question templates it is easier to answer factoid type questions. An example of a factoid question template could be: *What is a **. These templates could be extended to support more complex wildcards that tie into several question types. Another way to utilize question templates would be for query refinement. If the question: *What is a golden retriever?* Does not result in an answer it could be rewritten as *What is a dog?* This translation could be done by utilizing

WordNet's hypernym relationships. Other templates could be directly tied to database queries. Example *What are the prerequisites to <class name>?* Class name could qualify for a set of regular expression matching the different ways to type a class name. These templates could be tied to external resources such as databases containing class schedules etc. This way the QA system would automatically adjust when these resources are updated.

Another way to make use of question templates is to use them for query expansion. The ContextQA system could qualify a question to match a query template and derive a list of alternate way to formulate the same question. Table 7.1 gives an example how the following question: *How do I apply for graduation* can be rewritten. The question qualifier would in this case be the regular expression *^How do I*. The text following that prefix would be considered the question body.

Utilizing query expansion in this way would benefit the ContextQA system because of the way it stores free form questions in its repository. In question answering systems there exists a delicate balance weather to expand query terms or refine them.

Expose further query refinement by tying into A.L.I.C.E context terms. This way it could be translated into what thing that has been associated with it earlier in the discussion. By rewriting question based on the context of the discussion will increase the illusion of the system being intelligent.

The system should include better logging mechanisms to be able to perform empirical studies when running in live environments.

Given that the system already returns a collection of question candidates sorted on relevance the system could suggest question matches if the confidence level is too low. This could be done by rewriting the question that scores the highest. Example the following question: *Describe to me how I signup for courses?* Might not result in a match that scores high enough. The highest scoring question could be: *How do I register for classes?* The system could then rewrite that question as a suggestion. *Do you want to know how you register for classes?* This would be another way to supply a way for continuous discussion. Another way would be to cluster similar questions that lead to different answers. This way the system could pose a follow up question when an answer is provided.

Table 7.1. Example of Query Template Rephrasing

Can you let me know how I <question body>?
Can you let me know how I can <question body>?
Can you let me know how to <question body>?
Let me know how I <question body>?
Let me know how I can <question body>?
Let me know how to <question body>?
Can you tell me how I <question body>?
Can you tell me how I can <question body>?
Can you tell me how to <question body>?
Tell me how I <question body>?
Tell me how I can <question body>?
Tell me how to <question body>?
Can you describe how I <question body>?
Can you describe how I can <question body>?
Can you describe how to <question body>?
Describe how I <question body>.
Describe how I can <question body>.
Describe how to <question body>.
Do you know how I <question body>?
Do you know how I can <question body>?
Do you know how to <question body>?
How can I <question body>?
How do I <question body>?
How to <question body>?
I need to know how I <question body>.
I need to know how I can <question body>.
I need to know how to <question body>.

This could be very interesting because usually a client will continue asking additional questions covering the same topic as that of the leading question. The system could even learn from client feedback what follow up questions that are likely to occur. An example could be a client asking about the price of a product. A common follow up question would be where to purchase the product.

The ContextQA system would greatly benefit from a question typing algorithm. The question typing algorithm could be constructed in two parts. One aspect could use common aspects of question typing usually geared towards factoid type questions. Another question typing algorithm could be targeted towards the restricted domain that the QA system is being designed for. I really wanted to include question typing or question classification as it also is called as part of the ContextQA system. I would expect the reliability to increase even further if this was added. With the addition of question classification it might prove even more important to utilize a neural network to determine the importance of a certain algorithm output. The question typing algorithm should also be running during the collection of new qa pairs. This way the administrator could apply supervised learning to adjust the automatic assignment of question types. Jimmy. Lin (J. Lin 2002) shows that when analyzing the types of questions available in the 2001 TREC competition there are 50 question types that would cover more than 45% of all the questions. This shows that a question typing algorithm can efficiently reduce the number of question-answer candidates. Some non-factoid question type definitions have been defined by Lehnert (1977). Lehnert's, question type taxonomies would be likely to fit the FAQ style questions better because they are mostly non factual (see Table 7.2). The problem is that writing a question classifier for Lehnert's classes would be difficult because they require an understanding of the whole concept of a question. Many if not all features of the question would have to be evaluated to determine the proper class. Most factual question classifier can focus on factual properties such as time or objects. One solution to this problem would be to use a subset of Lehnert's classes that are more easily determined from a sub-set of question features. These classes could then be used in unison with a standard set of factoid type classifier to overall improve classifying how and why type questions.

