WebQA: A Web Querying System Using The QA Approach

by

Sunny K. S. Lam

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Mathematics

in

Computer Science

Waterloo, Ontario, Canada, 2002

I hereby declare that I am the sole author of this thesis.

I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

As the size of the World Wide Web (WWW or Web) has grown, access to the data on the Web has become a significant problem. Data on the Web are managed by many individuals, organizations, and companies, they are stored in many different locations, and adhere to very different formats. These factors contribute to the difficulty of retrieving Web data. The common paradigm of searching and retrieving information on the Web is based on keyword-based search using one or more search engines, and then browsing through the large number of returned URLs. This thesis investigates a declarative query-based approach to Web data retrieval that uses question-answering technology in extracting information from Web sites that are retrieved by search engines. The approach consists of first using meta-search techniques in an open environment to gather candidate responses from search engines and other on-line databases, and then using information extraction techniques to find the answer to the specific question from these candidates. A prototype system, called WebQA, has been developed to test this approach. Testing includes evaluation of its performance as a question-answering system using a well-known evaluation system called TREC-9. Its accuracy using TREC-9 data for simple questions is high and its retrieval performance is good. The system employs an open system architecture allowing for on-going improvements in various aspects.

Acknowledgements

The first person that I would like to sincerely thank is my supervisor and advisor, M. Tamer Özsu, for his useful guidance and support. He always provided help when I needed and I learnt a lot from him. His research skills has led my research in the right direction, which saved me a lot of time. His English skills has helped me in proof reading this thesis and other documents, which increased my written skills. His presentation skills has shown me how to make an effective presentation, which helped me in presenting materials at a conference. His theoretical computing knowledge has given me better algorithms for some components of WebQA, which made WebQA faster. His practical computing knowledge has provided me help in writing LaTeX documents, which is the format of this thesis. Without him, I would not able to develop WebQA and to write this thesis. I knew that I had chosen a right supervisor from the beginning!

The second person that I would like to thank is Karen Anderson. I was her teaching assistant for a first year computing course. She gave me an opportunity to teach in a classroom environment, which was very rare for graduate students. This opportunity had improved my presentation skills and verbal communication skills. Before then, I always felt nervous to speak in front of public, but now I have overcome that fear. Although, I already knew Java and Turing, her lectures helped me build a proper programming style. My teaching assistantship experience at Waterloo was fun!

The third person that I would like to thank is Ning Zhang, who is a fellow graduate student at Waterloo and a close friend of mine. His support and technical knowledge helped me solve many computing problems. He also provided a lot of help on Java. He is the model of graduate students.

I would like to thank Charlie Clarke and Kenneth Salem for reviewing this thesis. Furthermore, I would like thank Charlie Clarke and Vlado Keselj for their expert knowledge in information retrieval and question-answering, which provided me a background for understanding the fields.

I would like to thank my two officemates, Xuerong Tang and Hui Zhang. They were great officemates. Since, we were sharing an office, they were able to provide me technical advices immediately if I needed them. We had fun working together.

I would like to thank Andreas Doms and Mirko Seifert. Andreas implemented the Web interface of WebQA using JSP, allowing users a friendly interface for querying. Both of them provided help on using applications: Visual Age on Java, CVS, and JUnit.

Finally, I would like to thank other students who are working in the database lab. They are Ben Yao, Lei Chen, Khuzaima Daudjee, and Leon Cao. Their support and care created a positive working environment for my study at University of Waterloo.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Problems

The World Wide Web (WWW or Web) was born at the end of the 1980s [BLCL+94]. Since then, it has provided a platform for global and free sharing of information. It hides all the details of communication protocols, machine locations, and operating systems from a user, allowing her to easily move from one page to another accessing information. The amount of information on the Web continues to grow exponentially. The data on the Web is managed by many individuals, organizations, and companies. A study [BYRN99] shows that there are many difficulties in finding information on the Web. They are the following:

- **Distributed data:** Data are widely distributed. They are located at different sites and platforms. The communication links/protocols between machines vary widely. Furthermore, there is no clear topology of data organization.

- **High percentage of volatile data:** Web documents can be added or removed easily on the Web. Change to these documents are not noticed by other sites. There is a very high chance of broken links.

- **Large volume:** The growth of data is exponential. The scaling issue is a big problem.

- **Unstructured data:** Web pages are not well-structured. There is no schema behind them for specifying the semantic structure.

- **Redundant data:** Identical pages are duplicated.

- **Data Inconsistency:** Information in two different pages may conflict with each other. Ensuring the consistency of the data on the Web is complicated by the amount of data.

- **Quality of data:** A lot of Web pages do not involve any editorial process. In other words, data can be false, inaccurate, outdated, or poorly written.

- **Heterogeneous data:** There is no standard format for Web pages. The pages can be written in different formats such as HTML, XML, and multimedia files.

- **Dynamic data:** The content of Web documents change dynamically. It is hard to keep track all the changes on the Web.

1.2 Problem Definition and Motivation

The Web raises many interesting research issues. One such issue is how to find information on the Web. As we know, the most popular way to query the Web is using a search engine. Examples of search engines are Google [Goo02a] and AltaVista [Alt02]. These search engines allow a user to enter a set of keywords and they return a list of ranked links to pages according to their page ranking algorithm. However, search engines are not query systems, which gives rise to problems. First of all, there is a "semantic gap" between what the user wants and the set of keywords that the user specifies [KKS01]. Consequently, the system might not return the current list of documents that the user wants. Even if the relevant documents are returned, the system might locate them at the very end of the list. This means that the user needs to spend quite a large amount of time to go through all the unnecessary documents. Even when the system returns relevant Web pages, the user still needs time to read over these pages to find relevant answers. Moreover, different search engines have different strengths, and the user has to figure out which search engine to use to perform her search. Metasearch systems cope with the last problem by combining different search engines and ranking them; however, they still have the problem of returning a list of documents, not direct answers.

The ideal solution for querying the Web is to have a system that accepts an easy-to-pose query and the system searches many different sources on the Web (including the search engines). At the end, the system returns direct answers. This is similar to what a Database Management System (DBMS) does. This thesis is proposing a new query system called WebQA that takes this approach. WebQA

accepts a natural language (NL) question and returns a list of ranked answers. The natural language input is not essential; one can replace it with a Web query language if and when one gets developed. Furthermore, finding information on the Web is a large problem. We are trying to focus on a subset of the problem: short answers to factual queries.

Building WebQA involves many different techniques from many different areas. We are using question-answering (QA) techniques for finding the category of the NL query and for converting the query into a system understandable language. We are using metasearch techniques for selecting search engines or other data sources. We are using mediator/wrapper techniques to standardize and integrate results from different sources. We are using information retrieval (IR) techniques for filtering out unnecessary result pages from the selected sources. Finally, we use information extraction techniques for extracting answers from the filtered pages.

1.3 Problem Domain

The specific Web searching problem that we investigate is how to provide an effective way to answer short factual queries. We do not deal questions that returns multiple results such as "Who are the basketball players for Toronto Raptors?". We are not concerned with continuous queries, which are persistent queries that allow systems to return answers when the answers become available [CDTW00]. An example continuous query is *"Notify me whenever the Waterloo's temperature drops below zero"*. We also are not concerned with systems that use Web data mining to find answers to queries. Web mining systems analyze all the user queries in the

past and determine what other topics might be of interest to a user depending on the user's query. They allow the return of non-relevant answers as long as they may be of interest to the users. For our research, we only study systems that answer the user queries directly. Finally, we focus only on factual queries. Like databases, we query the facts. For example, *"What is the population of Canada?"*. Users can post factual queries on any topic. We are not concerned with procedural queries. An example of such a query is *"How do I make pancakes?"*. Nor are we concerned with works such as [BYBC+00] that only return Web statistic information. Finally, we assume the users input their queries in correct syntax, which means that we do not study the importance of syntax checking and spell checking over natural language queries.

1.4 Contributions

This thesis introduces a query system that enables a user to query the Web easily. The user specifies a query and WebQA returns the direct answer to the query. The major contribution of WebQA is that it returns actual answers rather than a list of links in response to a query. Currently, the input query is in natural language, but this can be changed to some standard query language like SQL to avoid English ambiguity. The output of the system is not a list of links, but rank of possible answers.

WebQA is an open system with a wrapper/mediator architecture. It has the ability to access any Web data source, such as CIA's World Factbook and weather sites, as well as search engines. A wrapper is written for the data source, converting

its format into a standard format. This way, we can use a mediator to combine and rank all the results from the sources. The user does not need to know the location or the access priorities of the data sources.

The operation of WebQA is as follows. First, the user specifies a query, which is passed to WebQA's first component: Query Parser. The Query Parser has the natural language processing (NLP) ability to determine the question's category. In addition, it converts the query into an internal query which the system can more easily manipulate. Then, the internal query is passed to the second component, Summary Retriever. The job of Summary Retriever is to determine which data sources to query and how to combine the results from the selected sources. The mediator techniques are applied here. Finally, the ranked results that are generated by Summary Retriever are passed to the last component, Answer Extractor. The Answer Extractor component extracts the answers from the source results and returns the answers to the user.

1.5 Possible Uses of WebQA

WebQA is a fast, accurate and easy-to-use system. It provides a convenient way for a user to ask short answer factual questions. There are many possible uses of WebQA, some if which are highlighted in this section.

The system is very good at answering questions involving people. Such questions could be, for example, the inventor of something. Examples of questions can be *"Who invented the telephone?"* and *"Who invented the airplane?"*. These questions can be posed as natural language and WebQA can provide the answer.

A use case may be querying by the elderly, who typically have limited knowledge of computers. They are likely to give up if the search is complicated. WebQA enables them to ask quick questions, such as *"What is the weather condition in Toronto?"*. The system returns the direct answer, rather than requiring them to navigate through a weather site to find the information. She does not have to know how to specify a set of keywords into the text input box. In addition, some weather sites require the user to know the geographic area in order to find the weather for that city [Env02]. But WebQA does not have this unnecessary requirement.

A third use is to embed WebQA in an online natural language information retrieval system such as Ask Jeeves [Ask02]. The user enters a natural language question and the system returns a list of relevant documents to the question. In addition, it shows all the direct pages to similar questions. In either case, it does not return the exact answer. By embedding WebQA into Ask Jeeves, the system can return the direct response to a short factual question and may return documents for more complicated ones. More details on Ask Jeeves are discussed in Section 2.8.2.

1.6 Research Areas

This thesis introduces a new query system that answers short factual questions. It covers many different research areas. First, it covers the area of Web querying. Second, it covers the area of question-answering. The users ask a natural language question and the system does natural language processing and returns the direct answer. Third, it covers the area of metasearch techniques, because WebQA uses

them for querying and ranking different search engines. Fourth, the thesis covers the area of information integration, because it demonstrates how to integrate information from different sources. Fifth, the thesis covers the area of information retrieval, because WebQA uses filtering techniques to filter out unnecessary data. Last, the thesis covers the area of information extraction as WebQA extracts answers from the relevant documents.

1.7 Thesis Organization

The thesis is organized as follows. In Chapter 2, we investigate all the current approaches to solve the Web querying problem. We also address the advantages and disadvantages of each approach. In Chapter 3, we provide a general overview of how WebQA works. We also demonstrate WebQA's architecture and its cache management. Chapters 4 to 6 focus on individual components of WebQA's engine. Chapter 4 covers the Query Parser component. Chapter 5 covers the Summary Retriever component. Chapter 6 covers the Answer Extractor component. Chapter 7 presents the results of performance experiments. Finally, Chapter 8 provides the summary and discusses any future work.

Chapter 2

Background and Literature

2.1 Web Searching

Web searching is related to the field of information retrieval (IR). "IR deals with the representation, storage, organization of, and access to information items" [BYRN99]. The goal of an IR system is to retrieve all the documents which are relevant to a query while retrieving as few non-relevant documents as possible. The user should easily be able to retrieve information of interest to her. There is a difference between information retrieval and data retrieval. In data retrieval, the result of a query must be accurate: it should return the exactly matched data, no more and no less. If there is no change to the database, the result of a query executed at different times should be the same. On the other hand, insignificant error is allows in information retrieval. The main reason for this difference is that information retrieval usually deals with natural language text in queries and data, which is not always well structured and could be semantically ambiguous. Data retrieval

deals with data that has a well-defined structure and semantics (e.g., a relational database).

There are different kinds of query interfaces for the Web. The most popular interface is the keyword expression that search engines or meta-searchers provide. However, keyword expression does not exactly represent what the user wants to search. A keyword expression has different meanings for different users. For example, the keyword expression *"microsoft office"* can be interpreted as the office location in Microsoft or the software product, Microsoft Office. On the other hand, a user intent can be expressed by different keyword expressions. For example, if a user wants to buy a used car, she can express the keyword expression as *"second-hand automobile"* or *"used car"*. Furthermore, a user might want to express a complex query such as *"Which city has the largest population in Canada?"*. It is difficult to express this query in a keyword expression. To present the user intent effectively, we can query by using standard query languages. Standard query languages go beyond keyword search by supporting complex queries. For example, such a query might be *"What is the distance between Waterloo and Toronto"*. However, there is no standard query language for QA systems that search online data. Therefore, we use the natural language for our query interface.

This chapter provides a discussion of the background of this thesis research. There are a number of research efforts addressing the Web querying problem. In this chapter, we classify all the approaches into 7 categories. The categories are keyword search approach, category search approach, database view approach, semistructured querying approach, Web query language approach, learning-based approach, and

question-answering approach. Keyword search and learning-based approaches use keyword expressions as inputs. Database view approach, semistructured querying approach, and Web querying language approach use standard query languages as inputs. Finally, question-answering approach uses natural language as an input.

2.2 Keyword Search Approach

The two main keyword search approaches are search engines and metasearchers. Both of them require a user to enter a list of keywords. These keywords are descriptive words that the result pages will contain. The more words the user enters, the narrower is the subset of pages that will be returned. The user should enter more obvious words first, then add more words to narrow down the search result. She should enter the words that will likely appear in the result pages. Since, stopwords (e.g., a, an, of, the) are very common in all Web pages, the user should not use them in the list of keywords. However, there are cases where it may be necessary to enter stopwords. For example, the user might want to search for a title that contains a stopword. In this case, it is customary to designate the search phrase using, perhaps, double quotes around the title.

The following sections will show the two approaches of keyword search category. They are search engines and metasearchers.

2.2.1 Search Engines

A search engine is a system that takes, as input, a list of keywords from a user and returns a list of related hyperlinks (with descriptions) to the user. Examples

of search engines are AllTheWeb.com [All02], AltaVista [Alt02], Excite [Exc02], Google [Goo02a], Overture [Ove02], Teoma [Teo02], and Yahoo [Yah02b]. The user accesses a search engine through a Web page. As one may notice, most of the search engines cannot store all Web pages in its database, because it would require a huge amount of storage. Instead, it stores a list of indices that point to the associated remote pages. An index uses variants of the inverted file (a list of sorted words) [BYRN99]. Each word has pointers that point to the related pages. Each pointer associates a short description about the pointed page. When a search is executed, a search engine uses a binary search to find the appropriate pointers according to the given list of keywords. Then it uses a page-ranking algorithm to rank the matched pointers. Later on, it returns the URLs and the descriptions from the top score pointers. The remaining question is how a search engine retrieves different pages from the Web. Every search engine has multiple agents called *Web crawlers* that get page information from the Web. Different Web crawlers have different crawling techniques. The simplest one is to start with a set of URLs and from these extract other URLs recursively by crawling the links inside the current page. The crawling can be done in a breath-first or depth-first manner. The breath-first manner covers wide but shallow pages and the depth-first manner produces deep but narrow pages. Of course, there are other techniques of Web crawling. Information can be found in [BYRN99] and [BP98].

2.2.2 Metasearchers

A metasearcher is a Web server that takes a given query from the user and sends it to multiple heterogeneous search engines. The metasearcher then collects the answers and returns a unified result to the user. It has the ability to sort the result by different attributes such as host, keyword, date, and popularity. Examples of metasearchers are Copernic [Cop02], Dogpile [Dog02], MetaCrawler [Met02], and Mamma [Mam02]. Different metasearchers have different ways to unify results and translate the user query to the specific query languages of each search engines. The user can access a metasearcher through client software or a Web page. Each search engine covers a smaller percentage of the Web. The goal of a metasearcher is to cover more Web pages than a single search engine by combining different search engines together.

2.2.3 Comments on Keyword Search Approach

Although the keyword search is the most popular way for searching the Web, it is not the optimal way for Web searching. The limitations of using keyword search are that it does not offer support for a global view of information in the Web. The keyword expression cannot fully represent the user's intent. There are multiple ways to enter a list of keywords for the same query and a list of keywords can be interpreted in multiple ways. There is no direct relationship between a user query and the keyword expression. Finally, keyword search is not sufficient enough to post complex queries (e.g., *"What is the distance between Toronto and Waterloo?"*). We need another method for these types of queries.

2.3 Category Search Approach

Category search addresses one of the problems of using keyword search, namely the lack of a global view of the Web. Category search is also known as Web directory, catalogs, yellow pages, and subject directories. There are a number of public Web directories available: dmoz [dmo02], LookSmart [Loo02], and Yahoo [Yah02b]. The Web directory is a hierarchical taxonomy, which classifies human knowledge [BYRN99]. Although, the taxonomy is typically displayed as a tree, since some categories are cross referenced, it is actually a directed acyclic graph.

If we know what category we are looking for, then the Web directory is a useful tool. However, not all Web pages can be classified, so the user can use the directory for searching. Moreover, natural language processing cannot be 100% effective for categorizing Web pages. We need to depend on human resource for judging the submitted pages, which may not be efficient or scalable. Finally, some pages change over time, so keeping the directory up-to-date involves significant overhead.

2.4 Database View Approach

Declarative querying and efficient execution of these queries has been a main area of focus in the development of database technology. It would be very beneficial if the database techniques can be applied to the Web. In this way, accessing the Web can be treated to a certain extent, similar to accessing a large database. The difficulty is that DBMS querying requires a schema. Furthermore, DBMS data representation (e.g., tables) are usually different than Web data structures. The question is how

to convert the Web data structures into database structures. One approach is to generate database views for the Web. The main goal of using the database views is that users do not worry about the physical storage of Web data. The user can perform Web querying based on the generated views. The query system handles the data integration from different relevant sources and it combines information into a view format. The following discusses the projects that take this approach.

Information Manifold (IM) [LRO96] is a data integration system that generates database views for querying. Although the Web is dominated by unstructured documents, there are still many structured information sources on the Web. Examples of these sources are name servers, bibliographic sources, product information, stock market information, real estate, and automobiles [LRO96]. The idea of IM is to combine all such sources into a database. The interfaces to these sources typically are fill-out forms. Their HTML code is usually well structured. IM tries to provide a uniform interface for structured Web site. The user does not need to know which sources to query, how to query each source, and how to combine the results from each source. The new system should handle all the details.

IM has the ability to determine the Web source dependence, an output of one source can be an input of another source. For example, let us say we want to retrieve the prices of all the cars of a given car type. Assume there are two sources available: Source 1 accepts a car type and returns all the models for that car type, Source 2 accepts a car model and returns the price of the car. If we query each source independently, we would not able to answer the query. However, if we query Source 1 first and get a list of car models, then we can use the output of Source 1

as an input of Source 2. This way, we are able to get our desired information.

Since the number of sources that IM supports is over 100, it is necessary to prune all the non-relevant sources. IM has effective techniques for pruning sources. It stores information on the query capabilities on each source. Therefore, it knows the interface requirements and the queries that the source can handle. Furthermore, it provides an algorithm for generating an efficient query plan for accessing these sources.

Araneus [AMM97] is a project for generating hypertextual views for representing portions of the Web. Data is extracted from the information sources. Given a database structure, structured views are first generated over the Web data. These generated views are re-organized and integrated by using relational technology. The research uses the Araneus Data Model (ADM) for describing the structure of Web documents. Based on the ADM, the research uses the Ulixes language for defining database views. ADM is a page oriented model, that is used to describe the structure of a set of homogeneous pages in a Web site. It addresses the characteristics of HTML pages, each of which has an identifier, the URL, and other attributes (e.g., text, images, etc). Ulixes is used for building relational views over the Web. Based on the description of the site provided by ADM, one can use the navigational expression (i.e., paths through pages) for navigating Web data. We start with some Web pages and navigate until the desired data is found.

Web-Supported Queries (WSQ) [GW00] generates database views for commercial search engines. There are two views for each search engine. They are *WebPages* and *WebCount* virtual tables. The *WebPages* is an infinite virtual table that con-

tains all of the URLs returned by the search engine for the keyword expression. *WebCount* is an aggregate view over *WebPages*. It contains the total number of URLs returned by the search engine for the input keyword list. The tuples in both views represent Web search engine results. WSQ does not perform data cleaning or filtering on the Web data. It enables users to write SQL queries. The system searches its local databases and the available search engines by converting the SQL to keyword expression. The searches on local databases and search engines are done concurrently. The results from the search engines are stored into the virtual tables mentioned above. The result tuples from the local databases and the result tuples from the virtual tables can be combined and return to the user.

There are other works that generate views for querying. However, they are not based on Web data. Abiteboul et al [ACM93] introduced the notion of a structuring schema that consists of database schema and grammar annotated with database programs. It specifies how terminals and non-terminals in the grammar are mapped to the schema. It has the ability to convert structured data stored in files to traditional database systems. Christophides et al [CACS94] provide a technique to convert SGML (superset of HTML) documents to object-oriented databases (OODB). It maps from the SGML DTD to OODB schema.

2.5 Semistructured Data Querying Approach

The semistructured modelling approach treats the Web as a collection of data that is partially structured and represents it as an edge directed graph model. Data may have some structure, however the structure is not as rigid, regular, or complete as

that of traditional databases [AQM+97]. The users do not need to be aware of the complete structure when they perform querying. Therefore, expressing a query should not require full knowledge of the structure.

There are two issues that need to be addressed regarding this approach. First, we need to find a way to transform Web data into a semistructured data model. Second, we need to study languages for querying semistructured data. The obvious approach is to use relational engine to store semistructured data and use SQL for querying. Unfortunately, this technique requires predefined schemas for querying, in which the users do not have the knowledge. However, there are a number of semistructured languages available today for querying semistructured data. The languages that we have investigated are Lorel [AQM+97], UnQL [BDHS96], and StruQL [FFL97]. In this section, we first discuss a technique for integrating Web data sources into a semistructured data model. Then we discuss each semistructured language and its data model.

2.5.1 Transforming the Web into Semistructured Data Model

In distributed database systems, mediators and wrappers (or translators) are used for data integration. They are used when querying on multiple heterogeneous databases. Of course, we can use this approach for generating semistructured data. TSIMMIS [CGMH+94, HGMI+95], which stands for "The Stanford-IBM Manager of Multiple Information Sources", provides a mediator/wrapper framework and tools for information integration. It integrates structured and unstructured data sources by representing their data in a common data model called OEM. OEM is

a semistructured model that is a self-describing (i.e., tagged) object model. It has labels, types, values, and optional identifier. We discuss OEM in Section 2.5.2.

For each information source that is registered in TSIMMIS, there is a wrapper built on top of it. The wrapper's responsibility is to perform the logical translation between the common model and the source. It converts queries over information in common model into requests the source can understand. After the source finishes its execution, the wrapper takes the result and converts it back to a representation in the common model.

Mediators are built on the top of wrappers. A mediator can be associated with multiple wrappers, but a wrapper can be associated with only one mediator. In addition, a mediator can be built on the top of other mediators if necessary. The job of mediator is to embody the knowledge that is necessary for processing a specific type of information. In other words, given a user query in common format (i.e., OEM-QL), it determines which information sources should be queried and sends the query to the appropriate wrappers. Furthermore, the mediator integrates the results that are returned by the wrappers.

For this approach to scale, it is necessary to automate the wrapper generation process. Subsequent work in TSIMMIS and elsewhere has partially addressed this issue.

2.5.2 OEM

OEM (Object Exchange Model) [PGMW95] is a semistructured data model. It's strength is its simplicity and flexibility as a self-describing model that does not

```
Guide &12
    restaurant &19
        category &17 ''gourmet''
        name &13 ''Chef Chu''
        address &14
            street &44 ''El Camino Real''
            city &15 ''Palo Alto''
            zipcode &16 92310
        nearby_eating_place &35
        nearby_eating_place &77
    restaurant &35
        category &66 ''Vietnamese''
        name &17 ''Saigon''
        address &23 ''Mountain View''
        address &35 ''Menlo Park''
        nearby_eating_place &19
        zipcode &54 ''92310''
        price &55 ''cheap''
    restaurant &77
        category &79 ''fast food''
        name &80 ''McDonald's''
        price &55
```

Figure 2.1: An sample OEM database [AQM+97]

require a schema. In OEM, all entities are objects. Each object has a object identifier (*oid*) that distinguishes it from other objects. There are two types of objects: *atomic* and *complex*. An atomic object contains a primitive value such as an integer, a real, or a string. A complex object contains a set of other objects, which can themselves be atomic or complex. A complex object is denoted as a *(label, oid)* pair, e.g., `(city, &15)`. Figure 2.1, taken from [AQM+97], shows an example of an OEM database. OEM can be represented as an edge-labelled graph. The graph representation of the database in Figure 2.1 is given of Figure 2.2.