Table 7.2. Conceptual Question Categories with Examples

Question Categories	Examples
Causal Antecedent	Why did John go to New York? What resulted in John's leaving?
Goal Orientation	For what purposes did John take the book? Why did Mary drop the book?
Enablement	How was John able to eat? What did John need to do in order to leave?
Causal Consequent	What happened when John left? What if I don't leave?
Verification	Did John leave? Did John anything to keep Mary from leaving?
Disjunctive	Was John or Mary here? Is John coming or going?
Instrumental/Procedural	How did John go to New York? What did John use to eat?
Concept Completion	What did John eat? Who gave Mary the book? When did John leave Paris?
Expectational	Why didn't John go to New York? Why isn't John eating?
Judgmental	What should John do to keep Mary from leaving? What should John do now?
Quantification	How many people are there? How ill was John?
Feature Specification	What color are John's eyes? What breed of dog is Rover?
Request	Would you pass the salt? Can you get me my coat?

It would also be interesting to further analyze the additional learning capacity that translation can have to the existing repository. This was covered to some degree in the thesis but could be expanded further. Being able to automatically expand the training set is a very attractive approach.

One final improvement to get better results would be to extend the student advisor test collection. As described earlier the current test collection is not sufficiently large to accurately detect small improvements or deficiencies in new algorithms.

REFERENCES

- Beowulf*. (2005). Retrieved October 9, 2005 from <http://www.beowulf.org>
- Bilotti, M., Katz, B., & Lin, J. (2004). What works better for question answering: Stemming or morphological query expansion? *Proceedings of the SIGIR 2004 Workshop IR4QA: Information Retrieval for Question Answering*, Sheffield, England, 25-29 July 2004, pp. 1-7. New York, New York: ACM Press.
- Breck, E., Light, M., Mann, G.S., Riloff, E., Brown, B., Anand, P., Rooth, M., & Thelen, M. (2001). Looking under the hood: Tools for diagnosing your question answering engine. *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL-2001) Workshop on Open-Domain Question Answering*, Toulouse, France, 9-11 July 2001, pp. 1-8. East Stroudsburg, Pennsylvania: The Association for Computational Linguistics.
- Burger, J., Cardie, C., Chaudhri, V., Gaizauskas, R., Harabagiu, S., Israel, D., et al. (2000). Issues, tasks and program structures to roadmap research in question & answering. *Technical report, National Institute of Standards and Technology*. Retrieved Feb 27, 2007 from http://www.inf.ed.ac.uk/teaching/courses/tts/papers/qa_roadmap.pdf
- Bush, N. *Why you don't need proprietary bot software*. (2001, June). Retrieved April 1, 2007 from <http://www.alicebot.org/articles/bush/dontpayalotforthatbot.html>
- Bush, R. *A basic FidoNet(r) technical standard*. (1995, September). Pacific Systems Group. Retrieved June 15, 2005 from <http://www.ftsc.org/download/docs/fts-0001.016>
- Curtis, J., Matthews, G., & Baxter, D. (2005). On the effective use of Cyc in a question answering system. *Proceedings of the IJCAI Workshop on Knowledge and Reasoning for Answering Questions*. Edinburgh, Scotland, July-August 2005. Austin, Texas: Cycorp.
- Diekema, A.R., Yilmazel O., & Liddy, E.D. (2004). Evaluation of restricted domain question-answering systems. *Proceedings of the ACL 2004 Workshop on Question Answering in Restricted Domains*. Barcelona Spain, 25 July 2004, pp. 2-7. Syracuse, New York: Syracuse University.
- Dumais, S., Banko, M., Brill, E., Lin, J., & Ng, A. (2002). Web question answering: Is more always better? *Proceedings of the 25th ACM SIGIR*, Tampere Finland, 11-15 August 2002, pp. 291-298. New York, New York: ACM Press.
- Experts Exchange*. (2005). Retrieved September 4, 2005 from <http://www.experts-exchange.com>
- Fellbaum, C. (1998). *WordNet: An electronic lexical database*. Cambridge, Massachusetts: MIT Press.
- Google Answers*. (2005). Retrieved June 5, 2005 from <http://answers.google.com/answers>

- Green, B.F., Chomsky, C., & Laughery, K. (1961). BASEBALL: An automatic question answerer. *Proceedings of the Western Joint Computer Conference*, 9-11 May 1961, pp. 219-224. New York: Institute of Radio Engineers.
- Green, L.E.S., Berkeley, E.C., & Gotlieb, C. (1959, October). Conversation with a computer. *Computers and Automation*, 9-11.
- Greenwood, M.A., & Gaizauskas, R. (2003). Using a named entity tagger to generalize surface matching text patterns for question answering. *Proceedings of the Workshop on Natural Language Processing for Question Answering (EACL03)*, Budapest, Hungary, 12-17 April 2003, pp. 29-34. Budapest: EACL.
- Grosz, B.J. (1983). TEAM: A transportable natural-language interface system. *Proceedings of the first Conference on Applied Natural Language Processing*, Santa Monica, California, 1-3 February 1983, pp 39-45. Morristown, New Jersey: Association for Computational Linguistics.
- Hammond, K., Burke, R., Martin, C., & Lytinen, S. (1995). FAQ Finder: a case-based approach to knowledge navigation. *Proceedings of the 11th Conference on Artificial Intelligence for Applications*, 20-23 February 1995, pp. 80-86. Washington DC, Virginia: IEEE Computer Society Press.
- Harlen, W. (2004, May). *Evaluating inquiry-based science developments*. University of Cambridge.
- Hirschman, L., & Gaizauskas, R. (2001). Natural language question answering: The view from here. *Journal of Natural Language Engineering, Special Issue on Question Answering*, 7, (4), 275-300, October.
- Hung, J.C., Wang, C.S., Yang, C.Y., Chiu M.S., & Yee, G. (2005). Applying word sense disambiguation to question answering system for e-learning. *Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA, 2005)*, Tamkang, Taiwan, 28-30 March 2005, pp. 157-162. Washington DC: IEEE Computer Society.
- Java Open Source Spell Checker*. (2006). Retrieved Tue 29, 2006 from <http://jazzy.sourceforge.net>
- Java Servlet Technologies*. (2006). Retrieved Aug 8, 2006 from http://java.sun.com/products/jsp/jsp_jservlet.ds.html
- Kantor, B., & Lapsley, P. (1986, February). *Network news transfer protocol---a proposed standard for the stream-based transmission of news*. Retrieved March 2004 from <ftp://ftp.rfc-editor.org/in-notes/rfc977.txt>
- Katz, B. (1997). Annotating the world wide web using natural language. *Proceedings of the 5th RIAO Conference on Computer Assisted Information Searching on the Internet (RIAO 1997)*, Montreal, Canada, 25-27 June 1997, pp. 136-155. Cambridge, Massachusetts: MIT Press.
- Kukich, K. (2000). Beyond automated essay scoring. *Institute for Electrical and Electronic Engineers (IEEE) Intelligent Systems*, 15, (5), 22-27, October.