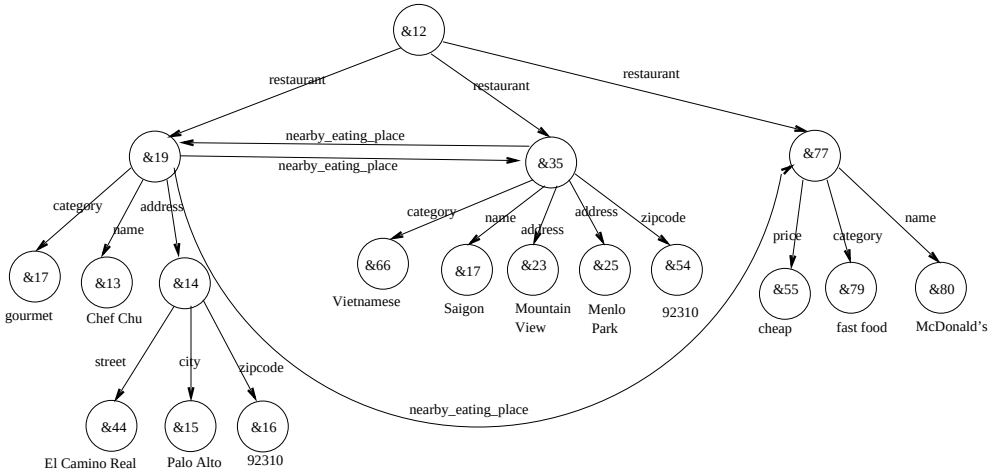

Figure 2.2: The corresponding OEM graph for the OEM database (Figure 2.1) [AQM+97]

Each line in the OEM database shows an object label and an object identifier. The atomic object has a label, an object identifier, a value, e.g., (`category &17 ''gourmet''`). The complex object is followed by subsequent indented lines. Each of the indented lines is another object (sub-object). The sub-object can be an atomic object or a complex object. For example, the complex object with oid &14 consists of three other sub-objects `(street, &44), (city, &15),` and `(zipcode, &16)`. As we can see at the beginning of the line in Figure 2.1, `Guide &12` denotes the name of the object that contains a set of restaurants.

OEM does not have a strict structure like relational tables. The data in OEM come from different heterogeneous sources, and their integration requires flexibility. As we can see, `restaurant` in Figure 2.1 can have zero, one, or multiple addresses. The address can be an atomic object or a complex object. The zip code can be written as a string or an integer. There is no specific type required for each object.

The following is a formal definition of an OEM database [AQM+97].

Definition *(OEM Database)* An OEM schema consists of a finite set of names R. An OEM instance of R consists of: (i) a finite labelled graph $(V_a \cup V_c, E)$ where V_a and V_c are disjoint sets of oid's corresponding respectively to atomic and complex objects, and the edges in E are labelled by strings; (ii) a name function from R to $V_a \cup V_c$; and (iii) a value function val that maps the objects in V_a to atomic values. The instance must also satisfy the following two conditions:

1. Atomic vertices have no outgoing edges.
2. Each vertex is reachable from object *name(N)* for some name N in R.

2.5.3 Lorel

So far, we have examined the technique for transforming Web data to semistructured data. We now investigate the proposed semistructured query languages. The first language that we look at is Lorel [AQM+97], which is the query language for Lore [MAGW97]. Lore is a prototype that provides a convenient and efficient way for storing, querying, and updating semistructured data. Lorel is designed for querying semistructured data in an OEM structured database. It is an extension of OQL [Cat94]. The additional features that Lorel supports are type coercion and powerful path expression. In the following, the language will be described by focusing on the language extensions.

Type coercion is used for equality. It relaxes the strong typing that is used in OQL. This is an important feature, because semistructured data are not typed,

irregularly typed, or may even miss fields. Consider the query "find the addresses of all restaurants that have zip code 92310 over the database of Figure 2.1" [AQM+97]. The Lorel statement is written as follows:

```
select Guide.restaurant.address

where Guide.restaurant.address.zipcode = 92310
```

The zip code can be written in integer or string and Lorel will coerce it accordingly. The user does not have to worry about the type of the data. Some restaurants may not have zip codes, such as the restaurant with oid &77. Lorel does not consider this as error (where SQL and OQL do) and does not include it in the result. Finally, Lorel can handle cases where some restaurants have multiple addresses or multiple zip codes. The results for the above Lorel query are the addresses with oid &14 and &35.

Besides type coercion, Lorel also supports powerful and flexible path expressions. The advantage of this feature is that users are not required to have a precise knowledge of the database structure. Path expressions are built from labels and wildcards (place-holders) using regular expression. For example, a user is looking for a cheap restaurant. She notices that she needs to grep the word "cheap", but does not know where to grep (price, description, or other objects). In addition, the user wants the system to return the matched zip codes, however she does not know that the zip code is sub-object of an address or sub-object of a restaurant. Lorel is able to express these queries because of its powerful path expression. The corresponding Lorel statement is as follows [AQM+97]:

```
select Guide.restaurant(.address)?.zipcode
```

```
where Guide.restaurant.% grep ''cheap''
```

The "?" after `.address` means the address is optional. The "`%`" is a wildcard, which matches any subobject of restaurant. The word "`grep`" is a comparison comparator. It returns true if the string "cheap" appears anywhere in that subobject value. The According to the OEM database in Figure 2.1, the match result is the zip code 92310.

Schema for Semistructured Data Model

Although we can perform querying on OEM by using Lorel, it would be better if we have a schema for querying. The advantage of this is that a schema enables users to understand the structure of the database, so the users can form meaningful queries over it. In addition, a query process relies on the schemas for generating better query access plan. A DataGuide [GW97] provides concise and accurate structural summaries of semistructured databases and can be used as a schema. The generated schemas are dynamic, which is useful for browsing database structure, formulating queries, storing statistical information, and enable query optimization.

2.5.4 UnQL

Another semistructured query language, which is similar to Lorel, is UnQL [BDHS96]. UnQL is a tree-traversing language that queries a labelled tree. The primary feature of UnQL is *traverse*, which restructures the tree to arbitrary depth. Like Lorel, it does not need a schema for querying. It is based on the premise that schema should not be used for browsing purposes even if we are dealing with well-structured

data. For example, it is difficult to write a relational algebra expression to express a search for all the character strings in the database to determine which tables contain a particular term.

The data model of UnQL is also an edge-labelled graph, similar to OEM. The following is the syntax of the data model (i.e., UnQL expression).

- $\{\}$: the empty tree
- $\{l \Rightarrow t\}$: the tree whose root has an outgoing edge labelled l attached to the subtree t
- $t_1 \cup t_2 = \{t_1, t_2\}$: the union of tree t_1 and t_2 formed by coalescing the roots of t_1 and t_2.

For example, if we specify the UnQL expression, $\{A \Rightarrow \{B \Rightarrow \{\}, C \Rightarrow \{\}\}, D \Rightarrow \{\}\}$, then we have the tree shown in Figure 2.3. For another example, if we specify the following UnQL expression, then we have the tree shown in Figure 2.4.

$\{R1 \Rightarrow \{Tup \Rightarrow \{A \Rightarrow$ "a", $B \Rightarrow 2, C \Rightarrow 3\}, Tup \Rightarrow \{A \Rightarrow$ "b", $B \Rightarrow$
$4, C \Rightarrow 5\}\}$,
$R2 \Rightarrow \{Tup \Rightarrow \{C \Rightarrow 3, D \Rightarrow$ "c"$\}, Tup \Rightarrow \{C \Rightarrow 5, D \Rightarrow$ "d"$\}, Tup \Rightarrow$
$\{C \Rightarrow 5, D \Rightarrow$ "e"$\}\}\}$

An UnQL query is generally of the following form:

`select` E

`where` $C_1, C_2, ..., C_m$

where E is a UnQL expression and each C_i is either a condition or a generator. A condition is a predicates on labels such as $l_1 = l_2$. A generator has the form

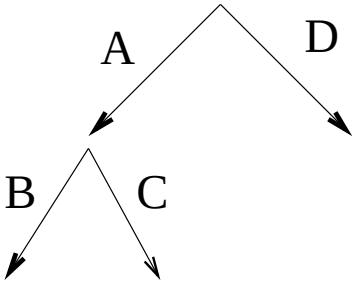

Figure 2.3: An example edge-labelled tree of UnQL

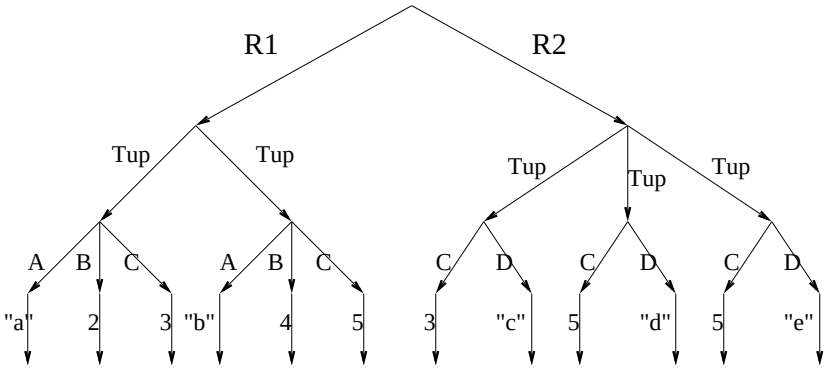

Figure 2.4: An example edge-labelled tree of UnQL

$P \leftarrow T$, where P is a pattern and T is a UnQL expression. A pattern is either a tree constant, a tree variable, or $\{R_1 \Rightarrow P_1, \ldots, R_m \Rightarrow P_m\}$, where R_i is either a variable or a regular expression and P_i is a pattern.

An UnQL query operates on a tree, and returns a tree. An example of UnQL query is the following. It uses the database shown in Figure 2.4.

`select` t

`where` $R1 \Rightarrow \backslash t \leftarrow DB$,

The meaning of the above query is *'Give me all the trees t such that DB has a parent node, where the edge between the parent node and DB is* $R1$. There are two trees that match this condition, $\{Tup \Rightarrow \{A \Rightarrow$ "a", $B \Rightarrow 2, C \Rightarrow 3\}\}$ and $\{Tup \Rightarrow \{A \Rightarrow$ "b", $B \Rightarrow 4, C \Rightarrow 5\}\}$. UnQL returns the union of the two trees of the two trees as the result. Therefore, the result of the query is

$$\{\{Tup \Rightarrow \{A \Rightarrow \text{“}a\text{”}, B \Rightarrow 2, C \Rightarrow 3\}\} \cup \{Tup \Rightarrow \{A \Rightarrow \text{“}b\text{”}, B \Rightarrow 4, C \Rightarrow 5\}\}$$

2.5.5 StruQL

Site TRansformation Und Query Language (StruQL) [FFL97] is a query language for STRUDEL web site management system. STRUDEL uses database concepts to build and manage Web sites. STRUDEL aims to solve the tedious task for changing and restructuring a Web site. It is able to separate the logical view of information at a Web site, the physical structure of Web files and the graphical presentation of Web pages.

StruQL can define the integrated view of data. It is able to define the structure of Web sites. It has the ability to construct graphs for outlining the structure of

Web sites. More importantly, it can query on a semistructured data model. The data model that StruQL uses is similar to OEM. An object in the STRUDEL data model is either a node with object identifier or a atomic value such as an integer, a real, a string, etc. The objects are connected by edges labelled with attribute names in string. The data model can represent a logical view of a Web site. It has the ability to integrate different sources on the Web, like TSIMMIS and Information Manifold. It can be shown to the user as a Web page. Besides the data graph, STRUDEL also has collections, where a collection is a set of nodes. The collection is used as a starting point to the data graph. For example, a collection *Root(x)* is a single node x, where x is the graph's root. To demonstrate StruQL, we show a simple query that asks "all Postscript papers directly accessible from home pages" [FFL97]:

`where` HomePages(p), $p \rightarrow$ "Paper" $\rightarrow q$, isPostScript(q)

`collect` PostscriptPages(q)

In the `where` clause, *Homepages(p)* is a collection of home pages denoted by p, which provides an entry point to the graph. $p \rightarrow$ *"Paper"* $\rightarrow q$ means that node p is connected to node q by an edge labelled *"Paper"*. *isPostScript(q)* is a predicate where node q is a postscript file. In the `collect` clause, *PostscriptPages(q)* means the result is a collection of postscript pages.

2.5.6 Other Semistructured Languages and Data Models

In this section, we have studied three semistructured query languages (Lorel, UnQL, StruQL) and their semistructured data models. Their data models are edge-labelled

graphs and their syntaxes on query languages are different, but all operate on these graphs. Besides these, there are other semistructured languages such as MSL [PAGM96, PGMU96]. MSL is specifically designed for querying OEM. It is a rule-based language, where goal is to specify the integration of data retrieved from different sources [AQM+97].

Furthermore, there are other semistructured data models that we have not studied, such as [DBJ99], that has a set of descriptive properties instead of single value in each label on the edge. This model provides more information and restriction on querying. For example, an edge has a property `price`. Then the viewer must pay the cost in order to access nodes further down the tree.

Another data model which is similar to semistructured data model is the graph model. Like semistructured data model, we can treat the Web as labelled graphs. However, their model is not specifically designed for the Web. Graph query languages have been developed such as G [CMW87], G+ [CMW88], GraphLog [CM90], and GOOD [PdBA+92, PTdB93] for querying graphs. The difference between the graph query language and the semistructured query language is that the semistructured query language emphasize the ability to query the schema of the data, and the ability to accommodate irregularities in the data, such as missing or repeated fields, heterogeneous records [FLM98].

2.6 Web Query Language Approach

Semistructured query languages are designed for querying semistructured data models. The labelled edges in the data models do not address a difference between a

document pointing to one of its part and a document pointing to another document. In this section, we study a number of different query languages for Web data. They combine the content-based queries (e.g., keyword expressions) and structure-based queries (e.g., SQLs). They are WebSQL [MMM97], W3QL [KS95], WebLog [LSS96], WebOQL [AM98]. WebSQL introduces the query locality and provides a formal semantics. W3QL focuses on extensibility. WebLog is based on a Datalog-like syntax. WebOQL supports a large class of data restructuring operations.

2.6.1 WebSQL

WebSQL [MMM97] is a query language for searching the Web. The system queries different index servers for retrieving results. The users do not need to be aware of different index servers. They do not need to know the query capabilities and the query interface of each index server. WebSQL provides a uniform interface for querying. It has the ability to analyze a given query and decide which index servers should be used for retrieving relevant documents. The choice is based on both the ability to return relevant results, and query costs. Like a metasearcher, it has the ability to choose index servers (search engines) for querying and able to combine the results from them. WebSQL supports both content-based querying and structure-based querying. The syntax of structured-based querying is very similar to SQL, which has `SELECT`, `FROM`, and `WHERE` clauses. If necessary, it navigates the results returned from the index servers by browsing different hyperlinks.

In WebSQL, there is a list of attributes that is associated with each Web document. The list can be associated with a virtual relation as follows:

Document[url, title, text, type, length, modif]

The *url* is the URL for identifying each Web document. It is the primary key of the relation. The *title* is the title of the Web document. The *text* is the text that appears in the document. The *type* is the type of the document such as HTML, PDF, image, postscript file, etc. The *length* is the size of the document. The *modif* is the last modification date of the document. Except *url*, all other attributes can have *null* values.

The following example demonstrates WebSQL. The example is to find all HTML documents that contain the word "hypertext" [MMM97]. We want to return the URL, the title, the length and the last modification date of all matched documents. The associated WebSQL statement is the following:

`SELECT` d.url, d.title, d.length, d.modif
`FROM` Document d `SUCH THAT` d `MENTIONS` "hypertext"
`WHERE` d.type = "text/html";

In the `SELECT` clause, we would like to return the URL, the title, the length, and the last modified date of object d. The `FROM` clause specifies that d is a document such that d contains the word "hypertext". The `WHERE` clause specifies the type of document d, which should be a HTML page.

WebSQL has the ability to retrieve information based on the content of Web documents. It also address the hypertext structure of the Web. A hyperlink (also known as anchor) has the following attributes.

Anchor[base, href, label]

The *base* is the URL of the HTML document that contains the link. The *href* is the URL of the referred Web document. The *label* is the text description on the link.

For demonstrating the ability for querying the document structure, here is an example: A student wants to find all links to universities from documents about "survey". The associated WebSQL statement is the following:

```
SELECT y.label, y.href
FROM Document x MENTIONS "survey"
        Anchor y SUCH THAT y.base = x
WHERE y.label CONTAINS "applet";
```

2.6.2 W3QL

W3QL [KS95] is another query language that is specifically designed for querying the Web. Like WebSQL, it has an SQL-like syntax and addresses the content and structure of the Web. It integrates new utilities and existing Unix tools. For example, it uses Unix standard services (such as grep) for identifying, filtering, and formatting data. It has a number of display options which includes relational view, special graph, HTML browser presentation, and Unix directory tree representation. The default output option is the relational view. It uses the database view concepts for simplifying the result information. It provides a facility for maintaining views, so the results are automatically refreshed following changes.

Here is an example of W3QL. The user wants to ask for articles by "A. Einstein". The format of the articles is Latex. The answer is written as an HTML page which

contains all the matched URLs of pages that might contain the information.

```
SELECT cap n/* result;
FROM n;
WHERE SQLCOND (n.format = Latex)
    AND (n.author = "A.Einstein");
```

The SELECT clause specifies that all the matched articles should be saved as files in the *result* directory.

2.6.3 WebLog

The third Web query language that we discuss is WebLog [LSS96]. WebLog is logic-based language. It is inspired by SchemaLog [LSS93] that provides interoperability in multidatabase systems. The syntax of SchemaLog and WebLog are very similar. Like all other Web query languages, WebLog supports content-based and structure-based queries. For querying, it exploits partial knowledge that the users might have for querying. For example, a user might know that the desired information can be found by navigating a specific Web site, which a keyword expression cannot handle. WebLog allows users to express what they know in order to find their information. Besides the query ability, WebLog also supports information restructuring. For example, a user might want to group all the matched links by the same month. WebLog has the ability to handle the dynamic nature of Web data. It uses external libraries for string processing and document analysis.

2.6.4 WebOQL

The last Web query language that we discuss is WebOQL [AM98]. Its schema-less data model is based on ordered arc-labelled trees, called hypertrees. A hypertree has two types of arcs, internal and external. An internal arc represents a structured object and is denoted by a full line arrow. An external arc represents a reference (i.e., hyperlink) among objects and is denoted by a dotted line arrow. Each arc is labelled with a record. A record consists of at least one field. Each field has a field name and a content for the field. The external arc must have a URL field. A hypertree can represent a document, a relational table, a directory hierarchy, etc. Figure 2.5 shows an example of a hypertree that represents a relational table. Figure 2.6 shows another example of a hypertree that models a HTML document. Figure 2.7 shows a hypertree that models a portion of the Web. Besides the hypertree, another main element in WebOQL is a *web*. A *web* is a set of hypertrees, each with an associated URL. It is used to model a small set of related pages (e.g., a recipe), a larger set of related pages (e.g., a local area network), or the WWW. WebOQL is a functional query language, which is able to navigate, query, and restructure the hypertrees.

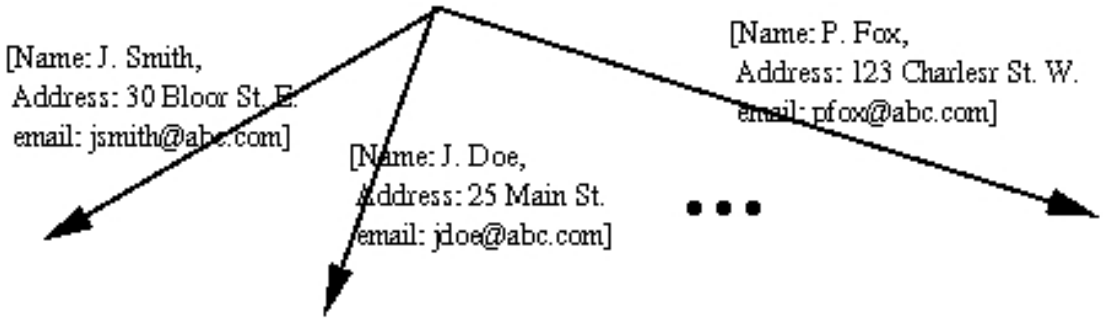

Figure 2.5: A hypertree that represents a relational table

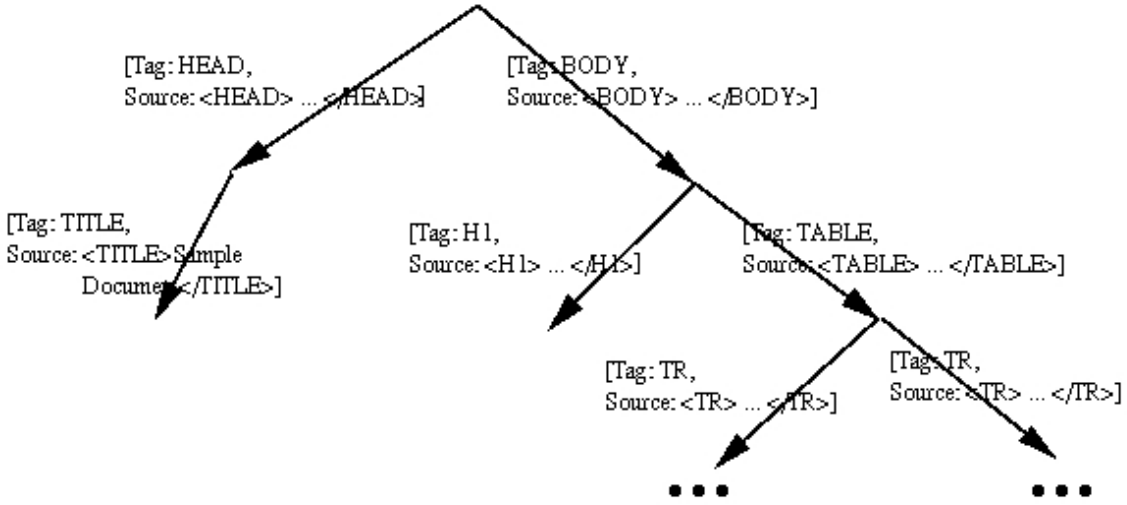

Figure 2.6: A hypertree that represents a HTML document

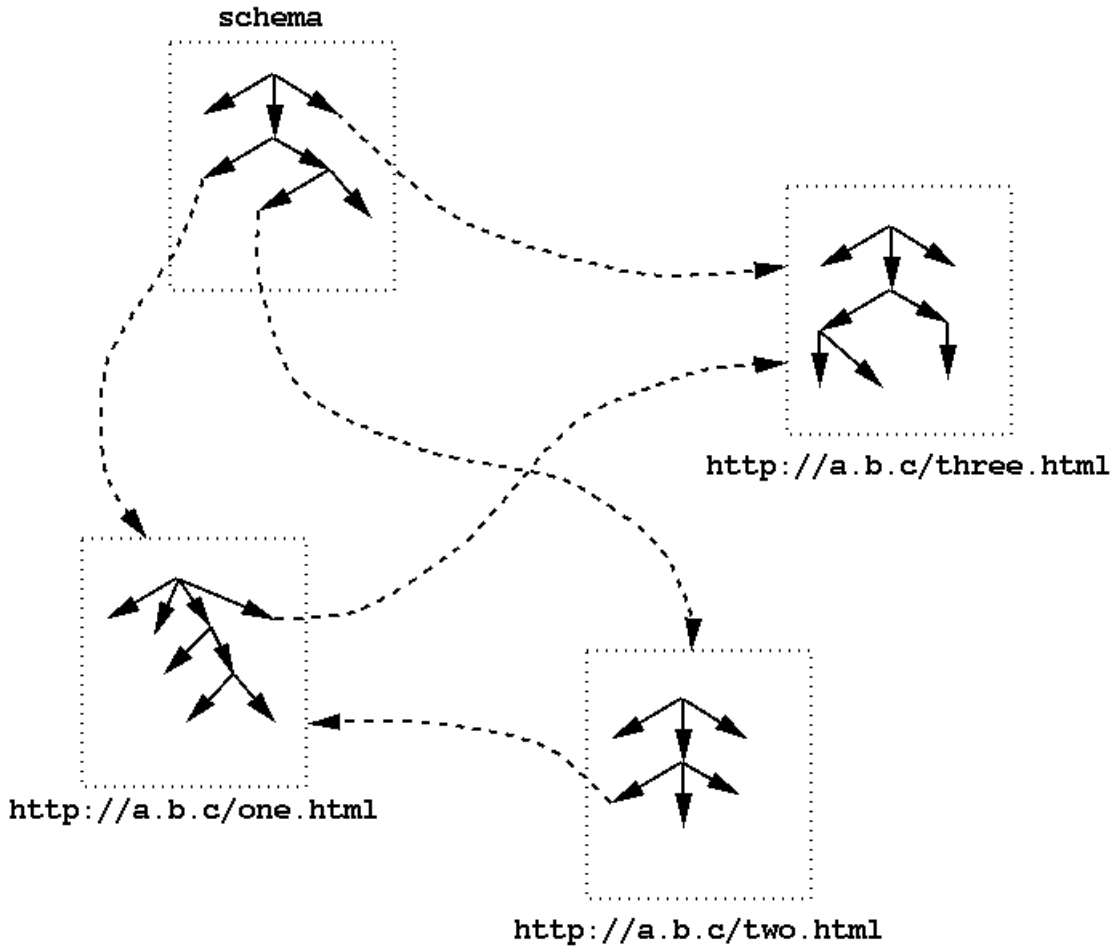

Figure 2.7: A hypertree that represents a portion of the Web

2.7 Learning-based Approach

One of the problems in keyword search is that keywords are insufficient to express users' search intent. This causes too many non-relevant results to be returned by search engines (low precision) [DLCT00]. There is a semantic gap between the keywords specified by the users and how the search engines interpret the keywords [KKS01]. Another problem is that the ranking of the search results is not based on the importance of user's preference. Furthermore, a word might have different meanings, therefore search engines may return non-relevant results.

The learning-based approach is introduced to solve these problems. Learning-based systems learn about the users' query requirements and provide efficient ways to locate the information during a query process. While the users browse the search results, the systems learn what the user's need and return more relevant results in subsequent searches. The process of Web searching is hidden from the users. However, the systems might ask users for future results. Some of them ask the users to rate Web pages for other users. Often, Web searching is done by calling commercial search engines. Searching is based on the users' intent, users submit query profiles to the learning-based systems, which use them in deciding what keyword expression to submit to search engines and how the systems navigate the result pages from the search engines. The system reorganizes the search results by filtering out or adding more information that match the users' need.

There has been much research following this approach. InfoSpider [MB98] takes an ecological model, which uses the idea of how ants behave when they searching for food without centralized control. It searches the Web by traversing hyperlinks

in an autonomous and intelligent fashion. It uses the search engines for providing a starting point of the search.

WebWatcher [JF97] emulates a tour guide in a museum. The agent in WebWatcher guides the user to the appropriate information. Guidance is based on the agent's knowledge of the user's interest and how other users interact with the items in the past. The agent suggests appropriate hyperlinks and learns from past experience for improving its guiding skill.

Fab [Bal97] provides Web page recommendation service to the users. It recommends Web pages that users may browse similar to how newspapers recommend which movies to watch. The recommended service concerns the importance of the shared interests among different users. Since there are a lot of research which has been focused on analyzing document content, Fab uses it for improving the recommendation service.

Syskill & Webert [PMB96] learns the user's profile to determine the user's interest. The agent provides two Web searching choices. In the first choice, the user rates some of the return Web pages and the agent determines what other pages might be of interest to the user. In the second choice, the agent returns all the relevant pages based on the user's profile and the user rates the returned pages for future improvement. The rating is a three point scale: hot, lukewarm and cold.

WebSifter II [KKS01] permits a user to submit her search intent into a tree-structured representation scheme that is a personalized search taxonomy. It consults a Web taxonomy agent to refine the terms that are specified in the tree. The reason for consulting a Web taxonomy agent is that a single word might have mul-

tiple meanings, which affects the search result. The tree is used to identify which meaning is the user's intent. In the end, the returned pages are rated according to the user's preferences. The preferences are semantic relevance, syntactic relevance, categorical match, page popularity and authority rating. Like all other agent-based systems, it calls search engines for fetching possible relevant Web pages.

Diao et al [DLCT00] improves the precision by filtering out non-relevant documents in future searches. At the early stage, the system returns the results obtained by a search engine to the user. However, it records the segment of the Web page that contains the query result and the sequence of a hyperlinks that the user navigates. The query results and the user actions are analyzed, and after the user browses a few pages, the system recognize the user's need. It scans future Web pages that are returned by the search engine. During the scanning, it follows links if necessary to locate the query results. In the end, the system is able to return the segment that contains the answer. This work also addresses the common problem of search engines which return URLs instead of answers. The system is able to return a rank of possible short passages. This way, the user do not need to browse the whole Web pages. In that sense, there are similarities to WebQA that is described in this thesis.

2.8 Question-Answering Approach

Question-answering (QA) systems allow users to pose natural language questions and return the natural language answers. The systems retrieve relevant documents and extract answers from these documents. Traditional QA systems use off-line

corpuses to retrieve relevant documents. Since, this thesis investigates the Web searching problem, we only focus on QA systems that use the open source (WWW). We call such systems *online question-answering systems*. Since they use the Web as data sources, the systems are not restricted to a limited set of knowledge. An advantage of using the QA approach is that the users are not required to study the complex syntax of input query, such as SQL. Therefore, novice users can perform Web searching effectively. The drawback is that natural language introduces language ambiguity and is not well-structured.

2.8.1 QASM

Radev et al [RQZ+01] propose a Web mining approach to question-answering. Given a natural language question N, the system uses its probabilistic algorithm QASM to recover a original query from the set of all possible queries that can be generated from N using a limited sequence of linguistically-justified transformation operators. Then the system uses the generated query to submit it to search engines. After that, the system uses another algorithm AnSel for answer extraction. The goal of the QASM algorithm is to find the best paraphrase for the given natural language question.

2.8.2 Ask Jeeves

Ask Jeeves [Ask02] is an online natural language information retrieval system. The users pose plain English questions, the system returns links to relevant Web pages. It simulates an online interview. The system combines natural language process-

ing (NLP), human editorial judgement, and information retrieval techniques. Ask Jeeves employs editorial staff to predefine relevant Web pages for frequently asked questions. Besides human editorial process, the system uses the Teoma [Teo02] search engine for generating automatic results. In addition to answering factual questions, it is also capable of answering procedural questions such as *"How do I write a business plan?"*. The drawback of the system is that it does not return direct answers, but URLs that the user has to subsequently search. In other words, it is like a search engine, except that it allows users to input natural language questions.

Ask Jeeves operation is described briefly in [BYRN99]. When a user poses a natural language question, the system matches the user query against the question types. For example, a question *"Who is the prime minister of Canada?"* is mapped to *"Who is the head of country of X (Canada)"*. The variable X is bound to *"Canada"*. As mentioned earlier, Ask Jeeves stores a list of Web pages for predefined questions. The system searches the question type against the set of predefined questions and returns the matched Web pages. In addition, it sends the user query to Teoma search engine to retrieve other relevant Web pages. [BYRN99] provides the following comment on Ask Jeeves:

> ... a system is only as good as its question templates. For example a question *"Where can I find reviews of spas in Calistoga?"* matches the question *"Where can I find X (reviews) of activities for children aged Y (1)?"* and *"Where can I find a concise encyclopedia article on X (hot springs)"*

2.8.3 Mulder

Mulder [KEW01] takes the natural language approach. It allows a user to pose a natural language question. More specifically, it is an online question-answering system like WebQA. It aims to solve short factual questions and returns exact answers. It uses the common components of a traditional QA system: Question Parsing, Question Classifier, Query Formulation and Answer Extraction. First, a user states a short factual natural language question. The Question Parsing component converts the NL question to a syntactic structure. It uses one of the best NL parsers, namely Maximum-Entropy-Inspired (MEI), for performing this component. The Question Classifier takes the syntactic structure and identifies the question type. The values of question type are nominal, numerical, and temporal. Nominal type is for the answer that is written in a noun phrase. Numerical type is for the answer that is written as a number. Temporal type is for the answer that is written in date format. The question (or answer) type is used later to extract answers. The Query Formulation component converts the syntactic structure to a set of keywords for a search engine. Later on, the Answer Extraction component extracts the answer from the search engine result using its extraction algorithm.

There are many differences between WebQA and Mulder. Mulder heavily relies on natural language processing and that is not the research focus of WebQA. Mulder uses 3 question categories and whereas WebQA uses 7. We think that more categories provide more specificity in extracting answers. Mulder uses a single search engine for retrieving online documents (in a sense, it is adaptation of Google for question-answering systems). On the other hand, WebQA has the abil-

ity to query multiple sources (search engines and on-line databases) for passage retrieval. These sources can be search engines, weather sites, and other Web sites that store facts. Furthermore, WebQA is more fault-tolerant, because if one source fails, there are other sources to support the passage retrieval. Finally, WebQA uses wrapper/mediator architecture, giving it further flexibility and scalability. More detailed discussion of the differences between the two systems is provided in Chapter 8.

Chapter 3

WebQA Architecture

This chapter covers the architecture of WebQA engine, its client/server implementation, and provides an overview of a query processing.

WebQA is implemented in Java. The compiler and interpreter is Java 2 SDK Standard Edition version 1. 3. 1. There are many reasons for choosing Java. A primary reason is that Java is object-oriented. Since the program is big and contains over ten thousand lines of code, it is much easier to develop using an object-oriented approach. Secondly, Java is portable. We can develop WebQA in one platform and deploy it on another platforms. Furthermore, Java provides us strong libraries, which allow us to develop WebQA easier. It has libraries that manipulate a list and a string which are common data structures for WebQA. It provides a class called `Thread`, which allows WebQA to concurrently connect to multiple Web sources. It provides a class call `URL`, which only requires us to specify the URL in order to download a Web page. Finally, we can use JSP, Java Servlets, or Java Applet to easily develop Web applications, so users can access WebQA online. One may think

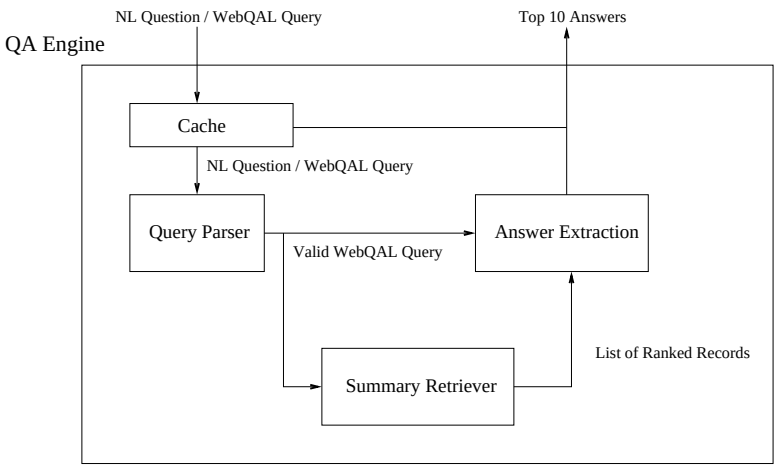

Figure 3.1: The architecture of WebQA's Engine

that Java is slow, however the execution time of WebQA is extremely fast. It takes less than half a second for locally processing a query excluding the waiting time from other web sources (see Chapter 7). WebQA's implementation involves over 120 Java classes.

3.1 QA Engine

WebQA engine takes a short factual question and returns a list of short answers. The corresponding Java class for the engine is called `QAEngine`. Its architecture consists of three main components: the Query Parser (QP),the Summary Retriever (SR), and the Answer Extractor (AE) (Figure 3.1). Each component consists of sub-components for further division. The component-based architecture facilitates incorporation of improvements into the system. In this way, the system is extensible and improvable.

Our current research provides a natural language interface but converts this to an internal language called WebQAL introduced in Section 4.3 that all system components understand. Of course, WebQAL can be used by users to pose queries if desired. The reason for introducing this language is to avoid English ambiguity.

The general steps of executing WebQA's engine are the following:

1. Given an input query, WebQA Engine passes it to the QP component (Chapter 4). The job of QP is to accept either a NL question or a WebQAL expression and return a valid WebQAL expression.

2. The engine checks whether or not the valid WebQAL expression is in its cache. If the query is found in the cache, then the answer list is retrieved from the cache and returned to the user, otherwise, processing continues.

3. The WebQAL expression is passed to the next main component: SR (Chapter 5). SR takes the WebQAL expression and produces a list of ranked records (ranking is done globally). A record (Section 5.3.1) consists of a text passage from a search engine, the data source name where the passage is come from, a local rank (rank within the source). The way of how SR gets the list of records is by asking different sources. Which source to ask is depended on source ranking algorithm.

4. AE (Chapter 6) can take the list of records and extracts a list of answers produced from SR. AE's job is to take a list of ranked records and a WebQAL expression, and to extract a list of answers as output. Different extraction techniques can be applied.

5. The engine returns the list of answers to the specified user interface.

3.1.1 Inputs and Outputs

When we instantiate an WebQA engine object in Java, there is information that we need to provide to configure it. First parameter is the debugger. If the debugger is set to be on, then system outputs debugging information to the user when the user runs WebQA using a textual interface (Section 3.4.1). Such debugging information are the execution time of each component and sub-components. The second parameter is the cache manager. The cache manager (Section 3.2) is used to increase the system performance. It stores a list of recent queries and the answers to these queries. The third parameter is the number of most relevant answers to return to the users. The fourth parameter is the time-out limit in terms of milliseconds. The user can specify how long WebQA should wait for the results from Web sources. Once the time limit is passed, WebQA gets the current result or terminates the connection to the Web sources. If the time limit is set to -1, then there is no time limit specified. The fifth parameter is the number of concurrent pages. In other words, the number specifies how many pages to download from a Web source concurrently. A typical search engine might return thousand or more pages, and we do not want to download them all. On the other hand, we don't want to download each page sequentially.

Once an engine object is instantiated, we can call `QAEngine`'s `getResult` method for returning a list of answers of a given query. The given query is written as a string. It is either a NL question or a WebQAL expression. If the query is a WebQAL

expression, the query has prefix "`webqal:`". If the query is a NL question, the query has prefix "`nl:`". If there is no prefix, then the query is treated as a NL question. The return type of the `getResult` method is a class called `Result`. The `Result` class has three attributes: an answer list, an execution time, and a boolean that specifies that the answer comes from a cache or from the sources. The answer list is an array of strings that is generated from the Answer Extractor component. The engine object sets the three attributes and returns the `Result` object.

3.2 Cache Management

WebQA takes a matter of seconds to process a query (see Section 7.4.3). Moreover, a simple cache is implemented to increase its efficiency. Depending on user query patterns, the cache can eliminate some retrievals from Web sources. A cache manager is implemented to provide accesses to its cache. The cache is implemented using Java's `HashMap` class. The entry in `HashMap` contains a key and a value. The class provides operations for adding an entry given a key and a value, removing an entry given a key, and returning a value given a key. It provides constant-time performance for those operations. We interpret the key of an entry in `HashMap` as a WebQAL expression. We interpret the value of an entry in `HashMap` as a object of `CacheEntryValue`. An object of `CacheEntryValue` has two attributes: an answer list and a time stamp. The answer list is the answer list returned by WebQA's engine and the time stamp is the time of the last access. The cache manager supports two methods: `read` and `write` for reading and writing a cache entry, respectively. The `read` method takes a WebQAL expression and returns the

corresponding answer list. The `write` method takes a WebQAL expression and an answer list. It instantiates an `CacheEntryValue` object with the specified answer list as the object's answer list attribute and the current time as the object's time stamp attribute. Then, it finds the oldest cache entry in the cache and replaces the oldest cache entry with the WebQAL expression as a key and with the `CacheEntryValue` object as a value. The cache's replacement algorithm is LRU (Least-Recent-Used) Approximation Algorithm [Gal98] (i.e., the `write` method). The cache management provides read and write locks for concurrent access to the cache. Algorithm 1 and 2 shows the pseudo code for implementing the `read` and `write` methods, respectively.

Algorithm 1 The pseudo code for Cache Manager's `read` method

Require: *webqal*, *cache* (implemented using Java's HashMap)
Ensure: *answerList* is the corresponding answer list of *webqal*
  Obtain a read lock on *cache*
  $value \Leftarrow cache$.get($webqal$)
  if $value \neq$ null **then**
    $answerList \Leftarrow value.answerList$
    $value.timeStamp \Leftarrow$ current time
  else
    $answerList \Leftarrow null$
  end if
  Release the read lock

3.3 Client/Server Architecture

So far, the internal operation of the WebQA engine has been described. This section covers how a client/server architecture can be set up to interact with remote clients through a Web browser. The architecture is shown in Figure 3.2. Since a client accesses WebQA through a Web page, a Web server is necessary to handle the

Algorithm 2 The pseudo code for Cache Manager's `write` method

Require: $webqal$, $answerList$, $cache$ (implemented using Java's HashMap)
Ensure: Put the new cache entry into $cache$
  Instantiate a $newValue$ object
  $newValue.answerList \Leftarrow answerList$
  $newValue.timeStamp \Leftarrow$ current time
  Obtain a write lock on $cache$
  if maximum size of $cache < 1$ **then**
    Do nothing
  else if current size of $cache <$ maximum size of $cache$ **then**
    $cache$.put($webqal, newValue$)
  else
    $oldestKey \Leftarrow$ null
    $oldestValue \Leftarrow$ null
    for each key $thisKey$ in $cache$ **do**
      $thisValue \Leftarrow cache$.get($thisKey$)
      if $oldestValue =$ null or $thisValue.timeStamp < oldestValue.timeStamp$ **then**
        $oldestKey \Leftarrow thisKey$
        $oldestValue \Leftarrow thisValue$
      end if
    end for
    $cache$.remove($oldestKey$)
    $cache$.put($webqal, newValue$)
  end if
  Release the write lock

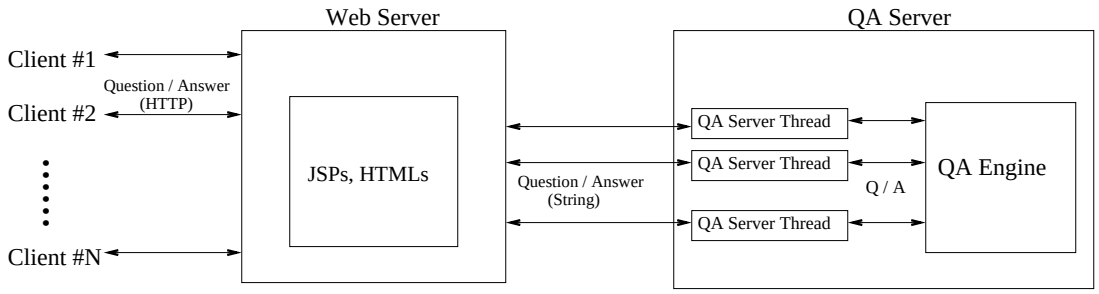

Figure 3.2: The client/server architecture of WebQA

HTTP protocol. The Web server that WebQA uses is Jakarta Tomcat 3.3, which enables Java Server Pages (JSP). Our Web user interface is written in JSP, which provides the ability to use Java to connect to the QA Server.

Once a Web server is started, it is able to accept any client requests (HTTP protocol). When a request comes, the Web server sends the request as string to the QA server using Java's Socket. The job of the QA server is to accept any request from the Web server and return the answer back to the corresponding Web server. When a request comes from the Web server to the QA server, the QA server immediately generates a thread that handles this request. Again, all threads in the QA server are running concurrently from each other. Each QA server thread asks WebQA's engine to get the answer of the given question. Finally the thread returns the answer back to the original Web server, which returns the answer back to the client as a new HTML page.

3.4 Running WebQA

There are three ways to execute WebQA's engine: using the evaluation classes that will be discussed in Chapter 7, locally using a textual interface, and through a Web browser.

3.4.1 Textual Interface

The textual interface provides a local access to WebQA. The advantage of using the textual interface is that it is fast and it provides debugging information. This information includes the execution time of each component and subcomponent. `TextUI` is a Java class that implements for the textual interface. There are 6 options for the class as listed in Table 3.1. In order to access the textual interface, the users need a copy of WebQA in their local machine which is not convenient. The main use of the textual interface is for developers to test the system.

Table 3.1: The options for the textual interface

Options	**Description**	**Default Value**
-debug on/off	Turn the debugger on/off	off
-cache n	n is the cache size	0
-ansi n	n is the number of returned answers	10
-timeliest n	n is the time limit for accessing Web sources in ms	no time limit
-record $file$	$file$ is the record file	$null$
-help	Output all the options	N/A

Depending on the Java IDE or the interpreter, access to the `TextUI` class is different. Assuming that we are using Sun Microsystems' Java 2 SDK Standard Edition Version 1.3.1, here is the command for accessing the class:

`java -cap` ⟨*class path*⟩ `ca.uwaterloo.db.webqa.evaluation.TextUI` ⟨*options*⟩

⟨*class path*⟩ indicates the directory where the Java class files are located. ⟨*options*⟩ is a list options that are shown in Table 3.1.

After we enter the command, the system asks the user to enter a query. The system will return the answer in lesser few seconds. Figure 3.3 shows a sample screen of a query *"Who invented the telephone?"* with the debugger turned on.

3.4.2 Graphical User Interface

The graphical user interface (GUI) of WebQA is a Web site that is written in JSP and HTML. The home page is shown in Figure 3.4. Section 3.3 describes how WebQA interacts with clients remotely through a Web browser. In order to allow users to use the Web page, we need to start up the Web server and the QA server. Depending on the Web server, there are different ways to start it up. The Web server that we use is Jakarta Tomcat 3.3. All our JSP and HTML pages for creating the Web site are stored in the following directory:

⟨*Web Server Directory*⟩`/webapps/webqa/`

JSP allows us to write Java code and HTML together. Inside the JSP pages, we write Java code in order to make a connection to the QA server. The compiled Java's class files (byte code) for the Java code in the JSP pages is stored in the following directory:

⟨*Web Server Directory*⟩`/webapps/webqa/WEB-INF/lib`

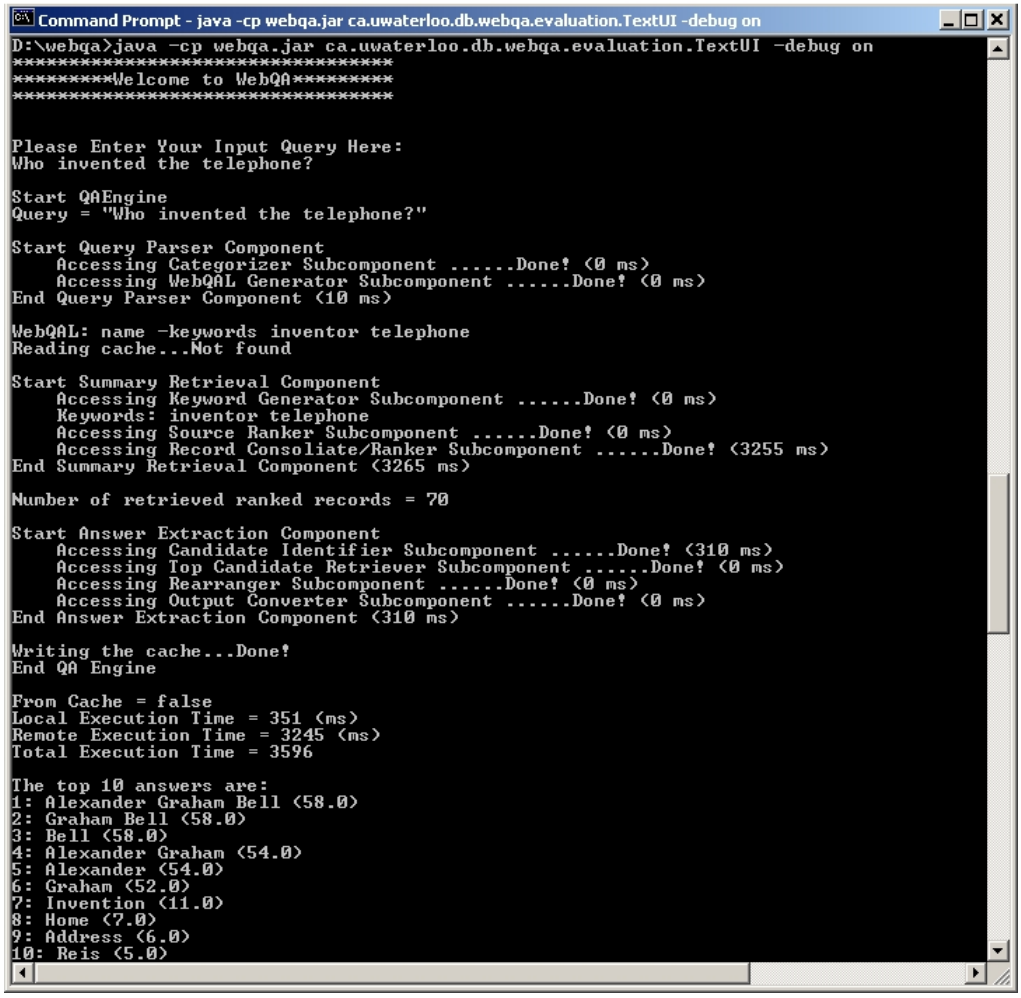

Figure 3.3: A sample screen of the textual interface of WebQA

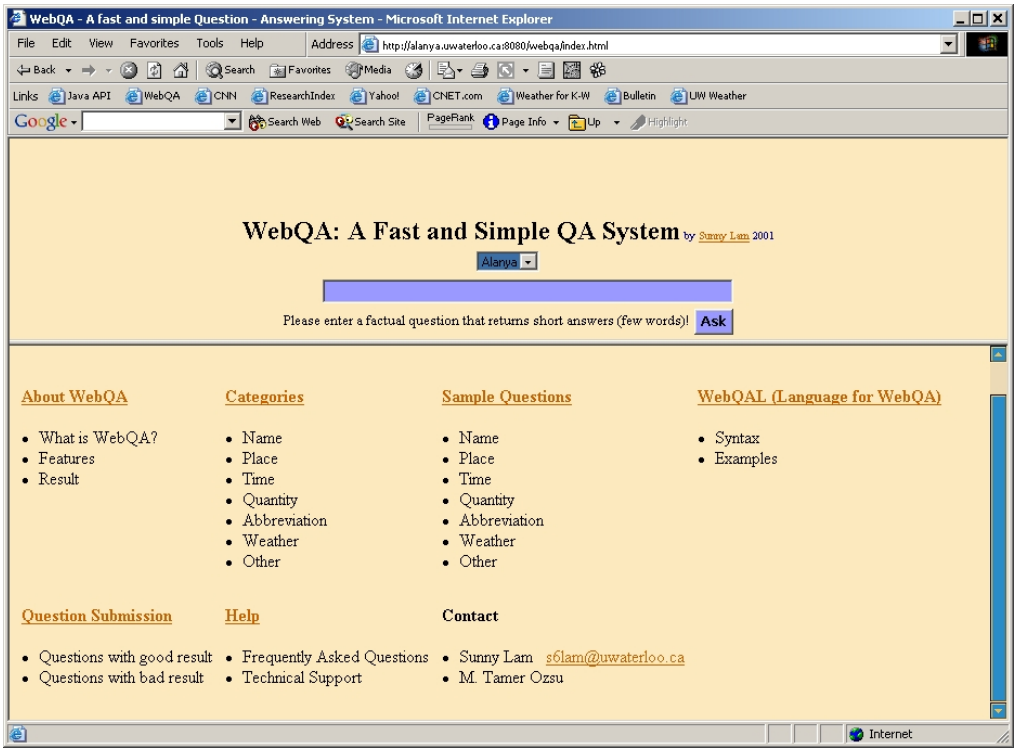

Figure 3.4: The WebQA's home page

To start up the Web server, we can use the following command:

⟨*Web Server Directory*⟩`/bin/startup.bat`

On the other hand, when we want to stop the Web server running, we can use the following command:

⟨*Web Server Directory*⟩`/bin/shutdown.bat`

Once we start up the Web server, we can start up the QA server. There are 5 options for the QA server as listed in Table 3.2. Notice that if we turn the debugger on, the debug information is only shown in the QA server, not the Web page. The QA server is implemented as a Java class called `QAServer`. The following command is to start up the QA server.