- Lancaster, F.W. (1968). *Information retrieval systems: characteristics, testing, and evaluation*. New York: John Wiley & Sons.
- Lehnert, W.G. (1977). A conceptual theory of question answering. *Proceedings of the fifth International Joint Conference on Artificial Intelligence*, Cambridge, Massachusetts, 22-25 August 1977, pp. 158-164. San Francisco, California: Morgan Kaufmann Publishers.
- Lehnert, W.G., Dyer, M.G., Johnson, P.N., Yang, C.J., & Harley, S. (1983). Boris: an experiment in in-depth understanding of narratives. *Artificial Intelligence*, 20, (1):15-62.
- Lenat, D. (1995). CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38, (11), 33-38.
- Lin, J., Quan, D., Sinha, V., Bakshi, K., Huynh, D., Katz, B., & Karger, D.R. (2003). What makes a good answer? the role of context in question answering. *Proceedings of the Ninth IFIP TC13 International Conference on Human-Computer Interaction (INTERACT2003)*, Zürich, Switzerland, 1-5 September 2003. The Netherlands, Amsterdam: IOS Press.
- Lovins, J.B. (1968). Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11, 22-31, March 1.
- Moldovan, D., Harabagiu, S., Girju, R., Morarescu, P., Lacatusu, F., Novischi, A., Badulescu, A., & Bolohan, O. (2002). LCC Tools for question answering. *Proceedings of the TREC-2002 Conference, NIST, Gaithersburg, Maryland*, 19-22 November 2002, pp. 144-154. Washington DC, Virginia: Department of Commerce.
- Moldovan, D., Pasca, M., Harabagiu, S., & Surdeanu, M. (2002). Performance issues and error analysis in an open-domain question answering system. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL-2002)*, Philadelphia, Pennsylvania, 6-12 July 2002, pp. 33-40. New York, New York: ACM Press.
- Morton, T.S. (1999). Using coreference for question answering. *Proceedings of the Workshop 'Coreference and Its Application' (ACL-1999)*, Baltimore, Maryland, 21 June 1999. Maryland: ACL.
- Porter, M.F. (1980). An algorithm for suffix stripping. *Program*, 14, (3), 130-137, July.
- Prager, J., Brown, E., Coden, A., & Radev, D. (2000). Question-answering by predictive annotation. *Proceedings of the Twenty-Third Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Athens, Greece, 24-28 July 2000, pp. 184-191. New York, New York: ACM Press.
- Radev, D.R., Libner, K., & Fan, W. (2002). Getting answers to natural language queries on the web. *Journal of the American Society for Information Science and Technology*, 53, (5), 359-364.
- Radev, D.R., Qi, H., Wu, H., & Fan, W. (2002). Evaluating web-based question answering systems. *Proceedings of LREC*, Las Palmas, Spain, 29-31 May 2002. Luxembourg: ELRA.

- Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24, (5), 513–523.
- Salton, G., & McGill, M.J. (1983). *Introduction to modern information retrieval*. New York: McGraw Hill.
- Schank, R.C., Goldman, N.M., Riesbeck, C.K., & Rieger, C.J. (1975). Inference and paraphrase by computer. *Journal of the ACM*, 22, (3), 309-328, July.
- Sneiders, E. (1999). Automated FAQ answering: continued experience with shallow language understanding. *Proceedings of the 1999 AAAI Fall Symposium on Question Answering Systems*, North Falmouth, Massachusetts, 5-7 November 1999, pp. 97-107. Menlo Park, California: AAAI Press.
- Stop Word List*. (2006). Retrieved Aug 15, 2006 from http://www.dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/stop_words
- SUN N1 Grid Engine*. (2005). Retrieved October 9, 2005 from <http://www.sun.com/software/gridware/>
- Tellex, S., Katz, B., Lin, J., Fernandes, A., & Marton, G. (2003). Quantitative evaluation of passage retrieval algorithms for question answering. *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, Toronto, Canada, 28 July – 1 August 2003, pp. 41-47. New York, New York: ACM Press.
- Turing, A.M. (1950). Computing machinery and intelligence. *Mind*, 59, 433-460.
- Van Rijsbergen, C.J. (1979). *Information retrieval*. London: Butterworths-Heinmann Newton.
- Voorhees, E.M. (1999). Overview of the eighth text retrieval conference (TREC-8). *Proceedings of the Eighth Text Retrieval Conference*, Gaithersburg, Maryland, 17-19 November 1999, pp. 1-24. Washington DC, Virginia: Department of Commerce.
- Voorhees, E.M. (2003). Overview of the TREC 2003 question answering track. *Proceedings of the Twelfth Text REtrieval Conference*. Gaithersburg Maryland, 19-22 November 2003, pp. 54-68. Washington DC, Virginia: Department of Commerce.
- Voorhees, E.M. (2004). Overview of the TREC 2004 question answering track. *Proceedings of the Thirteenth Text REtrieval Conference*, Gaithersburg Maryland, 16-19 November 2004, pp. 12-20. Washington DC, Virginia: Department of Commerce.
- Weizenbaum, J. (1966). Eliza: A computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9, (1), 36-45.
- Whitehead, S.D. (1995). Auto-faq: An experiment in cyberspace leveraging. *Computer Networks and ISDN Systems*, 28, (1-2), 137-146, December.
- Woods, W.A. (1973). Progress in natural language understanding: An application to lunar geology. *Proceedings of the AFIPS Conference*, New York, New York, 4-8 June 1973, pp. 441-450. Montvale, New Jersey: AFIPS Press.