`java -cp` ⟨*class path*⟩ `ca.uwaterloo.db.webqa.server.QAServer` ⟨*options*⟩

⟨*class path*⟩ indicates the directory where the Java class files are located. ⟨*options*⟩ is a list options that are shown in Table 3.2.

Table 3.2: The options of the QA server

Options	**Description**	**Default Value**
-debug on/off	Turn the debugger on/off	off
-cache n	n is the cache size	0
-port n	n is the port number	18081
-timeliest n	n is the time limit for accessing Web sources in ms	no time limit
-help	Output all the options	N/A

After we started up both servers, then a user can type the following URL for accessing WebQA.

`http://db.uwaterloo.ca:18080/webqa/`

The user just simply enter a English question or a WebQAL expression in the text box. A sample Web page is shown in Figure 3.5 for the question *"Who is the prime minister of Canada?"*.

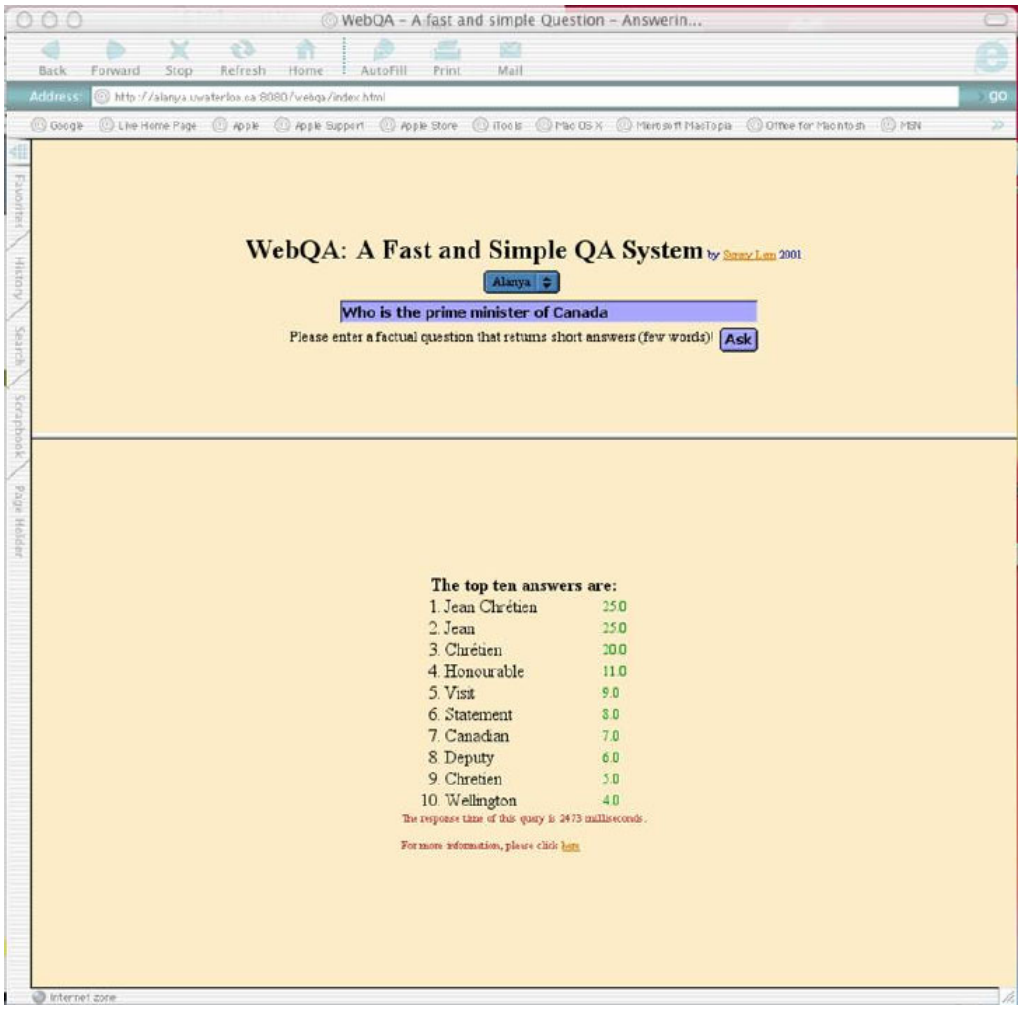

Figure 3.5: A sample answer page of WebQA

Chapter 4

Query Parser

The Query Parser (QP) performs one of two functions depending on its input. If the user has specified a natural language query, then QP "*understands*" the query, categorizes it, and translates it into WebQAL. If the user query is specified in WebQAL, then QP functionality is restricted to analysis and verification of the correctness of the WebQAL expression. The architecture of the QP is shown in Figure 4.1. If the given query is in natural language (NL), it passes the query to the Categorizer subcomponent. An example of a NL question is *"Who invented the telephone?"*. The Categorizer subcomponent determines the category. In this case, our category is *Name*. The category and the query are passed to the WebQAL Generator subcomponent. The WebQAL Generator generates a WebQAL (*name -keywords inventor telephone*) and QP returns the WebQAL query. On the other hand, if the query is a WebQAL query, QP passes it to the WebQAL Checker subcomponent. The subcomponent checks the validity of the given WebQAL. If the query is valid, then QP returns the WebQAL. Else, it returns null. These

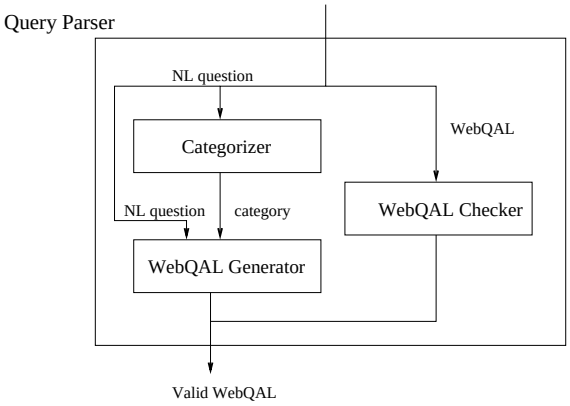

Figure 4.1: The architecture of Query Parser Module

functions are discussed in the following sections.

We should note that many QA systems use a full-fledged NL understanding system. The focus of these projects is *understanding* the user question and *extracting* the right answer, so the use of a heavy duty NL processor is warranted. In our case, the NL interface is not the research focus; in fact, it may be replaced by a more structured Web query language. Therefore, we have implemented a lightweight NL translator. This does not preclude the possibility of retrofitting a full NL understanding system as a front end to WebQA. As long as the front end performs categorization (most NL understanding systems can), it can be used with WebQA.

4.1 Stopwords

Stopwords are words that have little meaning or no meaning by themselves, appear quite frequently in text documents (such as *the, of, and,* and *a*). Information retrieval systems use lists of stopwords that vary from 15 words to over 500 words

[Kor97]. There is a reason to eliminate stopwords. Many extraction techniques use word frequency. If these stopwords have no relevance to the documents or the input questions, then it is necessary to eliminate or ignore them.

However, sometimes stopwords can be useful. For example, a user asks "the author of a book 'Eaters of the Dead' ", it is necessary to include the stopwords in the book title. Otherwise, some IR systems might return the death date of an author. Another example is asking for information on a well known phrase in English literature: "To be, or not to be" [Kor97]. If we eliminate all the stopwords, then there is nothing to search for. The solution is to add quotes to a title, phrase, or name where stopwords are important. A full list of stopwords that WebQA uses can be found in Appendix B.

4.2 Categorizer

The function of the Categorizer subcomponent is to determine the category of the given natural language question. The question is received from the QP component. And the category is returned to the next subcomponent, WebQAL Generator.

4.2.1 What are the categories?

User queries are categorized into several classes to improve the accuracy of the system. The category information is used by all modules of the systems. Currently, there are seven categories: *Name, Place, Time, Quantity, Abbreviation, Weather,* and *Other*; with the possibility of adding more categories in the future. Each category uniquely categorizes the output format. *Name* represents any names, such

as the name of persons, companies, and universities. *Place* represents any cities, divisions (i.e., states, provinces, territories, or districts), countries, and continents. *Time* represents date. *Quantity* represents numbers with or without units. *Abbreviation* represents acronyms. *Weather* represents information of the weather, such as conditions, temperatures, barometers, wind speeds, dew points, humidities, visibility, sun rise time, sunset time, moon rise time, and moon set time. *Other* category handles answers that cannot be categorized into any of the above categories. The reason for introducing categories is to improve system accuracy. More categories such as telephone number and stock quote can be added in future work.

4.2.2 How to categorize a natural language question?

The job of the Categorizer is to find out which of the seven categories the NL question belongs to. The categorizer scans each word of the question from left to right. If it finds a word that belongs to a specific category, then it stops and returns this category. In our categorization, one question belongs to exactly one category.

If the question starts with "Who" or "Whom", it is likely to ask for a name. For example *"Who invented the telephone?"*. However there are examples that the question may ask for something else, such as *"Who was George Washington?"*. The answer to this question should be "US president" or "first US president", not a name. The way to handle this problem is currently handled as follows. When a question starts with "Who" or "Whom", and after all stopwords are eliminated and one of the words does not start with a uppercase first character than the question belongs to *Name* category. For example, *"Who is the chairman of AOL Time*

Warner?" belongs to *Name* category and *"Who is Thomas Edison?"* does not belong to *Name* category (i.e., *Other* category).

If the given question starts with "Where" or "When", obviously, it belongs to *Place* category or *Time* category respectively.

Name questions can also be specified by "What" or "Which" keywords. For example, *"What company sells furniture?"* asks for a name. The categorizer checks questions that start with "What" or "Which" against a term list for each category. If the word is found in one of the term lists, then the processing stops and Categorizer returns this category. Of course, the term list for each category can be expanded in the future.

If the question starts with "How", it might be asking for *Quantity* category, such as "How many", "How much", "How far". The Categorizer checks the word following "How" against a list of words that can identify the question as belonging to *Quantity* category. The list consists of words such as "long", "short", "fast", "big", and "small".

Algorithm 3 shows how the Categorizer subcomponent categorizes a natural language question. The algorithm scans each word in the question. If the current word is "what" or "which", the algorithm initializes the $hasWhat$ variable to true. If the current word is "where", the algorithm returns "Place". If the current word is "when", the algorithm returns "Time". If the current word is "how" and the word followed by it is one of the word in *How-quantity* term list (Appendix C.7) , then the algorithm returns "Quantity". The *How-quantity* term list is a list of possible words after the word "how", which indicates the question is asking for

Algorithm 3 Categorization Algorithm

```
function categorize(String nlQuestion) : Category
boolean hasWhat ⇐ false
for each word X in nlQuestion do
  if X = "what" or "which" then
    hasWhat ⇐ true
  else if X = "where" then
    return "Place"
  else if X = "when" then
    return "Time"
  else if X = "how" then
    if the word after X is in howQuantityTermList then
      return "Quantity"
    end if
  else if X = "who" or "whom" then
    newQuestion ⇐ nlQuestion without stopwords
    if every word in newQuestion starts with a upper letter then
      return "Other"
    else
      return "Name"
    end if
  else if X is in nameTermList then
    if hasWhat then
      return "Name"
    end if
  else if X is in placeTermList then
    if hasWhat then
      return "Place"
    end if
  else if X is in timeTermList then
    if hasWhat then
      return "Time"
    end if
  else if X is in quantityTermList then
    if hasWhat then
      return "Quantity"
    end if
  else if X is in abbreviationTermList then
    if hasWhat then
      return "Abbreviation"
    end if
  else if X is in weatherTermList then
    if hasWhat then
      return "Weather"
    end if
  else if X is in otherTermList then
    if hasWhat then
      return "Other"
    end if
  end if
end for
return "Other"
```

the *Quantity* category. For example they are "how old", "how long", and "how many". The examples ask for the *Quantity* category. If the current word is "who" or "whom", then the algorithm removes all the stopwords in the given natural language question. If all words in the new question starts with a upper case letter, then the algorithm returns "Other", because the original question is asking for a person's background (e.g., *"Who is Jean Chrétien?"*). Otherwise, the algorithm returns "Name" (e.g., *"Who is the prime minister of Canada?"*). After that, the algorithm checks the current word is in one of the term lists (Appendix C.1 to C.6). Again, a term list is a list of possible words that can identify a category. If so, the algorithm returns the category that associates the term list. If the algorithm scans all the words in the question and cannot categorize the question, then the algorithm returns the "Other" category.

We have tested the effectiveness of categorization as well as its performance. The categorization time is proportional to question length, because of sequential scanning of the query and comparison with a constant length term list. Section 7.4.1 shows an experiment that test the performance of the Categorizer. The experiment uses a list of TREC-9 questions. There questions are first manually categorized against which the results of automatic categorization is compared.

4.3 WebQAL Generator

The WebQAL Generator subcomponent takes a short natural language and the category, and generates a WebQAL accordingly.

4.3.1 What is WebQAL?

WebQAL, which stands for *WebQA Language*, is the standard internal query language for the WebQA system. It is also available to users for query specification, but its main use is as a common internal language used by WebQA components. Its syntax, given below, is more specific than natural language, but more cryptic. It specifies the category, the output type, and the suitable keywords. Thus, it is better suited as an internal language into which the natural language query is translated. The formal specification of the language is given below. All the output options for each category are specified in Table 4.1.

⟨*Category*⟩ `[-output` ⟨*Output Option*⟩ `] -keywords` ⟨*Keyword List*⟩

4.3.2 How to generate a WebQAL query?

Given a category and a NL question, the function of WebQAL Generator is to produce the associated WebQAL expression. This is done by using the following stopword elimination, (see Appendix B for the stoplist that we use). Algorithm 4 shows how to eliminate the stopwords on a given question. During conversion, verb-to-noun translation is used whereby each word in the question is scanned and if the scanned word is among these in Appendix D, then it is converted to the appropriate noun. The reason for this is to increase the accuracy of the search. For example, if we have a question *'Who wrote the novel "Jurassic Park"?'*, it is more effective if we submit *'author "Jurassic Park"'* than *'wrote "Jurassic Park"'*. We can add more elements in Table D in the future.

Algorithm 4 The stopword elimination algorithm

Require: $nlQuestion$ (the natural language question)
Ensure: $newQuestion$ (a question without stopwords)
  $newQuestion \Leftarrow \epsilon$
  $isQuoted \Leftarrow$ false
  for each word X in $nlQuestion$ **do**
    if X begins with a double quote **then**
      $isQuoted \Leftarrow$ true
    end if
    if $isQuoted$ **then**
      $newQuestion \Leftarrow newQuestion + X$
    else
      if X is not in the stop list **then**
        $newQuestion \Leftarrow newQuestion + X$
      end if
    end if
    if X ends with a double quote **then**
      $isQuoted \Leftarrow$ false
    end if
  end for

All the categories except *Abbreviation* and *Weather* have to perform both stopword elimination and verb-to-noun conversion. All the categories use the remaining words as the value of WebQAL keyword parameter. The remaining issue is how to set up the output parameter. Since *Name* and *Other* categories do not have output parameters, the following shows how the stopword elimination and verb-to-noun conversion works through an example. Given the question, *"Who invented the paper clip?"*, the generator first eliminates the stopwords which results *"invented paper clip"*. Then the generator performs verb-to-noun conversion, which results *"inventor paper clip"*. Finally, the generator will plug these into the keyword parameter list and get the following WebQAL expression:

```
name -keywords inventor paper clip
```

To identify the output parameter for the *Place* category, the system checks

the output parameter by searching words of the query such as *"what country"* or *"which country"* for output parameter *"country"*, *"what state"* or *"which state"* for *"state"*, *"what continent"* or *"which continent"* for *"continent"*, and *"what city"* or *"which city"* for *"city"*, etc. For everything else, the generator assigns *"unknown"* to output parameter. For example, *"Which country has the most population in the world?"* is converted to the following.

```
place -output country -keywords most population world.
```

Identification of the output parameter (i.e., unit) for *Quantity* category is more difficult, but the idea is the same. The system checks which words follow "how", "how many", and "how much". A table is constructed to tell what specific words have what units associate with them.

There are only two output options for the *Abbreviation* category. They are *short* (looking for abbreviation of the given words) and *long* (looking for the full spelling of the given abbreviation). If the question contains keywords, "how" combined with one of "abbreviate", "abbreviates", or "abbreviated", or "what" together with "abbreviation for", then the output option is *short*. If the question contains keywords: "stand(s) for", "abbreviation for what", or "what" and "acronym", then the output option is *long*. For example, the question, *"What does CNN stand for"* contains keyword "stand for", so the output option is *long*. The value of the keyword parameter is determined by the stopword elimination without verb-to-noun conversion. In addition, the words that were used to identify the output parameter are eliminated. Thus, the WebQAL expression of the question above is:

```
abbreviation -output long -keywords CNN
```

For *Weather* category, there are twelve output options, as mentioned in Table 4.1. The generator uses the most straightforward way to identify the output parameter. If the question contains the word, "weather", then the output option is *all*. If the question contains "conditions", then the output option is *conditions*.

For generating the keyword parameter, the generator eliminates all the stopwords and the words used for identifying the output parameter, without verb-to-noun conversion. Currently, the system only handles the current weather information. For example, the question, *"What is the current wind speed in Toronto, Canada"* is converted to:

```
weather -output wind -keywords Toronto Canada
```

4.4 WebQAL Checker

Like all other subcomponents in QP, WebQAL Checker is category-dependent. Its function is to verify that the given WebQAL is syntactically valid. First, it needs to make sure that the first word is one of the seven categories mentioned in Section 4.2. Second, it verifies that there are two parameters (output and keyword) for each category, except *Name* and *Other*, which have only the keyword parameter. Third, it confirms that the value of the output parameter is allowed. If all the above are true, then WebQAL Checker returns a true, else it returns a false. This component is built based on the syntax of WebQAL.

Table 4.1: Output options of each category.

Category	**Output Option**
Name	N/A
Place	city division country continent unknown
Time	dd mm yyyy dd/mm mm/dd dd/mm/yyyy mm/dd/yyyy
Quantity	*any measurement unit*
Abbreviation	short long
Weather	all conditions temperature barometer wind dewpoint humidity visibility sunrise sunset moonrise moonset
Other	N/A

Chapter 5

Summary Retriever

The job of Summary Retriever (SR) component is to accept a WebQAL as an input and to return a list of ranked records as output. Figure 5.1 shows the detailed SR module architecture. The processing is as follows. First, SR passes the given WebQAL expression to its subcomponents Keyword Generator and Source Ranker. Keyword Generator accepts the WebQAL expression and generates a list of keywords for search engines or other Web sources to search. Source Ranker accepts the same WebQAL expression and produces a list of ranked sources for searching. Finally, SR passes the list of keywords and the list of ranked sources to Record Consolidator/Ranker. The Record Consolidator/Ranker submits the search to the sources included in the source list, consolidates the returned records, and ranks them according to its ranking algorithm. The output of Record Consolidator/Ranker, which is the output of SR, is a list of ranked records.

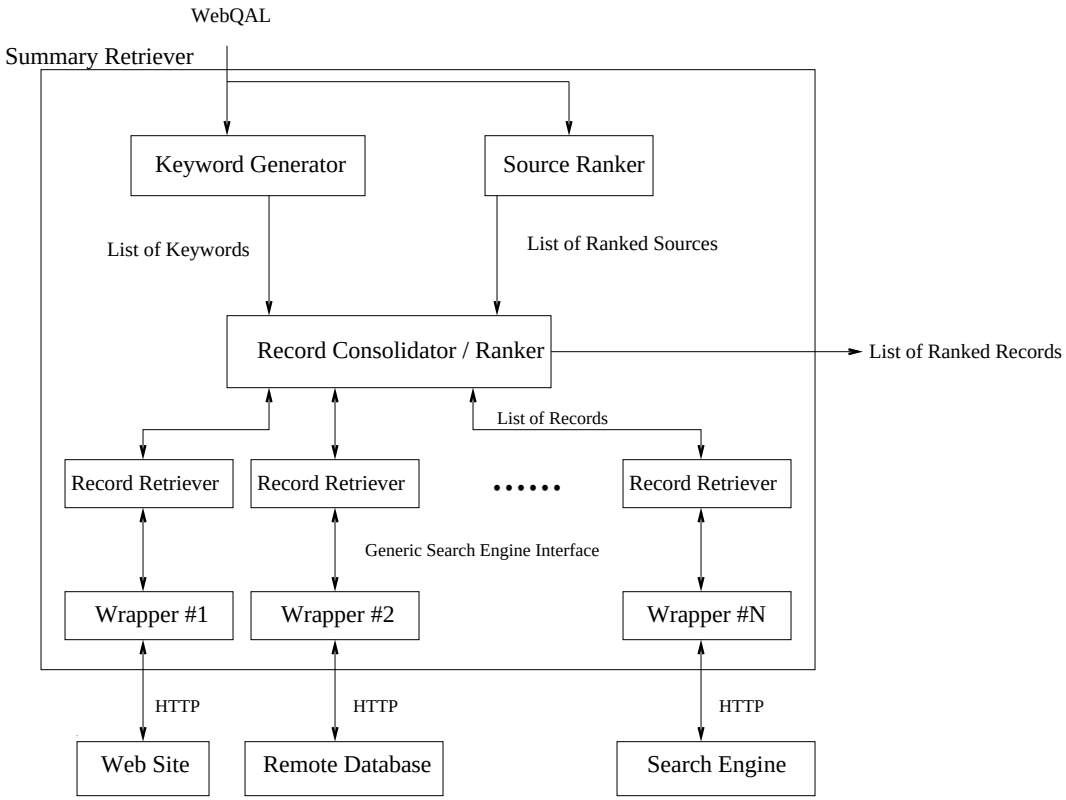

Figure 5.1: The architecture of Summary Retriever

5.1 Keyword Generator

The Keyword Generator subcomponent generates keywords for search engines to search. The list of keywords is extracted from the keyword parameter of the given WebQAL. However, *Abbreviation* category adds an extra word *abbreviation* at the end, so that the search engines know it is looking for abbreviation. For example, if the WebQAL expression is the following:

```
name -keywords inventor telephone
```

The subcomponent returns the keyword expression–*"inventor telephone"* for submitting to Web sources. In another example, we might have the following WebQAL expression:

```
abbreviation -output long -keywords OGS
```

In this case, the subcomponent returns the keyword expression, *"OGS abbreviation"*.

5.2 Source Ranker

Source Ranker takes a WebQAL expression and produces a ranked list of preferred sources. A source can be a search engine (such as Google), a Web site that contains facts (such as CIA's World Factbook or a Weather site), or an external database. At this stage, the source ranker is static with no learning component. It produces a different list for different query categories. We determined the sources that are most suitable for each query category by running a number of experiments based on the TREC-9 questions (see Chapter 7). Table 7.2 shows the sources that WebQA

uses for each category. In the future, we will incorporate learning algorithms to enable the system to adapt dynamically to changes.

5.3 Record Consolidator/Ranker

We can divide the Record Consolidator/Ranker subcomponent into two parts: Record Consolidator and Record Ranker. As suggested by its name, Record Consolidator retrieves records from the sources and consolidates them. Record Ranker ranks these retrieved records. Before we discuss the two parts, we need to understand what is a record and its structure.

5.3.1 Record

A record is a piece of information that might contains the exact answer. It is used in the Answer Extractor (AE) component for extracting answers. A record consists of three attributes: a source name, a snippet, and a local rank. The source name is the name of the source where the information is come from. The snippet is a piece of text that might contain the answer. Different wrappers have different snippet structure, which will be described in Section 5.4. The local rank is the rank of the record within the source. The structure of the record is shown in Table 5.1. These three pieces of information are used in AE for extracting answers.

Table 5.1: The structure of a record

Attribute	Data Type
Source Name	String
Snippet	String
Local Rank	Integer

5.3.2 Record Consolidator

Record Consolidator takes the keyword expression from the Keyword Generator subcomponent and the source ranking from the Source Ranker subcomponent. For each source, Record Consolidator instantiates a record retriever. Each record retriever requires two inputs. The first input is the keyword expression and the second input is the wrapper (Section 5.4) for the record retriever's assigned source. According to the source name, Record Consolidator creates an appropriate wrapper and passes the wrapper to the record retriever. A record retriever retrieves all the records from the wrapper and returns them to Record Consolidator. All record retrievers are implemented as threads, so they can run concurrently to save time. There are time limits for all the record retrievers, in order to produce answers in bounded time. When the system times out, Record Consolidator interrupts all active record retrievers and gets the records that have been retrieved so far. Algorithm 5 gives the pseudo code for Record Consolidator.

5.3.3 Record Ranker

Record Ranker is really dependent upon the extraction techniques which are described in Chapter 6. At this stage, our extraction technique does not worry about

Algorithm 5 The pseudo code for Record Consolidator

Require: *keywordExpression*, *sourceRanking*
Ensure: *recordList* is a list of consolidated records
  for each source s in *sourceRanking* **do**
    wrapper $\Leftarrow$ the wrapper that is associated to s
    Instantiate an record retriever which takes *keywordExpression* and *wrapper* as inputs
    Start the record retriever
  end for
  while time is not out and at least one record retriever is still active **do**
    Nothing
  end while
  stop all the active record retrievers
  recordList $\Leftarrow$ combine all the records from each record retriever

ranking the records, so a simple record ranking algorithm is applied. According to the source ranking list, the record ranker ranks all the records for the best source first, and same thing for other records of other sources.

Sometimes, we might retrieve more records than what we need, for backup uses. In other words, if one search engine dies, we still have records to extract from other sources. If we have a list of records of our first choice sources, then we can simply ignore the second choice sources. The way to proceed is to set up a number to determine how many records we want to retrieve from the given list of records.

5.4 Wrappers

A wrapper is a database technique that is used for information integration [Ozs99]. It provides a general view to callers where the data sources have different data structures, interfaces and computing capabilities. Different wrappers are assigned to different Web sources. Each wrapper knows how to communicate to its assigned source. On the other hand, all the wrappers must implement the same interface,

Wrapper API. This way, we only have to implement only one Record Retriever which can retrieve records by calling the methods that are provided in the interface. The interface is described in Section 5.4.1. There are three kinds of wrappers: wrappers for search engines, wrappers for country information, and wrappers for weather site.

5.4.1 Wrapper API

There are two methods (function calls) in the API for wrappers. They are `submit` and `next`. All wrappers have to implement this interface and all callers (i.e., Record Retriever threads) can only call these methods in order to interact with the sources. The `submit` method allows a wrapper to take a keyword expression and submit it to its source. Its definition is shown below:

```
function submit(KeywordExpression k) :  void
```

The wrapper fetches the relevant pages from the source and possibly extracts snippets and stores them into a record list. The return type of this method is *void*, so there is nothing to return from the wrapper. After, the `submit` method has been called, the `next` method returns the next available record. Its definition is shown below:

```
function next() :  Record
```

If there are no more records to return, the `next` method returns *null*. The implementation details of each wrapper is discussed in the following subsections.

5.4.2 Search Engine Wrappers

WebQA has a number of wrappers for various search engines: AllTheWeb.com, AltaVista, Excite, Google, Overture, Teoma, and Yahoo. All search engine wrappers are very similar, so their source codes can be reused for other search engine wrappers. The following describes the data structure, how to implement the two methods in the wrapper interface for a typical search engine wrapper, Yahoo.

What is the data structure?

The data structure for the search engine wrapper is a record list. We implement the record list as a queue. The record has been discussed in Section 5.3.1. The wrapper stores the record list, so it can return a record when the caller invokes the `next` method. Initially, the record list is an empty queue.

How to implement the `submit` method?

For implementing the `submit` method in a search engine wrapper, the wrapper first needs to determine the URL, which contains a host address followed by a list of keywords. The wrapper submits the URL to the search engine. For example, if the search engine is Yahoo and keyword expression is *"inventor telephone"*, then the result URL is the following:

```
http://google.yahoo.com/bin/query?p=inventor+telephone&b=1
&hc=0&hs=3&xargs=
```

As we can see, `inventor+telephone` represents the keyword expression. Furthermore, we need to add a plus sign (`+`) between each keyword. This is exactly like a

human submitting the keyword expression in the text box of Yahoo's home page.

After the search engine wrapper submits the URL, the search engine returns a result page that contains a list of summaries. Each summary contains meta data about a relevant Web page. A summary in a Yahoo result page contains a title, a snippet, and a URL as shown in Figure 5.2. The search engine wrapper extracts all the summaries in the result page by parsing HTML code of the page. We instantiate a record for each summary. As mentioned in Section 5.3.1, there are three attributes for a record: a source name, a local rank, and a snippet. The search engine wrapper sets the source name to its source. In this case, the Yahoo wrapper sets the source name of a record to *"Yahoo"*. From the source name, we can induce the source type (i.e., search engine). The search engine wrapper sets the local rank to the rank of the summary within the search engine. In addition, the wrapper combines the title of the summary and the snippet of the summary and sets the combined string to the snippet of the record. Each record is added in the search engine wrapper's record list in order of their local rank.

In order to save time, the search engine wrapper fetches a number of pages concurrently instead of fetching one page-at-a-time. On the other hand, we do not want to fetch all the relevant pages from a search engine. A typical keyword expression might have thousands of pages available. For example, the keyword expression, *"inventor telephone"* has over 50,000 available pages in Yahoo as shown in Figure fig:SampleSummary. Furthermore, it is highly likely that a search engine might return non-relevant summaries at the end. We use to fetch 5 pages concurrently to get a relevant result. Of course, WebQA allows the user to modify the number of

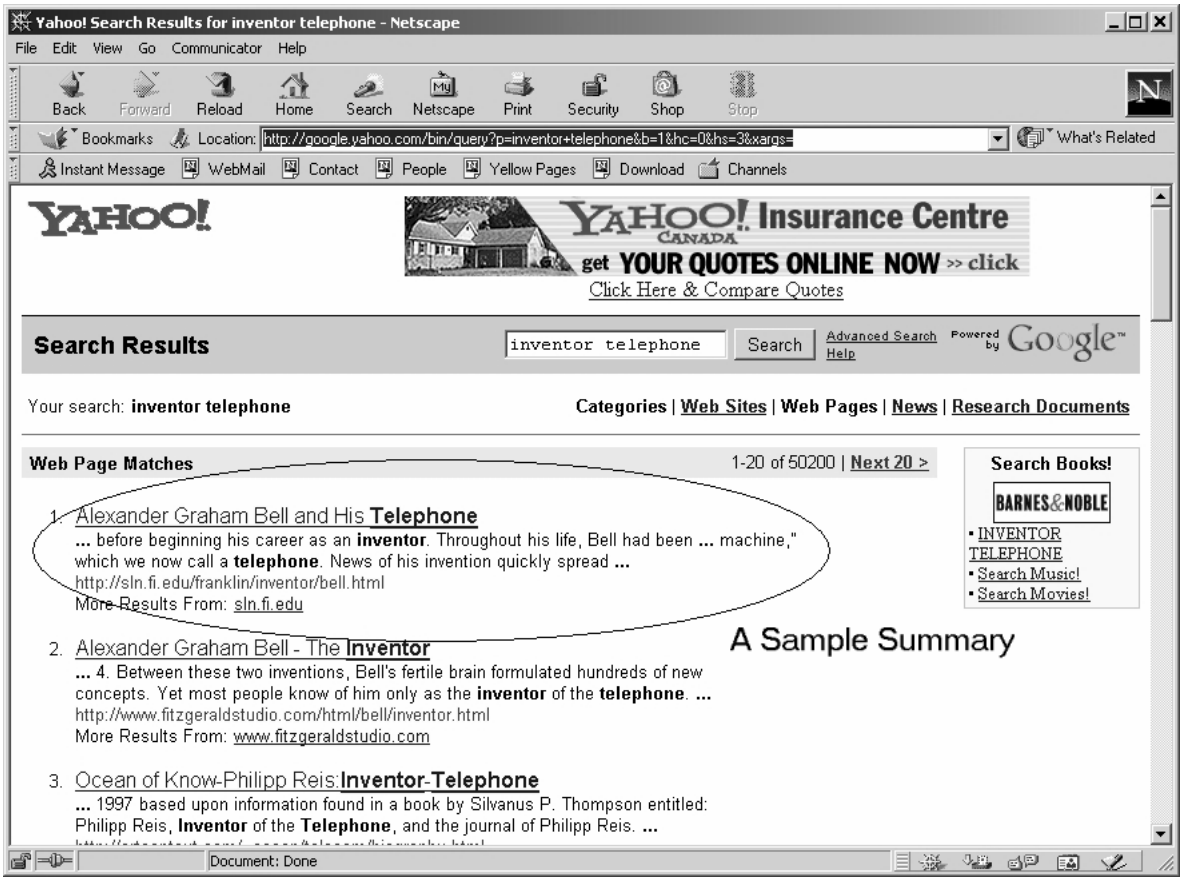

Figure 5.2: A sample summary from a Google's result page

pages to fetch concurrently in its parameter.

Since we can only submit a URL in order to fetch a page, we need to determine all the URLs for all the pages that we want to fetch concurrently. For each search engine, there is a parameter we can change in the URL in order to fetch a specific page. For Yahoo, we have written a sample URL above. The parameter `&b=`x means to fetch the page where the first summary is summary number x (starts from 1). We know there are 20 summaries for each Yahoo result page. Therefore, we have the following formula for finding x:

$$x = pageNum * 20 + 1 \tag{5.1}$$

where *pageNum* is the page number (starts from 0). All we need is to change the value of x for each page and we submit a set of URLs concurrently.

In general, the `submit` method of a search engine wrapper performs the following steps:

1. Determine a set of URLs
2. Submit the set of URLs concurrently and fetch a set of Web pages
3. Parse all the Web pages and extract the records
4. Add all the records into the record list

How to implement the `next` method?

The method `next` removes and returns the first record of the record list. If there are no records stored in the list, the `next` method calls the `submit` method for

retrieving the next 5 result pages. Of course, the `submit` method stores all the retrieved records in the record list. The `next` method removes and returns the first record of the record list. If the list is still empty, then the `next` method returns *null*. Algorithm 6 shows the pseudo code for the `method`.

Algorithm 6 The pseudo code for the search engine wrapper's `next` method

```
function next() : Record
if length of recordList > 0 then
   X ⇐ dequeue recordList
   return X
else
   submit(keywordExpression)
   if length of recordList > 0 then
      X ⇐ dequeue recordList
      return X
   else
      return null
   end if
end if
```

Web Connection to Google

All wrappers that are discussed in this section use the HTTP protocol to connect to their sources, except the Google wrapper. Since Google needs to protect their quality of search results and prevent their servers from being overloaded, they have denied access to automated queries [Goo02b]. Instead of accessing Google through the HTTP protocol, the Google wrapper access it by calling Google Web APIs [Goo02c]. This way, we can access Google, but with a different protocol. However, there is a limitation of 1000 queries per day, which is not *scalable*.

5.4.3 Country Wrappers

There is only one country wrapper implemented at this stage. The country wrapper is for World Factbook from CIA's Web site [WFB02]. It stores information for each country around the globe. Examples of information are population, geographic coordinates, total area, birth rate, and capital city. Figure 5.3 shows a sample Web page of World Factbook. The information of the Web site is very valuable and accurate. We can query this source if a user wants to ask information about a country. Therefore, it is worth it to implement a wrapper for the Web site. This section, we are going to discuss the data structure and how to implement the two methods in the wrapper interface.

What is the data structure?

The data structure for the World Factbook wrapper is only a record. Initially, the record is set to *null*.

How to implement the `submit` method?

As mentioned in Section 5.4.1, the `submit` method takes a keyword expression as an input. It tries to determine the URL, so the wrapper can fetch the relevant pages.

There is a URL associated with each country in the World Factbook. When the `submit` method is called, the World Factbook wrapper downloads a list of country names and the corresponding URL for each country from the source. Since we are dealing with short factual questions as described in Section 1.3, the user

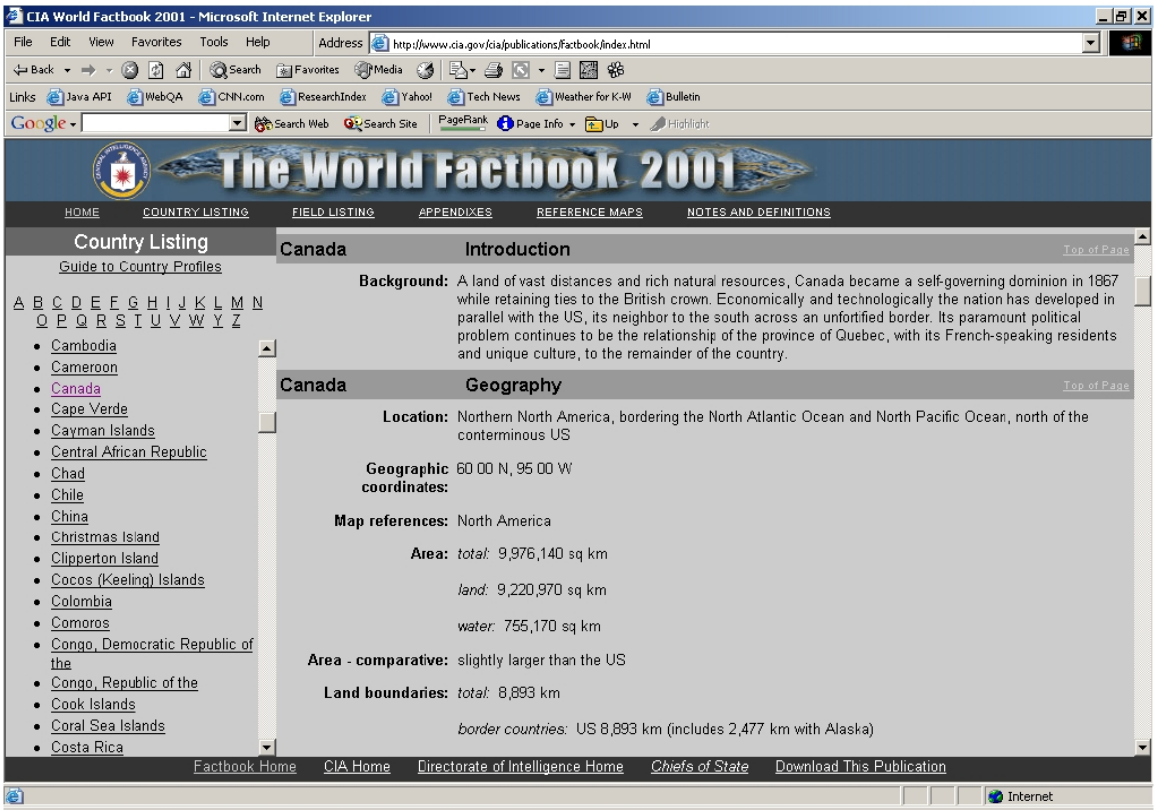

Figure 5.3: A sample Web page of CIA's World Factbook

can only ask information about only one country. Whenever the country name is a substring of the keyword expression, the wrapper the corresponding URL and submits it in order to download the country Web page. The country Web page contains information about the country such as population and the capital city.

The wrapper generates record. The record's source name is set to World Factbook. The record's local source rank is set to 1. The wrapper then parses the country Web page for setting up the record's snippet. First, it extracts the country name in the page by looking at the HTML title tag. For example, if the country is Canada, then the title tag is written as `<title>CIA -- The World Factbook -- Canada</title>`. The wrapper converts this to country tag and stores it in the record. In this case, we have country tag as `<country>Canada</country>`). Later on, the wrapper extracts all other information about the country. For example, the wrapper extracts the population from the page and stores the population as `<population>26,813,057 (July 2001 est)</population>` into the record.

The following shows a list of steps for implementing the `submit` method.

1. Download the list of country names and the corresponding URLs
2. Submit the URL and fetch the country page
3. Instantiate a record
4. Set the record's source name to World Factbook
5. Set the record's local rank to 1
6. Set the record's snippet by parsing the country page

How to implement the `next` method?

When the wrapper's `next` method is called, the wrapper returns its data structure, a record. If the record is null, the wrapper returns *null*. Unlike the search engine wrapper, it does not re-submit the keyword expression. The reason is that a search engine returns thousand pages, but World Factbook returns only one page for a country.

5.4.4 Abbreviation Wrappers

The Abbreviation Wrapper is used to provide the interface between a record retriever and an abbreviation site. At this stage, we have only one wrapper for the abbreviation page of CIA's World Fact book. The abbreviation page is shown in Figure 5.4. The URL of the page is the following:

`http://www.cia.gov/cia/publications/factbook/docs/app-a.html`

The page contains a list of acronyms and what each acronym stands for. The acronyms represent the communities and organizations around the world. For example, *AMF* stands for *Arab Monetary Fund*. The wrapper is called by a record retriever when the category is abbreviation.

What is the data structure?

The data structure for the abbreviation wrapper is a record list. Like the search engine wrapper, the record list is implemented a queue. Each element of the queue contains a record. Initially, the record list is empty.

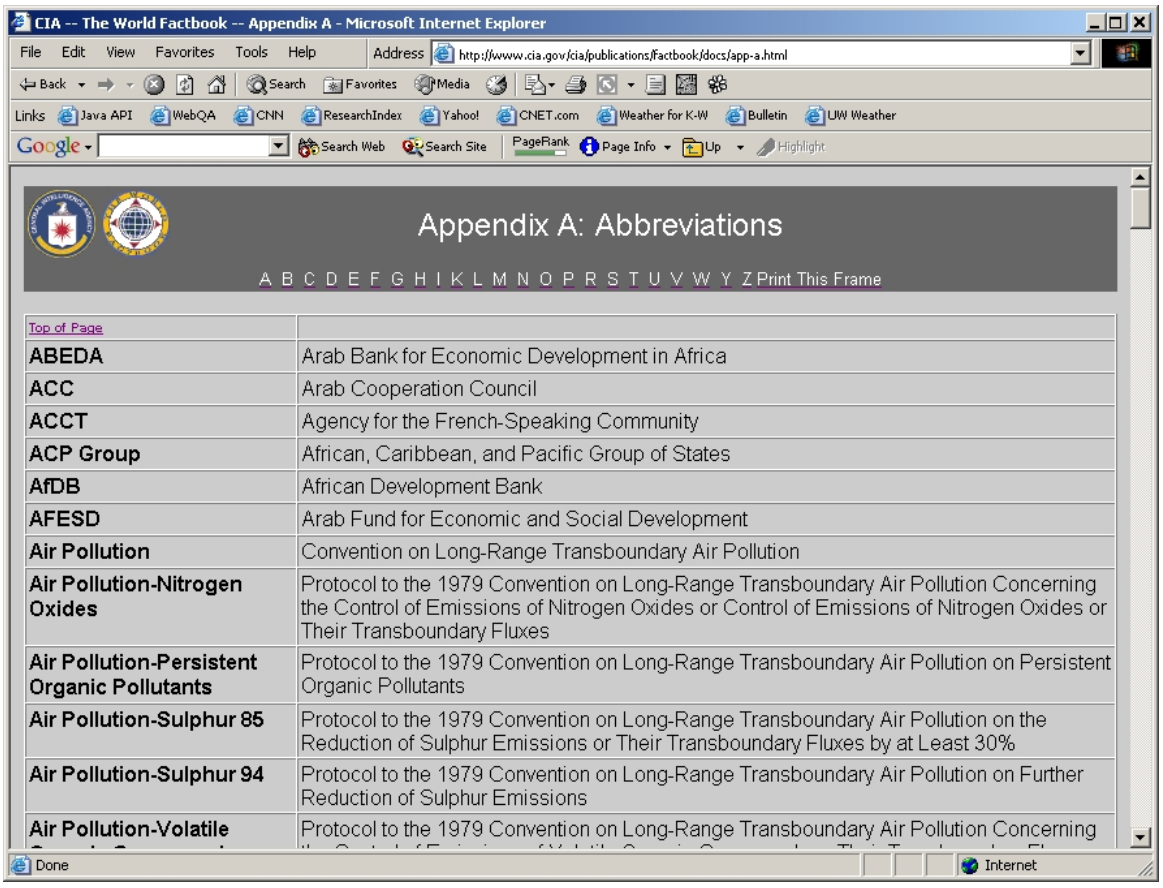

Figure 5.4: The abbreviation Web page of CIA's World Factbook

How to implement the `submit` method?

Although, the `submit` method takes a keyword expression as a parameter, the abbreviation wrapper does not use it. First, it downloads the Web page using the URL above. Second, it extracts all the acronyms and the full spellings from the page.

After the wrapper extracts all the acronyms and the full spellings, for each pair of an acronym and a full spelling, the wrapper instantiates a record object. It sets the record's source attribute to World Factbook's Abbreviation site. It sets the record's local rank to the order of the acronyms that is shown in the page. Later on, it adds "`<`" and "`>`" to the acronym for creating an open tag. It adds "`</`" and "`>`" to the acronym again for creating an close tag. It appends the open tag, the full spelling, and the close tag together. For example, if the acronym is *ABED* and the full spelling is *Arab Bank for Economic Development in Africa.* Then the appended string is:

```
<ABEDA>Arab Bank for Economic Development in Africa</ABEDA>
```

The appended string is set to the record's snippet attribute. The wrapper adds the record to its record list. The wrapper repeats the action for each pair of an acronym and a full spelling, and add each record to its record list.

How to implement the `next` method?

For the next method, the wrapper removes and returns the first record of its record list if there is one available. Otherwise, it returns *null.*

5.4.5 Weather Wrappers

At this stage, we have implemented only one weather wrapper, a wrapper for Weather Underground [WUn02]. Weather Underground is a Web site that provides weather information on over 60,000 major international cities. Such weather information are temperature, humidity, dew point, wind, pressure, condition, clouds, sunrise, sunset, moon rise, and moon set. It provides a text box for a user to submit a keyword expression (city name) to the site. Whenever a user asks a weather information to WebQA, WebQA can query this weather site and returns the weather information to the user.

What is the data structure?

The data structure for the weather wrapper is a weather page list. The weather page list is implemented as a queue. Each element in the weather page list is a weather Web page, which contains weather information about a city. Initially, the weather page list is empty.

How to implement the `submit` method?

The weather wrapper's `submit` method takes a keyword expression as an input. The keyword expression is the name of the city or possibly followed by a country name or a state if the country is United States. The reason why the keyword expression is the name of the city is that the WebQAL Generator subcomponent treats the weather query this way.

When the weather wrapper's `submit` method is called, the wrapper takes the

keyword expression and submits it to Weather Underground. In order to submit the keyword expression to Weather Underground, we have to do the similar tricks as the search engine wrapper. Submitting the keyword expression in the text box is just like submitting the URL with keyword expression attached to it. For example, if the keyword expression is *"Toronto Canada"*, then the URL is the following:

```
http://www.wunderground.com/cgi-bin/findweather/getForecast
?query=Toronto+Canada
```

As we can see, the keyword is attached at the end of the URL with a plus sign between every keyword. This is just like submitting the keyword expression in the text box of Weather Underground.

After the weather wrapper submits the keyword expression to the Weather Underground, there are three types of pages that the Web site returns. There are *no-result*, *exactly-one-result*, and *multiple-results*. The *no-result* type means the keyword expression is not a valid city name or the weather site does not have weather information for that city. For example, if we submit *"Kitchener Canada"* as the keyword expression, Weather Underground will return an error page as shown in Figure 5.5. Therefore, the weather wrapper performs nothing further.

If the result page is a *exactly-one-result* type, then the return page contains weather information about the city. For example, if we submit *"Ottawa Canada"* as a keyword expression, Weather Underground will return the weather information about the city as shown in Figure 5.6. Then the weather wrapper adds the result page to its weather page list.