Word List. (2006). Retrieved Aug 03, 2006 from <http://wordlist.sourceforge.net/>

Zaanen, M., Pizzato, L.A., & Moll'a, D. (2005). Classifying sentences using induced structure. *Proceedings of the 12th International Convergence for String Processing and Information Retrieval*, Buenos Aires, Argentina, 2-4 November 2005, pp. 139-150. Heidelberg, Germany: Springer-Verlag.

Zweigenbaum, P. (2003). Question answering in biomedicine. *Proceedings of the EACL2003 workshop on NLP for Question Answering*, Budapest, Hungary, 12-17 April 2003, pp. 41-61. Cambridge, Massachusetts: MIT Press.

APPENDIX A
TECHNICAL SPECIFICATIONS

TECHNICAL SPECIFICATIONS

This section describes the technical details of the system hardware used during the development of the ContextQA system. This section also describes the software used to run the ContextQA system, and software used during development.

HARDWARE

Hardware Specifications

OS	Windows XP
CPU	AMD Athlon 1.25 Ghz
Memory	1.0 GB of PC2100 DDR DRAM
Storage	2 Maxtor 20GB configured in RAID-0

SOFTWARE

Software Specifications

Database	MySQL-Server 5.0 The database contains 1091 questions and 756 answers.
JVM	Java J2SDK 1.6
Webserver	Apache Tomcat 6.0
IDE	Eclipse 3.1, and TogetherJ 3.0
Editor	UltraEdit 13.0
3rd party software	Alicebot Program D, WordNet 2.0, Jazzy 1.0

APPENDIX B

QA ADMINISTRATION INTERFACE

MAIN INTERFACE

In the main interface you are presented with two different sections which are described below.

Create New Question

The first section is designed to add additional question-answer pairs to the knowledge base. Two text fields are available, question and answer. If both fields are filled out and the “Create” button is clicked the new qa-pair will be inserted into the knowledge base if the question does not already exist.

Search for Questions

The second section presents a search field, and a table listing questions. Initially the search field is blank. This results in all questions being presented in the table. Any search terms entered in the search field will result in a Boolean or-condition search. If you search on “A B” the result will be A OR B. Any questions containing either A or B or both will show up in the list. The table has seven columns. Each of the columns is described below:

Main Search Result Table

Edit	This button will cause a popup window to appear where the question and its associated answer can be edited. See the edit question interface section for additional details.
Q-ID	This is the question ID as it is represented in the QA database.
QUESTION	This is the question sentence.
A-ID	This is the answer ID as it is represented in the QA database.
#Q/A	This column represents the number of questions that are linked to the same answer as the question presented in the table row. If this column lists only one, it means that this is the only question that leads to that answer. The higher the number the better. A high number means that several differently formulated questions are pointing to the same answer. Many auxiliary questions per answer will improve the QA systems matching performance.
CREATED	This is the date when the question was created. The date is formatted using the ISO date standard (yyyy-mm-dd).
UPDATED	This is the date when the question was updated. The date is formatted using the ISO date standard (yyyy-mm-dd). If the question has never been updated this field will be blank.

EDIT QUESTION INTERFACE

When a question is selected to be edited in the main interface a popup window appears. The popup window has several sections which are each described in detail below.