If the result page is a *multiple-results* page, the result page contains a list

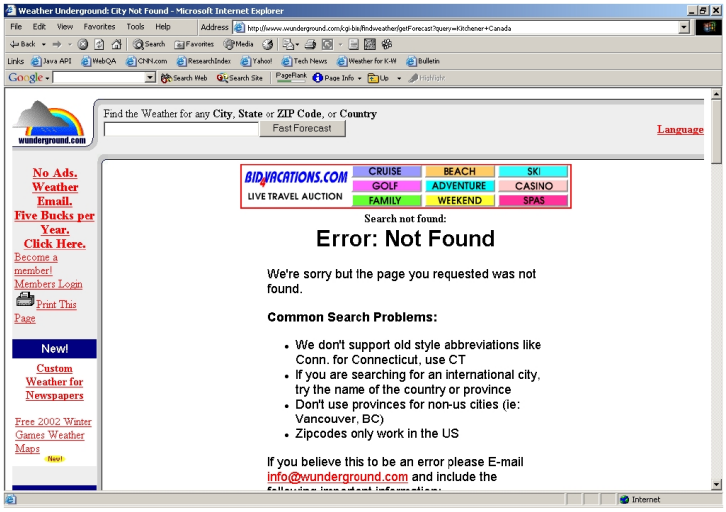

Figure 5.5: A sample Weather Underground page with no result

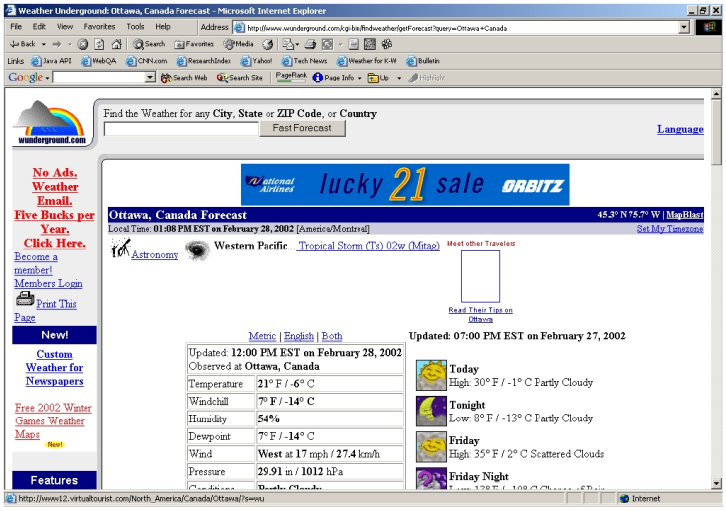

Figure 5.6: A sample Weather Underground page with exactly one result

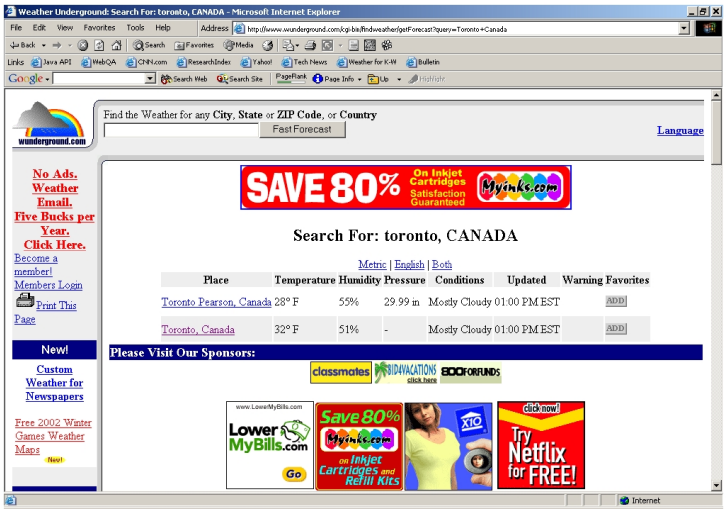

Figure 5.7: A sample Weather Underground page with multiple results

of matched cities with hyperlinks. For example, if submit a keyword expression *"Toronto Canada"*, Weather Underground returns a list of matched cities: *Toronto* and *Toronto Pearson* as shown in Figure 5.7. There is a hyperlink for each matched city. The weather wrapper concurrently download all the weather pages for the listed hyperlinks. Each return weather page is added to the weather wrapper's weather page list.

Algorithm 7 shows the pseudo code for implementing the Weather Underground wrapper.

How to implement the `next` method?

When the weather wrapper's `next` method is called, the weather wrapper checks whether or not its weather page list is empty. If the weather page list is empty, then it returns *null*. Otherwise, it removes the first element (i.e., a weather

Algorithm 7 The pseudo code for the Weather Underground wrapper's `submit` method

```
function submit(KeywordExpression k) : void
resultPage ⇐ downloaded page by submitting k to Weather Underground
if resultPage is a error page then
   Do nothing
else if resultPage has exactly one result then
   Add the current page to the weather page list
else
   //resultPage has multiple results
   For each city, download the weather page concurrently with the associated
   hyperlink
   Add each weather page to the weather page list
end if
```

page) from the weather page list, converts the first element into a record, and returns the record. In order to convert the first weather page into a record, we need to instantiate a record and initialize its attributes. Recall that there are three attributes of a record: a source name, a local rank, and a snippet. The source name is set to *Weather Underground*. The local rank is set to i for the ith weather page removed from the weather page list. As for initializing the snippet attribute, we need to parse the HTML code in the weather in order to find the desired weather information. It first examines the HTML title tag (i.e., `<TITLE>`). It contains the name of city, from which the wrapper extracts the city name and adds city tag around it. For example, if we are looking for the weather on *Toronto, Canada*, then the HTML title tag contains `<TITLE>Weather Underground: Toronto, Canada Forecast</TITLE>`. The wrapper converts the title tag into city tag, therefore, we have `<city>Toronto, Canada</city>`. We append `<city>Toronto, Canada</city>` into the snippet. Then the weather wrapper uses the similar technique for extracting other weather information and store them in a tag format. At the end, all tagged information are put into a record, and

return the record.

Chapter 6

Answer Extractor

Answer Extraction (AE) is the last main component of the QA Engine. It extracts answers from the ranked records. The inputs of this component are a WebQAL expression and a list of ranked records returned by SR. As shown in Figure 6.1, AE passes the WebQAL expression and the list of ranked records to a subcomponent, Candidate Identifier (Section 6.1), which identifies a list of the candidates from the list of records. It verifies each candidate by checking it against the category and the output parameters that are listed in the WebQAL. A candidate is a possible answer with a score. The higher the score, the more likely is the candidate the answer to the user's query. The score is based on the frequency of the candidate that appears in the records. AE passes the list of candidates to the next subcomponent, Best Candidate Retriever (Section 6.2). Best Candidate Retriever outputs the top ten candidates using 10 simple linear searches. AE passes the top ten candidates to subcomponent, Rearranger (Section 6.3), which rearranges the top ten candidates if necessary. Finally, AE passes the rearranged list of candidates to Output Converter

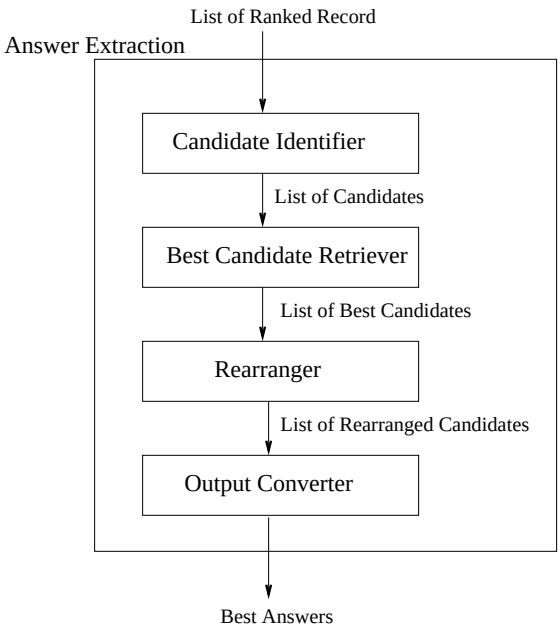

Figure 6.1: The architecture of Answer Extraction

(Section 6.4) for converting to user readable format. This would be an array of strings where each string contains a possible answer. A score can be attached with the answer for confirming its accuracy.

6.1 Candidate Identifier

6.1.1 What is the data structure?

A candidate list is the data structure of the Candidate Identifier (CI) subcomponent. It contains a list of candidate, where a candidate has two attributes: a name and a score. The name is written in string format and the score is written in real number format as shown in Table 6.1. The candidate list is an object of a class

`CandidateList`. `CandidateList` inherits another class called `SortedArrayList`, which inherits Java's `ArrayList` class.

Table 6.1: The data structure of a candidate

Attribute	Data Type
Name	String
Score	double

According to Sun Microsystems' Java 2 version 1.4 API specification, the `ArrayList` class is inside the `java.util` package [Jav02]. `ArrayList` is a resizable array. It is similar to a vector. It provides a number of important methods. The `size` and `get` operations are run in constant time. The `add` operation runs in amortized constant time, that is, adding n elements requires O(n) time.

The `SortedArrayList` class takes a comparator in the constructor. The comparator is used to compare two objects for deciding which one precede the other. The `SortedArrayList` class has all the services that the `ArrayList` class provides. It adds a `binarySearch` method and overrides the `add` method. The `binarySearch` method takes an object as a parameter. It returns the index of the specified object in its sorted array using the binary search algorithm. If the specified object is not found, it returns (-(insertion point) - 1). The array must be sorted into ascending order according to the natural ordering of its elements by using the comparator. Algorithm 8 shows the pseudo code for the method. The `add` method takes an object as an parameter. It calls the `binarySearch` method for finding the index of the specified object. If the object is not found, it adds the object according to the index using `ArrayList`'s `add` method (i.e., add($index, Object$)). Otherwise, it does

nothing. Algorithm 9 shows the pseudo code for the method.

Algorithm 8 The pseudo code for `SortedArrayList`'s `binarySearch` method

```
function binarySearch(Object object) : int
return binarySearch(object, 0, size−1)

function binarySearch(Object object, int beginIndex, int endIndex) : int
if endIndex < beginIndex then
   return −1 ∗ beginIndex − 1
end if
middleIndex ⇐ middleIndex(beginIndex, endIndex)
middleObject ⇐ this.get(middleIndex)
if object = middleObject then
   return middleIndex
else if object is behind middleObject then
   return binarySearch(object, middleIndex + 1, endIndex)
else
   //if object precedes middleObject
   return binarySearch(object, beginIndex, middleIndex − 1)
end if

function middleIndex(int beginIndex, int endIndex) : int
size ⇐ endIndex − beginIndex + 1
if size is even then
   return ⇐ size/2 + (beginIndex − 1)
else
   return size/2 + beginIndex
end if
```

Algorithm 9 The pseudo code for `SortedArrayList`'s `add` method

```
function add(Object object) : void
index ⇐ binarySearch(object)
if index < 0 //If the the object does not exist in the list then
   ArrayList's add(−1 ∗ index − 1, object)
end if
```

The `CandidateList` class inherits `SortedArrayList` class, therefore, it has all the services that the `SortedArrayList` class provides. It overrides the `add` method. The `add` method takes a candidate as a parameter. The method calls `SortedArrayList`'s `binarySearch` method for finding the index of the specified

candidate in its sorted array. If the specified candidate exists in `CandidateList`'s sorted array, then the method adds the score of the specified candidate to the score of the candidate that exists in the sorted array. Otherwise, the method simply adds the candidate to the sorted array with its current score. The algorithm for the `add` method of the `CandidateList` class is shown in Algorithm 10.

Algorithm 10 The pseudo code for `CandidateList`'s `add` method

```
function add(Candidate candidate) : void
index ⇐ binarySearch(candidate)
if index ≥ 0 //If the the candidate exist in the list then
   existedCandidate ⇐ get(index)
   existedCandidate.score ⇐ existedCandidate.score + candidate.score
else
   //If the the object does not exist in the list
   SortedArrayList's add(−1 ∗ index − 1, object)
end if
```

6.1.2 How does CI work?

The Candidate Identifier (CI) subcomponent takes a WebQAL expression and a list of ranked records as input and produces a list of possible candidates where each candidate has its own score. It instantiates an empty candidate list. For each record in the list of ranked records, CI checks the source type of the record and calls one of the four sub-identifiers accordingly. If the source type is a search engine, CI calls the Search Engine Sub-identifier (Section 6.1.6). If the source type is a country site, CI calls the Country Sub-identifier (Section 6.1.3). If the source type is a abbreviation site, CI calls the Abbreviation Sub-identifier (Section 6.1.4). If the source type is a weather site, CI calls the Weather Sub-identifier (Section 6.1.5). Algorithm 11 shows the pseudo code for CI.

Algorithm 11 The pseudo code for CI

```
Require: recordList, webqal
Ensure: candidateList is a list of identified candidates
  candidateList ⇐ ϵ
  for each record r in recordList do
    snippet ⇐ r.snippet
    recordType ⇐ source type that r is generated from
    if recordType = search engine then
      Call Search Engine Sub-identifier
    else if recordType = country site then
      Call Country Sub-identifier
    else if recordType = abbreviation site then
      Call Abbreviation Sub-identifier
    else
      //recordType = weather site
      Call Weather Sub-identifier
    end if
  end for
```

6.1.3 Country Sub-identifier

The Country Sub-identifer takes a snippet, a WebQAL expression, and a candidate list. The sub-identifier is called by CI when the record is from a country information source (i.e., CIA's World Factbook). Therefore, the specified snippet is written in a tag-like format that is described in Section 5.4.3. The sub-identifier creates an empty string called *possibleAnswer*. For each open tag, the sub-identifier checks that the open tag is a substring of the keyword expression in the specified WebQAL expression. If so, the sub-identifier appends the content between the open tag and the associated close tag to *possibleAnswer*. After each open tag has been examined, the sub-identifier instantiates a candidate and sets *possibleAnswer* to the candidate's name attribute. Since the snippet is retrieved from CIA's Web site, the information in it is pretty accurate. The sub-identifier sets the candidate's score to maximum, which is the maximum value of a double type. In the end, the

sub-identifier adds the candidate to the specified candidate list.

6.1.4 Abbreviation Sub-identifier

The Abbreviation Sub-identifier takes a snippet, a WebQAL expression, and a candidate list. From the WebQAL expression, the sub-identifier extracts the output option and the keyword expression. Since it is called by CI when the record is from an abbreviation site, the specified snippet is written in a tag-like format that is described in Section 5.4.4. For each open tag, the sub-identifier extracts the word in the open tag, which is the acronym. In addition, the sub-identifier extracts the words between the open tag and the close tag, which is the full spelling of the acronym.

When the output option is "*long*", the sub-identifier checks the acronym is a match of the keyword expression. If so, it instantiates a candidate. It sets the candidate's name attribute to the full spelling and sets the candidate's score attribute to approximate infinity (i.e., the maximum number of a double type in Java). In the end, it adds the candidate to the candidate list and returns the candidate list.

When the output option is "*short*", the sub-identifier checks the full spelling against the keyword expression. If they match, the sub-identifier instantiates a candidate. It sets the candidate's name attribute to the acronym and sets the score to approximate infinity. At the end, it adds the candidate to the candidate list and return the candidate list.

Here is an example of how the sub-identifier identifies a candidate. Let's say the

given WebQAL expression is `abbreviation -output long -keywords ACC`. The given snippet is written in the following:

```
<ABEDA>Arab Bank for Economic Development in Africa</ABEDA>
<ACC>Arab Cooperation Council</ACC>
<ACCT>Agency for the French-Speaking Community</ACCT>
...
```

First, the sub-identifier extracts the first acronym (*ABED*) and the corresponding full spelling (*Arab Bank for Economic Development in Africa*). Since the output option is *long*, it compares the acronym and the keyword expression in the WebQAL expression. Unfortunately, *ABED* is not equal to the keyword expression, *ACC*, so the sub-identifier examines the next set of tags. It extracts the second acronym (*ACC*) and the corresponding full spending (*Arab Cooperation Council*). Since ACC exactly matches the keyword expression, the sub-identifier instantiates a candidate. The sub-identifier sets the candidate name to the full spelling, *Arab Cooperation Council* and sets the score to approximate infinity. It adds the candidate to the given candidate list.

6.1.5 Weather Sub-identifier

The Weather Sub-identifier takes a snippet, a WebQAL expression, and a candidate list as inputs. If the output option is "*all*", then it extracts all the tag information and information between the open tags and close tags from the snippet. The sub-identifier instantiates a candidate. All the tag information are appended together and are set to the candidate's name. the sub-identifier assigns an approximate

infinity to the candidate's score. It adds the candidate to the given candidate list.

If the output option is not "*all*", the sub-identifier finds the open tag that matches the output option. It instantiates a candidate, where the content between the open tag and close tag is the candidate's name and an approximate infinity is the candidate's score. The candidate is added to the given candidate list.

6.1.6 Search Engine Sub-identifier

The inputs for the Search Engine Sub-identifier is a snippet, a WebQAL expression, and a candidate list. Like all other sub-identifiers, it is called by CI. The first thing that this sub-identifier does is to set the range of the number of words of a candidate. The range is category dependent. Table 6.2 shows the range of the number of words of a candidate for all categories except the *Abbreviation* category. The first column lists all the categories. The second and third column lists the minimum and maximum number of words of a candidate, respectively. These numbers are generated based on the possible number of words that can appear as answers. Algorithm 12 shows how to calculate the ranges of words of a candidate for the *Abbreviation* category.

Table 6.2: The ranges of the number of words for each category

Category	Minimum	Maximum
Name	1	3
Place	1	4
Time	1	4
Quantity	1	5
Weather	N/A	N/A
Other	1	5

Algorithm 12 The pseudo code for calculating the number of words of a candidate *Abbreviation* category

Require: $outputOption$, $keywordExpression$
Ensure: min/max is the minimum/maximum number of words of a candidate
  if $outputOption$ = long **then**
    $min \Leftarrow 1$
    $max \Leftarrow$ number of characters in $keywordExpression$
  else
    $//outputOption$ = short
    $min \Leftarrow 1$
    $max \Leftarrow 1$
  end if

After defining the range of number of words of a candidate, the sub-identifier instantiates a validator for verifying whether a string is a possible candidate. In other words, if the validator returns false, then the given string cannot be a possible answer. Again, the sub-identifier instantiates a specific validator for specific category.

Remember that the sub-identifier sets the range of number of words for a candidate. For each number of word, the sub-identifier breaks the snippet into sentences. For each sentence, the sub-identifier looks at all the possible sequence of words. The number of words must match the sequence. Then, the sequence is checked against the given validator. If the validator determines that the given sequence is a possible answer, then sub-identifier instantiates a candidate, sets the candidate's name attribute to the sequence and sets the candidate's score attribute to 1. It adds the candidate to the given candidate list. Algorithm 13 shows the pseudo code for the Search Engine Sub-identifier.

Algorithm 13 The pseudo code for the Search Engine Sub-identifier

```
Require: snippet, webqal, candidateList
Ensure: //Identify all the possible candidates and add them to candidateList
  min ⇐ minimum number of words of a possible answer (category-dependent)
  max ⇐ maximum number of words of a possible answer (category-dependent)
  validator ⇐ a validator that verifies the possible answer (category-dependent)
  for i = min to max do
    for each sentence s do
      for each sequence of words with length i, called w do
        if validator returns true for w then
          Instantiate candidate
          candidate.name ⇐ w
          candidate.score ⇐ 1
          candidateList.add(candidate)
        end if
      end for
    end for
  end for
```

Validators

As mentioned in Section 6.1.6, there is a validator for each category, except *Weather*. Each validator verifies whether or not the given sequence of words is a possible answer for that category. Name Validator ensures that the given sequence of words does not contain any value in the keyword parameter, that it contains only letters, that each word starts with a capital letter, and that not all the words are stopwords.

Place Validator first checks the output parameter. If the output parameter asks for a country, then the possible answer must be found in the predefined list of countries. Same techniques are used for asking a state, a city, and a continent. We retrieve the predefined Lists of all possible states, cities, countries, and continents from CIA's World Factbook. If the output parameter is "*unknown*", then the same technique is used in name category.

Like Place Validator, Time Validator first checks the output parameter to see

what kind of output the query is looking for. If the query is looking for a month, then the validator checks the sequence words must be length of one and it has to be one of the twelve months. Same techniques for other values of the output parameter.

Quantity Validator depends on whether or not the output parameter is a unit. If the parameter value is a unit (not a dollar sign, \$), then the validator checks whether or not the last word of the given sequence of words matches the output unit. If the output parameter is a dollar sign, then it makes sure the given sequence starts with a dollar sign or ends with the word "dollar" or "dollars". At this stage, the quantity validator cannot check other currencies such as *yen*.

Regardless of the output parameter, the Quantity Validator makes sure that the rest of the sequence of the words is a number. A number can be written in a numerical form such as *35*, or in English form such as *thirty-five*. For the latter case, it is a little bit difficult, but a converter has been written, so it can convert a number in English form to numerical form. Algorithm 14 is the pseudo code for the converter. It returns -1 if the given sequence of word does not mean a number.

As stated earlier, *Abbreviation* category has two output options: "*long*" and "*short*". If the output option is "*long*", then the keyword is the short form (abbreviation) and the possible answer is the long form (full spelling of the abbreviation). If the output option is "*short*", then the keyword is the long form, and the possible answer is the short form. In both cases, Abbreviation Validator makes sure that the long form does not start with a stopword or end with a stopword. It removes all the stopwords in the long form. The new long form cannot be empty string and

Algorithm 14 The pseudo code for converting a number in English to numerical value

Require: $numberInEnglish$
Ensure: $numericalValue$ = numerical value of $numberInEnglish$
  Change all the hyphens (-) in $numberInEnglish$ to a space
  Remove the word "and" in $numberInEnglish$
  $numbericalValue \Leftarrow 0$
  $chunkInEnglish \Leftarrow$ empty string
  for each word w in $numberInEnglish$ **do**
    if $w =$ "thousand" or "thousands" **then**
      $chunkValue \Leftarrow$ chuckToNum($chunkInEnglish$) (Algorithm 15)
      if $chunkValue < 0$ **then**
        return -1
      else if $chunkValue = 0$ **then**
        $numericalValue \Leftarrow numericalValue + 1000$
      else
        $numericalValue \Leftarrow numbericalValue + chunkValue * 1000$
      end if
      $chunkInEnglish \Leftarrow$ empty string
    else if $w =$ "million" or "millions" **then**
      $chunkValue \Leftarrow$ chuckToNum($chunkInEnglish$)
      if $chunkValue < 0$ **then**
        return -1
      else if $chunkValue = 0$ **then**
        $numericalValue \Leftarrow numericalValue + 1000000$
      else
        $numericalValue \Leftarrow numbericalValue + chunkValue * 1000000$
      end if
      $chunkInEnglish \Leftarrow$ empty string
    else if $w =$ "billion" or "billions" **then**
      $chunkValue \Leftarrow$ chuckToNum($chunkInEnglish$)
      if $chunkValue < 0$ **then**
        return -1
      else if $chunkValue = 0$ **then**
        $numericalValue \Leftarrow numericalValue + 1000000000$
      else
        $numericalValue \Leftarrow numbericalValue + chunkValue * 1000000000$
      end if
      $chunkInEnglish \Leftarrow$ empty string
    else
      $chunkInEnglish \Leftarrow chunkInEnglish + "" + w$
    end if
    $chunkValue \Leftarrow$ chuckToNum($chunkInEnglish$)
    if $chunkValue < 0$ **then**
      return -1
    else
      return $(numericalValue + chunkValue)$
    end if
  end for

Algorithm 15 The pseudo code for converting a chunk in English to numerical value

Require: $chunkInEnglish$
Ensure: $chunkValue$ = numerical value of $chunkInEnglish$
  $chunkValue \Leftarrow 0$
  for each word w in $chunkInEnglish$ **do**
    $wordValue \Leftarrow$ wordToNum(w) (Algorithm 16)
    if $wordValue < 0$ **then**
      $chunkValue \Leftarrow -1$
      break
    else
      if $wordValue = 100$ **then**
        if $chunkValue = 0$ **then**
          $chunkValue \Leftarrow wordValue$
        else if $chunkValuemod10 \neq 0$ **then**
          $chunkValue \Leftarrow chunkValue * wordValue$
        else
          $chunkValue \Leftarrow -1$
          break
        end if
      else
        if $wordValue < 10$ **then**
          if $chunkValue$ mod $10 = 0$ and $(chunkVallue$ mod $100)/10 \neq 1$ **then**
            $chunkValue \Leftarrow chunkValue + wordValue$
          else
            $chunkValue \Leftarrow -1$
            break
          end if
        else
          if $chunkValue$ mod $100 = 0$ **then**
            $chunkValue \Leftarrow chunkValue + wordValue$
          else
            $chunkValue \Leftarrow -1$
            break
          end if
        end if
      end if
    end if
  end for

Algorithm 16 The pseudo code for converting a word in English to numerical value

```
Require: wordInEnglish
Ensure: wordValue = numerical value of wordInEnglish
  if wordInEnglish = "one" then
    wordValue ⇐ 1
  else if wordInEnglish = "two" then
    wordValue ⇐ 2
  else if ... then
    ...
  else if wordInEnglish = "twenty" then
    wordValue ⇐ 20
  else if wordInEnglish = "thirty" then
    wordValue ⇐ 30
  else if ... then
    ...
  else if wordInEnglish = "hundred" then
    wordValue ⇐ 100
  else
    wordValue ⇐ −1
  end if
```

it contains only letters. Also, every word in the long form must start with an upper case letter. Finally, the validator checks the short form is the abbreviation of the long form.

The simplest algorithm for checking for a match between the short form and the long form is to extract the first letters of all the words of the long form and compare them with all the letters in the short form. For example, the abbreviation **IBM** stands for **I**nternational **B**usiness **M**achines. However, this algorithm does not work for all the abbreviations. Here are two counter examples: the acronym **TREC** stands for **T**ext **Re**trieval **C**onference and the abbreviation **Max** stands for **Max**imum. All abbreviations are not constituted from the first letters of every word in the long form. We have designed an function (Algorithm 17) that can handle most, but not all abbreviations. It returns true if *shortForm* stands for *longForm*.

For example, *Y2K (Year 2000)* and *XML (Extensible Markup Language)* can not be properly validated. Although, the algorithm misses some abbreviations, the Validator can consult some Web sites that contain a list of abbreviations and what they stand for (e.g., CIA World Fact Book 2002). We have tested this combined validation approach against TREC-9 questions and obtained very good result which are discussed in Section 7.1.

Algorithm 17 The abbreviation validator

```
function match(String shortForm, String longForm) : boolean
if shortForm = ε then
  return true
else if longForm = ε then
  return false
else
  x ⇐ first letter of shortForm
  x' ⇐ rest of shortForm
  y ⇐ first letter of longForm
  y' ⇐ rest of longForm
  if x = y then
    if match(x', y') then
      return true
    else
      z ⇐ longForm without the first token
      return match(x', z)
    end if
  else
    return false
  end if
end if
```

6.2 Best Candidate Retriever

The function of the subcomponent, Best Candidate Retriever is to retrieve the best candidates from the given candidate list. By best candidates, we mean the candidates that have highest scores. This given candidate list is sorted by the

Candidate's name, not its score. First, we need to store the list by the score. An WebQA's engine option, "`-ans` n", sets the number of returned answers to n. Then we pick the first n candidates from the specified list and returned the best candidate list.

6.3 Rearranger

The job of Rearranger is to rearrange the best candidate list. The extraction technique that is used in Candidate Identifier is based on the most frequent word count. In other words, the score is equal to the frequency of the sequence of the words. Consequently, a candidate that contains another candidate will have lower score, since the sequence of words that appears in the containing candidate will also appear in the contained candidate. For example, if we are looking for the inventor of the telephone. Two of the possible candidates are *"Alexander Graham Bell"*, and *"Bell"*. *"Bell"* has a higher frequency than *"Alexander Graham Bell"*, because *"Bell"* appears inside of *"Alexander Graham Bell"*. However, *"Alexander Graham Bell"* should have a higher score, because it contains more information. Therefore, rearrangement of the best candidate list is necessary.

Rearranger rearranges each candidate in the best candidate list starting from the end. For each candidate, the current index is passed to the function, *getBestPosit*. The function returns a new index of the candidate. Rearranges moves the candidate in the current index position to the new index position. Algorithm 18 shows the pseudo code for the function, *getBestPosit*. We call the result list, the rearranged candidate list.

Algorithm 18 The *getBestPosit* function.

```
function getBestPosit(CandidateList bestCandidateList, int currentIndex) :
int
currentCandidate ⇐ bestCandidateList.get(currentIndex)
for i = 1 to bestCandidateList.size() do
  candidate ⇐ bestCandidateList.get(i)
  if number of words in candidate.name < number of words in
  currentCandidate.name then
    if all the words in candidate.name appears in currentCandidate.name
    then
      return i
    end if
  end if
end for
return currentIndex
```

6.4 Output Converter

The Output Converter subcomponent, as suggested by its name, converts an internal format to a user readable format. Its output is the output of WebQA's engine. It converts the rearranged candidate list to an array of strings. Each string has an answer followed by a score with brackets.

Instead of returning a list of exact answers or just one exact answer, a better way is to return a English sentence that is able to answer the question like a dialog. An example of such a system is START [STA02]. If a user asks the question, *"What is the capital city of Canada?"*, START replies *"Ottawa, Ontario is the capital of Canada."*. To return a sentence instead of just the exact answer would be a future research in the area of natural language understanding.

Chapter 7

Evaluation

WebQA has been implemented. The implementation is based on the techniques that described in Chapter 3. We need to measure WebQA performance in order to understand its effectiveness. In this chapter we report on the results of experiments that evaluate WebQA's effectiveness and performance. Measurements are made using the TREC-9 (Ninth Text REtreival Conference) specification, which is a well-known QA evaluation competition. TREC-9 is described in Section 7.1, followed by a discussion of the performance evaluation methodology (Section 7.2) and experimental results (Section 7.4).

7.1 TREC-9

TREC-9 is used to measure the accuracy of QA systems. TREC-9 specifies a list of 693 questions that a QA system is expected to answer using a corpus of off-line documents. QA system answers are ranked responses, and, for each the rank

questions, a score is assigned to the system based on , at which the correct answer is found. The score is $1/n$ where n is the rank of the correct answer. Thus, the score is between 0 and 1. The system gets a score of 1 if it ranks the correct answer as the first one; if the correct answer is not found in the list, then the score of 0 is given. In the end, the average score of all questions is computed as the score of the system. Table 7.1 shows the TREC-9 score for each of the 22 groups that participates in TREC-9 [TRE01]. The groups submitted 50-byte-limit runs with their best results.

Table 7.1: TREC-9 score for TREC-9 participants

Rank	Score	Group
1	0.580	Southern Methodist University
2	0.321	University of Waterloo MultiText Project
3	0.318	USC/Information Sciences Institute
4	0.315	IBM T. J. Watson
5	0.284	Queen's College, CUNY
6	0.260	Korea University
6	0.249	Syracuse U., Center for National Language Processing
7	0.231	NTT DATA Corp.
8	0.231	Imperial College of Science, Technology and Medicine
9	0.230	Alicante University
10	0.227	Xerox Research Centre Europe
11	0.212	Korea Advanced Institute of Science and Technology
12	0.206	University of Sheffield
13	0.196	Microsoft Research Ltd
14	0.195	Fudan University
15	0.180	University of Alberta
16	0.179	University of Montreal
17	0.178	LIMSI
18	0.135	CL Research
19	0.101	Seoul National University
20	0.084	Dipartimento di Informatica (Pisa)
21	0.064	Conexor Oy
22	0.038	The MITRE Corporation

7.1.1 Problems with TREC-9

Although TREC-9 is a well-known QA evaluation system, it is not well suited for QA systems that uses online data, such as WebQA. The problems of using TREC-9 in our setting are the following.

First, the TREC-9 evaluation system expects a QA system to return a 50-byte or 250-byte passage, not the exact answers. As long as the correct answer is found in the passage, TREC-9 assumes it is correct. If a new QA system allows to return a list of passages instead of a list of exact answers, then it is more like a search engine.

The evaluation system requires a QA system to find answers from the off-line document corpus. This allows the computation of standard information retrieval metrics such as precision and recall. However, systems such as WebQA search for answer in an open environment such as the Web, where calculation of these metrics is not possible.

Since the data source of WebQA is open, the data changes dynamically. The TREC-9 score for WebQA changes dynamically as well. Today's score might be different than tomorrow's score.

In addition, the evaluation system has no correct answer provided to some question. For example, Question 443–"What is the chemical formula/name for napalm?" and Question 333–"What is the name of the second space shuttle?". In addition there are 9 more questions that don't have an answer in the pattern file. They are Questions 322, 339, 443, 591, 598, 656, 663, 794, 796, and 811.

Some questions have more than one answer, the TREC-9 correct answer list

(the pattern file) does not cover all of them. For example, Question 563–"Name a movie that the actress, Sandra Bullock, had a role in." Certainly, the pattern file does not cover one of her latest movies, "Miss Congeniality".

7.1.2 New Pattern File

A pattern file is a text file that contains all possible correct answers for each TREC-9 question. As mentioned in Section 7.1.1, the file is outdated. The pattern file is designed so the off-line QA systems are able to find correct answers from a corpus of off-line documents, which is out-dated as well. However, WebQA uses the Web as data sources, so we need to correct the pattern file for finding the most updated answers. To solve this, we can read each question by hand and find out all the correct answers. Then we use the new pattern file for evaluating WebQA.

7.2 Efficiency Analysis

Although the accuracy is the most important measure of a QA system, efficiency should be measured as well. If the system produces a good result, but requires more than minutes to process a question, then the user might as well use the search engines. The TREC-9 evaluation system only focuses on the accuracy of a QA system.

To test WebQA's performance is a difficult problem. The reasons are due to the nonstable response time of network and source execution time. However, we can analyze WebQA's query response time in terms of its components. The response

time of answering a query consists of communication time and the execution time.

$$t_{resp} = t_{comm} + t_{exec} \tag{7.1}$$

where t_{resp} denotes the total response time of a query; t_{comm} denotes the communication cost between a client and WebQA; t_{exec} denotes the execution time for WebQA engine. Execution time can be divided into local execution time (t_{exec}) which accounts for the cost of executing WebQA engine and remote execution time (t_{remote}) which accounts for the cost of waiting the Web sources as shown in Equation 7.2.

$$t_{resp} = t_{comm} + t_{local} + t_{remote} \tag{7.2}$$

Furthermore, the local execution time (t_{local}) consists of the cost for WebQA engine before access the Web sources (denoted by $t_{pre-source}$) and the cost for WebQA engine after access the Web sources (denoted by $t_{pre-post}$) as shown in Equation 7.3.

$$t_{resp} = t_{comm} + t_{pre-source} + t_{post-source} + t_{remote} \tag{7.3}$$

All Web sources are running concurrently for retrieving records. Therefore, the remote execution time is the maximum cost of waiting for all search engines.

For t_{comm} and t_{remote}, there is nothing we can do, because there are outside the WebQA. $t_{pre-exec}$ is source-independent. As long as the workload of the executing computer is the same, we should able to have the same pre-source execution for each query. $t_{post-exec}$, however, is source-dependent. Although the retrieved list of records from different sources is different, there should only be a slightly difference

in terms of response time, because the extraction technique is the same.

7.3 Experimental Procedure

Since WebQA is not a traditional QA system, the way to use TREC-9 to evaluate WebQA should be somewhat different. First of all, WebQA uses search engines and other Web sites as its data sources, not a list of off-line documents. Therefore, WebQA does not require TREC-9 off-line documents. Second, a list of questions is inserted to WebQA, so we need a program that is able to take a list of TREC-9 questions and plug them one-by-one to WebQA's engine, then format all the generated lists of answers, so that the TREC-9 evaluation system is able to accept the submission file as input. The architecture for generating a TREC-9 submission file from WebQA is shown in Figure 7.1.

First, a file that contains a list of questions is input to the WebQA evaluation system. The evaluation system passes the list of questions to its TREC-9 Question Parser. The parser converts all the questions from TREC-9 format to an array of questions in string format, without changing the content of the questions. Here is an example of a TREC-9 question in TREC-9 format.

```
<top>
<num> Number:  216
<desc> Description:  What is the primary language of the Philippines?
</top>
```

After that, the evaluation system takes each question in the array and sends it to

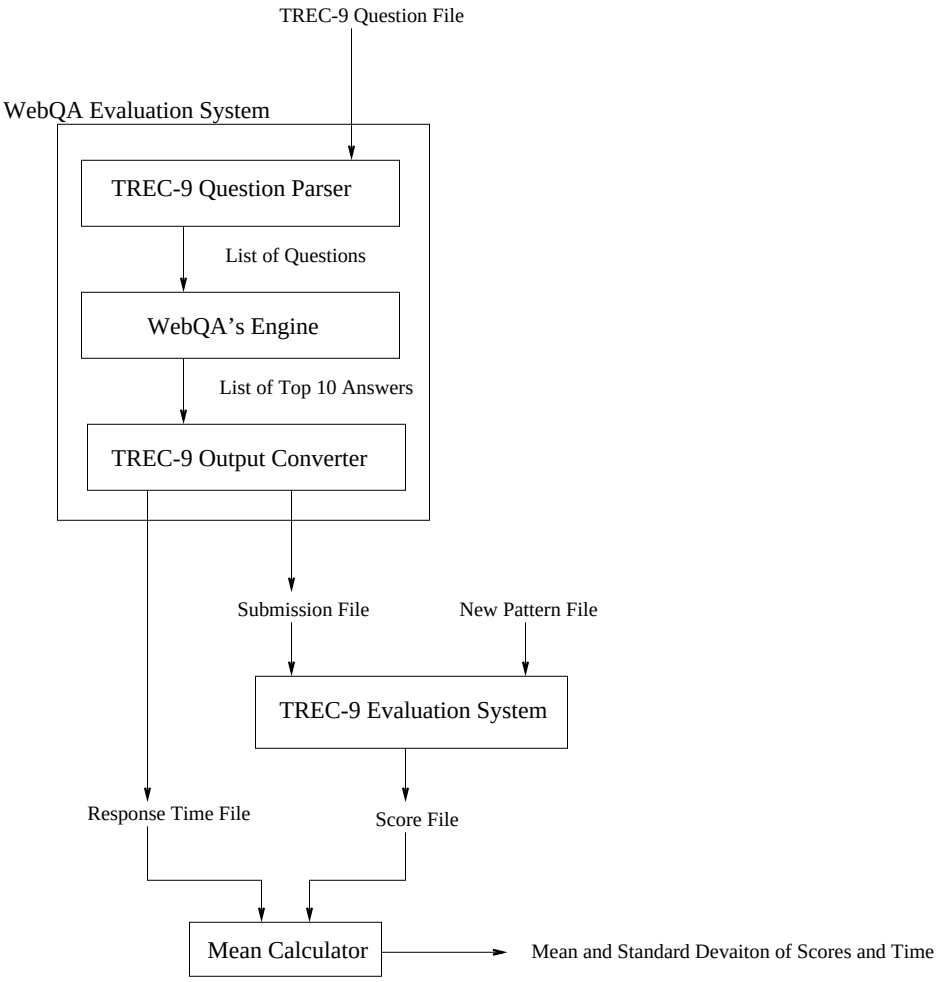

Figure 7.1: The architecture for evaluation

WebQA's engine. Of course, the engine produces a list of answers for each question. Later on, the list of answers is taken by Output Converter which converts them to a TREC-9 submission file. The result is a submission file and a response time file. The response time file is generated by the engine, which keeps track of the local and remote execution times of each question. Finally, the TREC-9 evaluation system takes the submission file and the new pattern file, and generates a score file. The score file and the response time file are submitted to the Mean Calculator, which calculates the mean and standard deviation of the score, local execution time, remote execution time, and response time for each category and the overall. A sample output of the Mean calculator is shown in Figure 7.2. The following subsections show the commands for running the evaluation and the format of all files.

7.3.1 Commands

The command for running the WebQA evaluation system

We have implemented the WebQA evaluation system using a Java class called `WebQAEvaluation`. The command for executing the `WebQAEvaluation` class is

> `java -cp` ⟨*class path*⟩ `ca.uwaterloo.db.webqa.evaluation.WebQAEvaluation`
> ⟨*question file*⟩ ⟨*output file*⟩ n ⟨*source ranking file*⟩ ⟨*off-line record file*⟩
> c t

where ⟨*class path*⟩ indicates the path where the WebQA's Java class files are located; ⟨*output file*⟩ is the output file name without the file extension; n is the number of

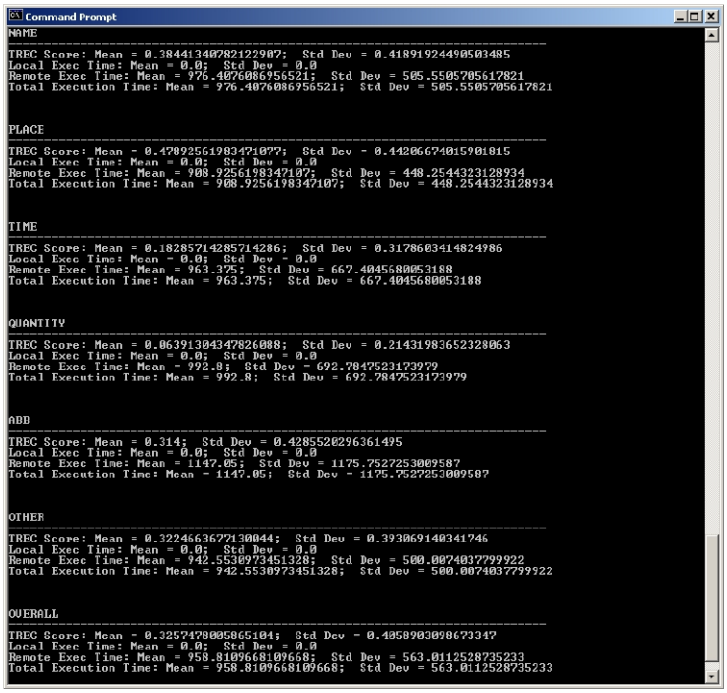

Figure 7.2: An sample output of the Mean calculator

concurrent pages; ⟨*source ranking file*⟩ is the file that stores the source ranking; ⟨*off-line record file*⟩ is the off-line record file, *null* means to retrieve records online; c is the cache size ($c \leq 0$ means no cache available); and t is the test query size (meaningful if $c > 0$).

The command for running the TREC-9 evaluation system

TREC-9 evaluation system is a Perl program that is given by the TREC-9. The command for executing the evaluation system is as follows:

`perl pp_eval.pl` ⟨*pattern file*⟩ ⟨*submission file*⟩ `>` ⟨*score file*⟩

where ⟨*pattern file*⟩ is the pattern file; ⟨*submission file*⟩ is the submission file that is generated by the WebQA evaluation system; ⟨*score file*⟩ is the score file that is going to submit to the Mean Calculator.

The command for running the Mean Calculator

A Java class called `TRECMean` has been implemented for the Mean Calculator. Here is the command for executing the `TRECMean` class:

`java -cp` ⟨*class path*⟩ `ca.uwaterloo.db.webqa.evaluation.TRECMean`
⟨*score file*⟩ ⟨*response time file*⟩

where ⟨*class path*⟩ indicates the path where the WebQA's Java class files are located; ⟨*score file*⟩ is the score file from the TREC-9 evaluation system; and ⟨*response time file*⟩ is the response time file from the WebQA evaluation system.

7.3.2 File Formats

Submission File

The submission file has a file extension, "`.sub`". Each line in the submission file represents an answer for a TREC-9 question. We choose 5 as the number of returned answers; therefore, there are at most five lines in the submission file for each TREC-9 question. The format of each line in the file is as follows:

⟨*question number*⟩ `Q0 X` i $(n + i - 1)$ `X` ⟨ith *Answer*⟩

where ⟨*question number*⟩ represents the TREC-9 question number; i represents the rank of the answer; n represents the number of returned answer, which is 5; and ⟨ith *Answer*⟩ is the ith answer. Here is an example of a submission file for 3 TREC-9 questions 201 to 203:

```
201 Q0 X 1 5 X Space
201 Q0 X 2 4 X News
201 Q0 X 3 3 X American
201 Q0 X 4 2 X CANAVERAL
201 Q0 X 5 1 X Web
202 Q0 X 1 5 X Caye
202 Q0 X 2 4 X Caribbean
202 Q0 X 3 3 X Ignacio
202 Q0 X 4 2 X Central
202 Q0 X 5 1 X St
203 Q0 X 1 1 X null
```

Response Time File

The response time file has an extension, "`.time`". It stores a list of local execution time and remote execution time for each question. Each line in the response time file as the following format:

⟨*question number*⟩ ⟨*local execution time*⟩ ⟨*remote execution time*⟩

where ⟨*question number*⟩ represents the TREC-9 question number; ⟨*local execution time*⟩ and ⟨*remote execution time*⟩ represents the local and remote execution time for the question, respectively. Here is an example of a score file for 3 TREC-9 questions 201 to 203:

```
201 0 1382
202 0 451
203 0 1442
```

Score File

The score file has an extension, "`.score`". Each line in the score file represents the result of each question. If no correct answer can be found in the submission file, then the line has the following format:

`Question` ⟨*question number*⟩ `: No correct answer found.`

where ⟨*question number*⟩ represents the TREC-9 question number. If one of the answers in the submission file matches the correct answer, then the line has the following format:

`Question` ⟨*question number*⟩ `Correct answer found at rank` i `(`$1/i$`).`

where ⟨*question number*⟩ represents the TREC-9 question number; and i is the rank of the answer that matches the correct answer. Here is an example of a score file for 3 TREC-9 questions 201 to 203:

```
Question 201:  No correct answer found.
Question 202:  Correct answer found at rank 1 (1.00).
Question 203:  No correct answer found.
```

7.4 Experimental Results

We have setup a number of experiments for evaluating WebQA's accuracy and efficiency. The following sections explain each experiment and its outcome. All the experiments are run on a Pentium III 800 machine with 256 MB RAM running Windows 2000. They use an interpreter of Sun Microsystems' Java 2 SDK Standard Edition Version 1.3.1.

7.4.1 Experiment 1

Motivation

To see the performance for categorizing questions (i.e., Categorizer subcomponent).

Methodology

First, we read all the TREC-9 and TREC-10 [1] questions and determine the correct category for each question by hand. Then, we submit the same set of questions to the Categorizer subcomponent. Finally, we check how many matches and mismatches in the questions.

A Java class has been implemented to perform the test. After compilation, we use the following command to run the class.

```
java -cp ⟨class path⟩ ca.uwaterloo.db.webqa.evaluation.
CategorizerEvaluation t9.tqf
```

[1] TREC-10 is a successor evaluation system to TREC-9. Even through we performed most of the experiments using TREC-9 questions, we tested the categorizer using TREC-10 as well as TREC-9.

where ⟨*class path*⟩ indicates the path where the WebQA's Java class files are located; `t9.tqf` is the question file that stores the TREC-9 questions.

Result

As for TREC-9 questions, the total number of questions is 693. The number of matches is 686. Therefore, the accuracy is **98.99%**.

Since we have examined a subset of TREC-9 questions to build the Categorizer subcomponent, we decided to use another set of questions (i.e. TREC-10 questions) for evaluating the categorization accuracy. The total number of TREC-10 questions is 500. The number of matches is 461. In other words, the accuracy is **92.2%**.

7.4.2 Experiment 2

Motivation

To determine the best source ranking for each category.

Methodology

We use the experimental procedure that is described in Section 7.3. The difference is that we are using only one search engine as a data source for retrieving 50 records. As mentioned in Section 5.4.2, Google is not scalable due to 1000 queries per day for querying. Therefore, we do not choose Google as our Web source for retrieving records. Fortunately, Yahoo's search engine is powered by Google with minor differences [Yah02a]. Therefore, their results are very similar.

Result

Figures 7.3 to 7.8 show the TREC-9 scores for each category. From these figures, we can determine which search engines perform well for each category. From these figures, we conclude the search engines that we use for retrieving records as shown in Table 7.2.

Table 7.2: The search engines that WebQA uses as data source for each category

Category	Rank	Sources	Number of Records Needed
Name	1	Yahoo	50
Place	1	Yahoo	50
	2	Excite	50
	3	Overture	50
Time	1	Yahoo	50
	2	All The Web	50
Quantity	1	Yahoo	50
Abbreviation	1	Yahoo	50
	2	All The Web	50
Other	1	Yahoo	50

7.4.3 Experiment 3

Motivation

To see differences in terms of accuracy and efficiency between using and not using a secondary source for retrieving records.