Question / Answer

The question with its associated answer is presented as editable text fields. The question field has the following buttons:

Question Controls

Save	Will save the current state of the question, close the popup window, and bring the user back to the main interface.
Re-link	Option to link the question to a different answer This is done by providing a page where all available answers are listed. If a new answer is selected the current answer might end up being deleted if no other questions are associated to that answer.
Delete	Will delete question and possibly also the answer if no other questions are associated to that answer.

The answer text field also has a set of buttons:

Answer Controls

Save	Will save the current state of the answer, close the popup window, and bring the user back to the main interface.
Close	Will close the popup window, and bring the user back to the main interface.

The answer also has a check box which indicates whether or not the answer has been verified for correctness or not.

Create an Additional Question

This section includes a field where an additional question can be typed in. If this question is submitted, it is linked to the answer that is currently being displayed.

Questions Linked to the Same Answer

This section presents a table that lists any other questions that are linked to the current answer. If there are no other questions linked to the current answer this table will be blank. This table also includes edit buttons for each line item, which would bring up the edit question interface for that particular question.

Similar Questions

The similar questions section includes a table which lists questions that are similar to the question currently being edited. The questions listed in this table are not linked to the current answer. Each line item has a link button which will re-link that particular question to the current answer. Each table row will also include a column that indicates whether the answer that particular question points to have been verified or not. Re-linking questions that point to verified answers are not recommended unless the question is being re-linked to another answer that also has been verified.

APPENDIX C
TEST QUESTIONS

QA TEST QUESTIONS

The following table lists an excerpt of questions used when testing the different QA Agents listed in section Agent Results. Each row has an associated answer ID or a negative one if no answer exists in the knowledge repository. These questions were extracted from various websites to ensure that they are separate from any data the system has previously been tested on.

Test Questions with Associated Answer ID's

Question	Answer ID
Do you offer financial support?	133
How important is where the student received his bachelor's degree?	-1
How long will a BS take?	148
What is your minimum GRE requirements?	45
How do I find out if I am meeting your requirements?	-1
What kind of job do I get if I study computer science?	947
Does the College assist me in finding a job?	393
Do you offer a computer repair class?	-1
Do you offer classes online?	1017
Do you offer classes that corresponds to some industry certificates?	-1
When do classes start?	1018
Which are the requirements for foreign students?	-1
What is the student visa procedure?	306
How much do i have to spend per month for accommodation and living expenses?	-1
How do I get into the lab?	609
Who is this program for?	780
How long will it take me to complete the degree?	148
How much does the program cost?	592
Do I have to take the GRE?	43

ABSTRACT OF THE THESIS

ContextQA: Experiments in Interactive Restricted-Domain Question Answering

by

Martin Erik Liljenback

Master of Science in Computer Science

San Diego State University, 2007

The need for more advanced data mining and search engine technologies has been steadily increasing since the introduction of the Internet. With the exponential growth of information available on the web combined with a public that is becoming more educated in search technology, there exists a great need to quickly and efficiently be able to provide results for a large range of very specific questions. The current natural language processing is still in a primitive state. There is no single solution that will be able to provide quality results to the broad range of potential questions by using indexed data extracted from the web. However there exist several ways to provide more efficient results. One way is to develop more extensive ways to interact with users to target results related to the individual's specific needs.

This thesis focuses on a particular field of research that is called Question Answering Systems. In Question Answering the system provide answers on plain text questions through natural language processing, information retrieval, and data mining on structured or unstructured text data. A summary of the research development in this area is provided and also a description of how the algorithms and techniques have evolved over time until we are left in the current state.

Furthermore, I conclude that there are many compelling reasons to build more refined and targeted knowledge bases. With a targeted knowledgebase and knowledge about an individual specific needs, several algorithms can be applied which provides better results and efficiency than that of an open-domain question answering system. I show that index based search engines are far from providing the same level of accuracy as a restricted-domain QA systems. As part of the thesis a complete restricted-domain QA system is developed named ContextQA. A series of experiments are conducted where ContextQA is configured to use different approaches on restricted-domain question answering algorithms. The results show that high accuracy can be obtained within a restricted-domain with limited resources.