Methodology

Table 7.2 shows the source ranking that does not use a secondary source. We would like to know what happens to the accuracy and the efficiency if we add a secondary

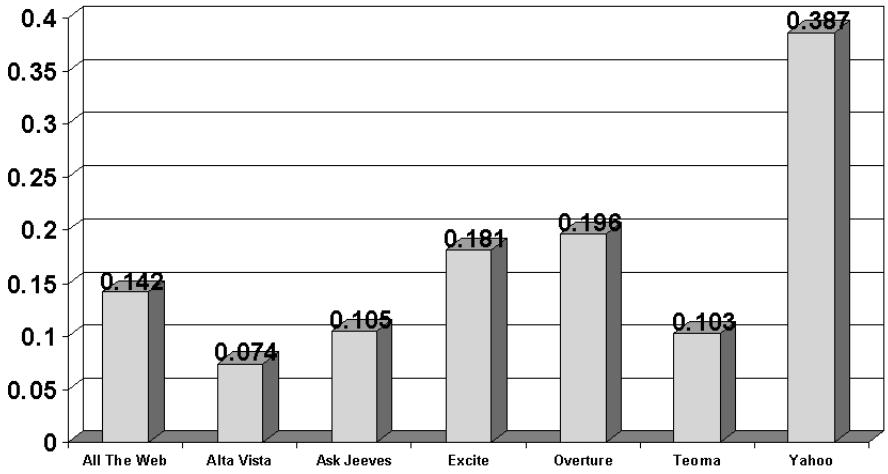

Figure 7.3: Experiment 2: TREC-9 scores for Name category

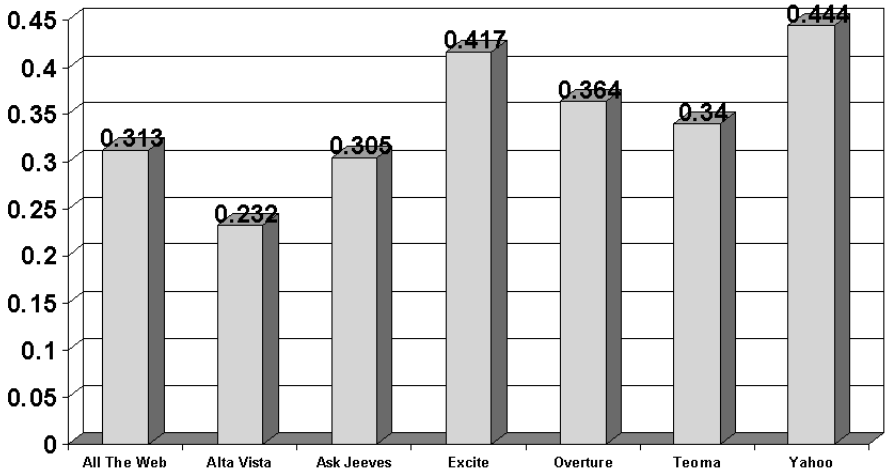

Figure 7.4: Experiment 2: TREC-9 scores for Place category

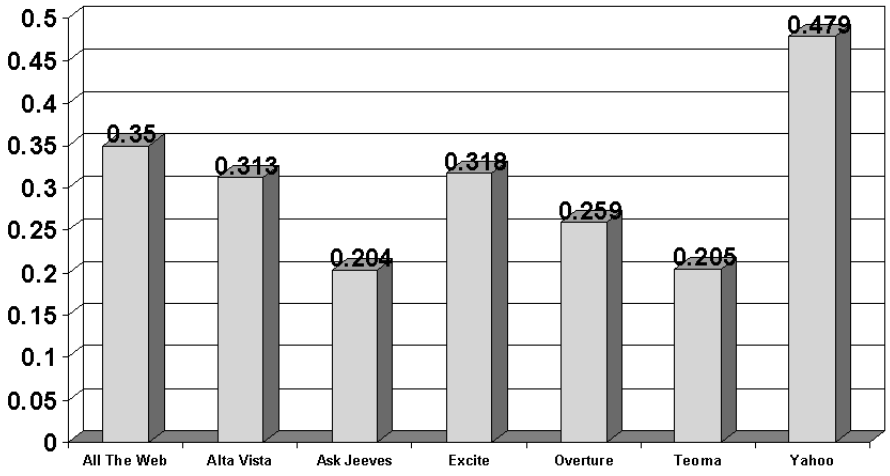

Figure 7.5: Experiment 2: TREC-9 scores for Time category

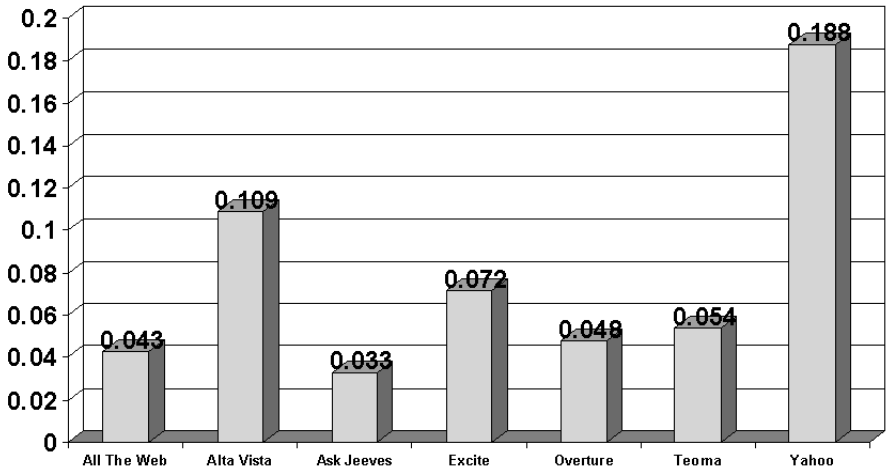

Figure 7.6: Experiment 2: TREC-9 scores for Quantity category

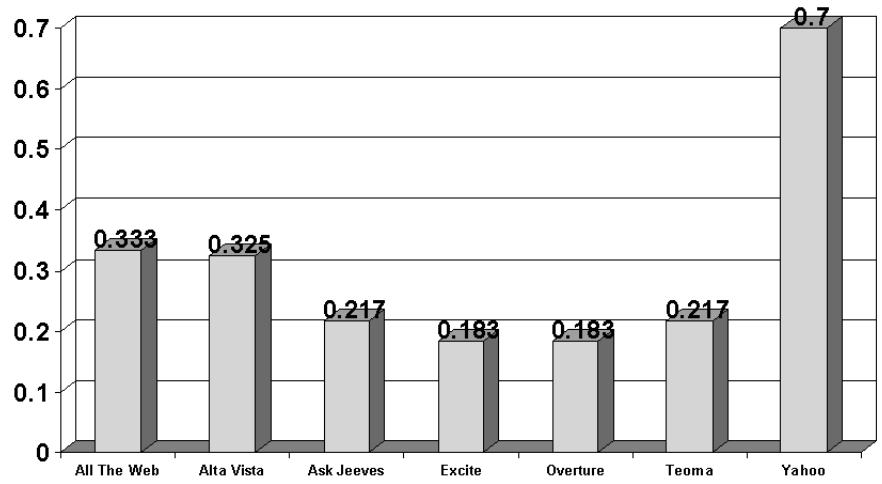

Figure 7.7: Experiment 2: TREC-9 scores for Abbreviation category

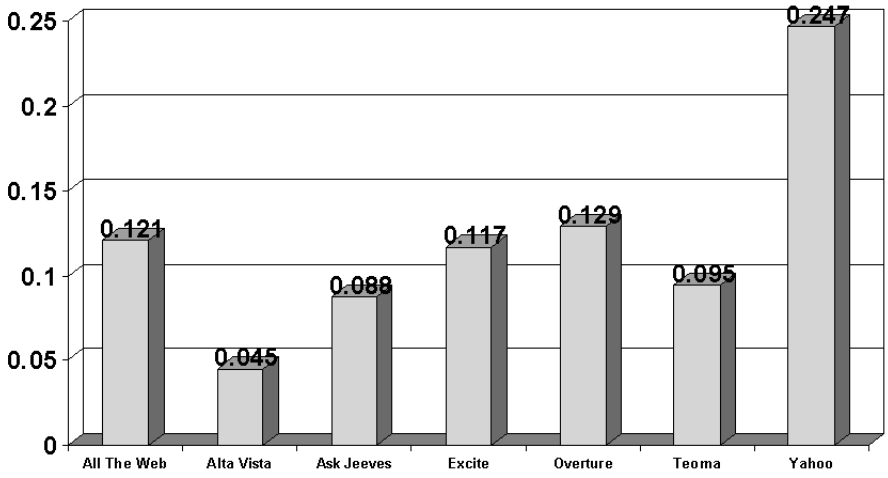

Figure 7.8: Experiment 2: TREC-9 scores for Other category

source such as World Factbook to the source ranking as shown in Table 7.3.

Table 7.3: The search engines that WebQA uses as data source for each category

Category	Rank	Sources	Number of Records Needed
Name	1	Yahoo	50
Place	1	World Factbook	5
	2	Yahoo	50
	3	Excite	50
	4	Overture	50
Time	1	World Factbook	5
	2	Yahoo	50
	3	All The Web	50
Quantity	1	World Factbook	5
	2	Yahoo	50
Abbreviation	1	World Factbook	5
	2	Yahoo	50
	3	All The Web	50
Other	1	World Factbook	5
	2	Yahoo	50

Result

Table 7.4 compares the TREC-9 score for WebQA with and without the secondary source (World Factbook). If WebQA uses the secondary sources, then the source ranking is described in Table 7.3. Otherwise, the source ranking is described in Table 7.2. In addition, it shows the response time for using and not using the secondary source. As we see in these figures, the accuracy increases by 0.09 if we use the secondary source. However, the response time increases by 10 milliseconds, which is negligible.

Figure 7.9 shows the TREC-9 score of WebQA that uses the secondary source for each category. WebQA performs reasonably well in the *Name, Place,* and *Time* categories with scores over 0.3. It performs extremely well in the *Abbreviation*

Table 7.4: Experiment 3: TREC-9 Scores of WebQA using and not using the secondary source

	With Secondary Source	Without Secondary Source
TREC-9 Score	0.35	0.359
Response Time	2981	2991

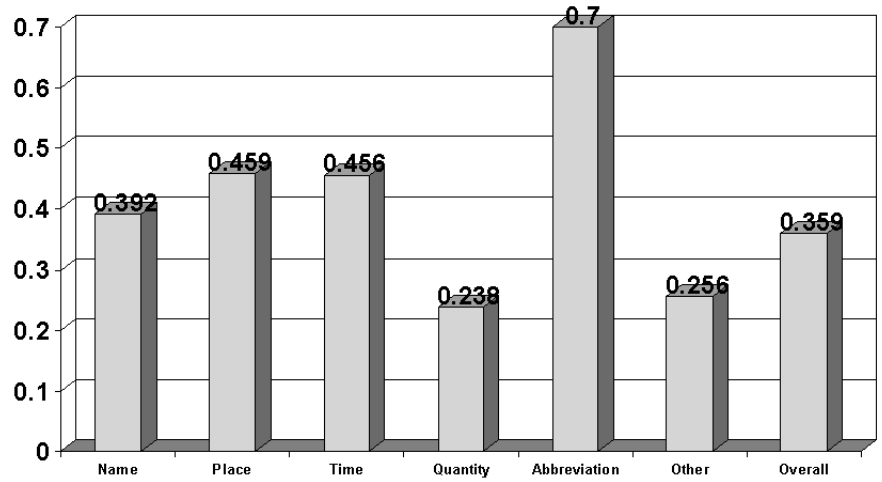

Figure 7.9: Experiment 3: TREC-9 scores for each category in WebQA

category with a score of 0.7. Overall, it has a score of 0.359. The TREC-9 score for each question can be found in Appendix F.

Figure 7.10 shows the response time of WebQA for each category. All categories require less than 10 seconds for processing. As we can see, the local execution time takes less than one second. The bottleneck is the waiting time for the data sources. The *Abbreviation* category takes the longest, because it spends time waiting for the abbreviation page of World Factbook Web site. On average, WebQA takes around 3 seconds to process a query.

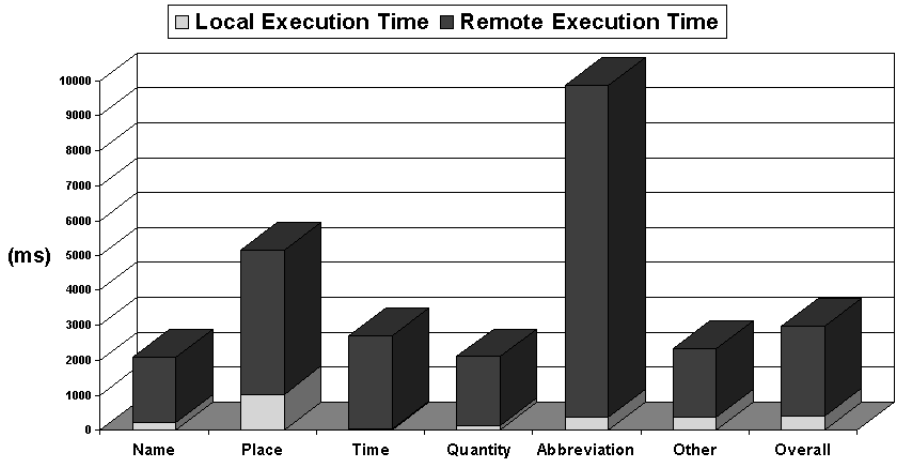

Figure 7.10: Experiment 3: Response time for each category in WebQA

7.4.4 Experiment 4

Motivation

Compare WebQA with other query systems.

Methodology

For each search engine, we submit each TREC-9 question (without eliminating stopwords) to the search engine. We extract the top five snippets returned by the search engine. The top five snippets are submitted to TREC-9 evaluation system. The system generates a TREC-9 score for the search engine. Furthermore, we keep track of the end user latency for the search engine. The end user latency includes the network latency between the search engine's server and our machine.

A Java class called `SearchEngineEvaluation` has been implemented for running this experiment. The command to run the class is as follows:

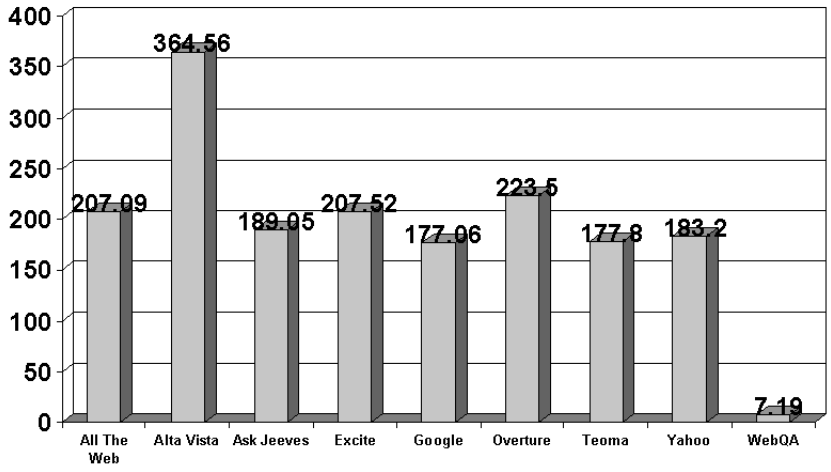

Figure 7.11: Experiment 4: Average number of characters

`java -cp` ⟨*class path*⟩ `ca.uwaterloo.db.webqa.evaluation.SearchEngineEvaluation t9.tqf` ⟨*search engine*⟩

where ⟨*class path*⟩ indicates the path where the WebQA's Java class files are located; `t9.tqf` is the question file that stores the TREC-9 questions; ⟨*search engine*⟩ is the name of the tested search engine.

Finally, we can compare the search engine result with the result in Experiment 3.

Result

Figure 7.11 shows the average number of characters for each search engine and WebQA. As we see, WebQA returns around 9 characters, which is a lot shorter than all other search engines. In other words, users spend shorter time on reading.

Figures 7.12 to 7.17 compare WebQA's TREC-9 score with other search engines

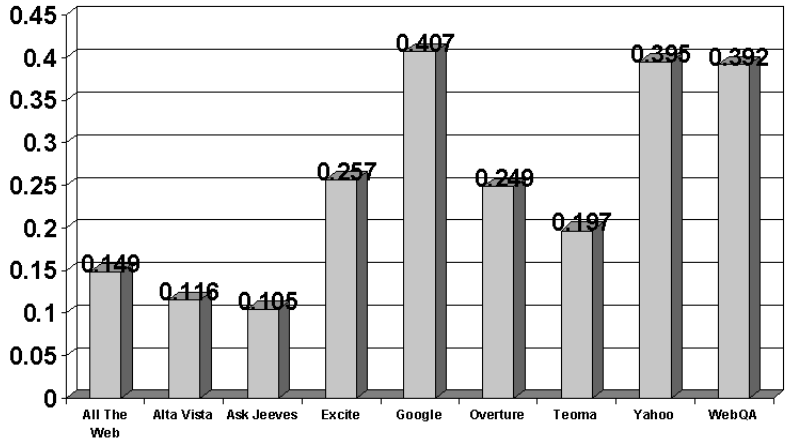

Figure 7.12: Experiment 4: Compare WebQA's TREC-9 score with other search engines for the Name category

for each category. Figure 7.18 compares WebQA's TREC-9 score with other search engines over all the categories, where WebQA has the best accuracy overall.

Figure 7.19 compares the response time of WebQA with other search engines. Since WebQA uses search engines as data sources, there is no way that WebQA can perform faster than the search engines that it uses for data sources. However, WebQA only takes few seconds to process a query.

Furthermore, WebQA is ranked second when we compare it with traditional QA systems in Table 7.1. We should keep in mind that there are differences on how we and traditional QA systems use TREC-9 for evaluation as mentioned in Section 7.3. We cannot exactly follow the same procedure as the traditional QA systems, which is mentioned in Section 7.1.1.

In addition, there are no published results about the efficiency of other systems. Therefore, we can not compare WebQA's response time with other QA systems.

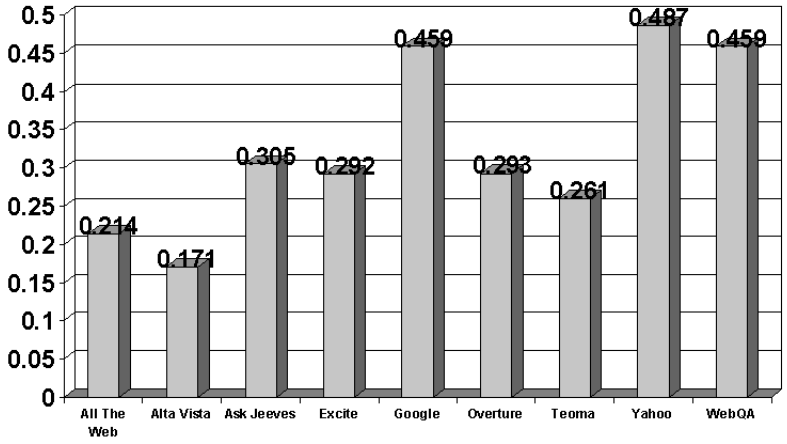

Figure 7.13: Experiment 4: Compare WebQA's TREC-9 score with other search engines for the Place category

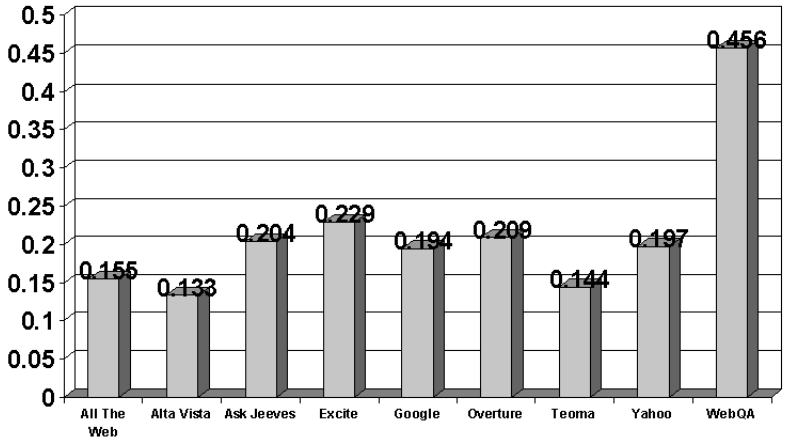

Figure 7.14: Experiment 4: Compare WebQA's TREC-9 score with other search engines for the Time category

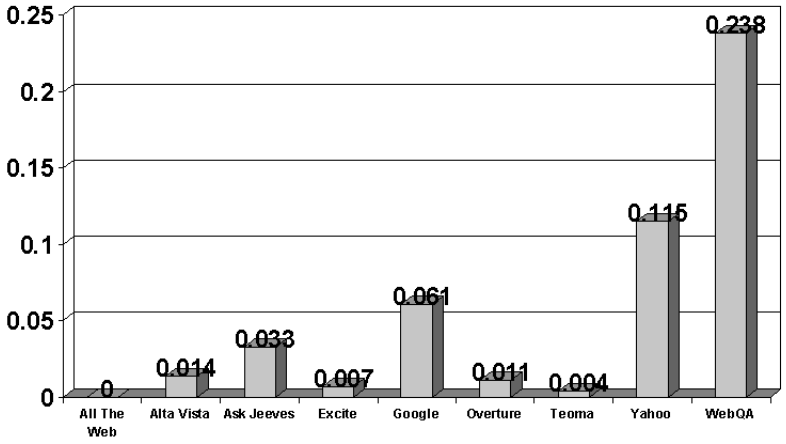

Figure 7.15: Experiment 4: Compare WebQA's TREC-9 score with other search engines for the Quantity category

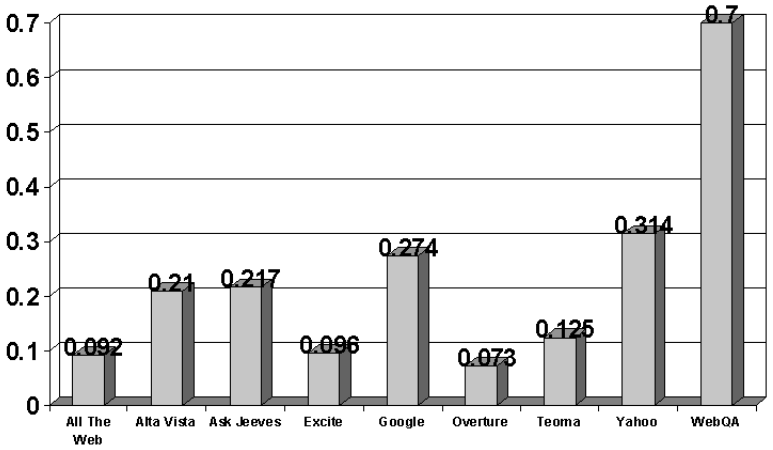

Figure 7.16: Experiment 4: Compare WebQA's TREC-9 score with other search engines for the Abbreviation category

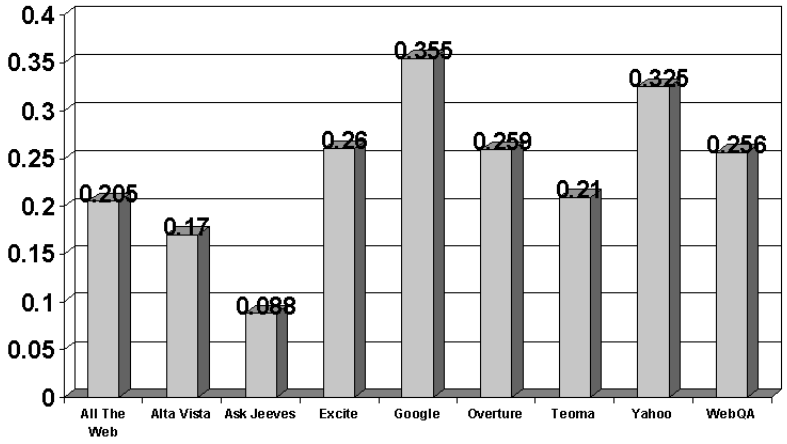

Figure 7.17: Experiment 4: Compare WebQA's TREC-9 score with other search engines for the Other category

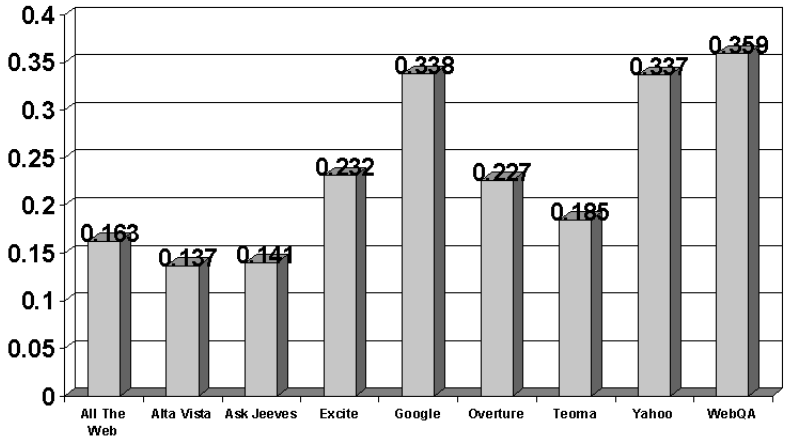

Figure 7.18: Experiment 4: Compare WebQA's TREC-9 score with other search engines for over all the categories

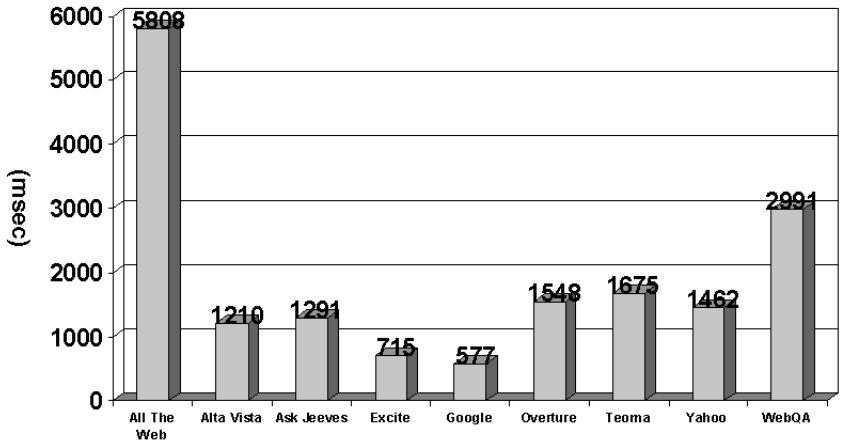

Figure 7.19: Experiment 4: Response time for each search engine and WebQA

7.4.5 Experiment 5

Motivation

Since TREC-9 does not cover the *Weather* questions, we would like to demonstrate WebQA in the *Weather* category by trying out some sample questions.

Methodology

In the textual interface, we input the following three questions:

1. What is the temperature in Toronto, Canada?

2. What is the temperature in Tokyo, Japan?

3. What is the dew point in Shanghai, China?

Result

For Question 1, WebQA produces the following answer:

> 1: City: Toronto, Canada Temperature: 25 F Observed Time: 4:00 PM EST on March 25, 2002 (*)
> 2: City: Toronto Pearson, Canada Temperature: 27 F Observed Time: 4:00 PM EST on March 25, 2002 (*)

The local execution time is 230 milliseconds. The remote execution time is 14761 milliseconds, and the response time is 14991 milliseconds.

For Question 2, WebQA has the following answer:

> 1: City: Tokyo Downtown, Japan Temperature: 46 F Observed Time: 3:00 AM JST on March 26, 2002 (*)

The local execution time is 131 milliseconds. The remote execution time is 15752 milliseconds. And the response time is 15883 milliseconds.

For Question 3, WebQA has the following answer:

> 1: City: Shanghai, China Dew Point: 51 F Observed Time: 2:00 AM CST on March 26, 2002 (*)

The local execution time is 230 milliseconds. The remote execution time is 3415 milliseconds. And the response time is 3645 milliseconds.

7.4.6 Experiment 6

Motivation

To see how the cache affects the response time of processing one question by varying the cache size.

Methodology

We use the best combination as indicated in Table 7.3. For each category, we run the experiment with various cache sizes and number of submitted queries. In order to test the cache, we uniformly pick a question from the TREC-9 question list.

Result

Figure 7.20 shows the performance of cache hit rate by varying the cache size and number of submitted queries. Figure 7.21 shows the response time by varying the cache size and number of submitted queries. As shown in Figure 7.20, as the cache size increases, the cache hit rate increases as well. Once the cache size reaches 693 (i.e., the possible number of different queries), the cache hit rate stabilizes. Because the cache is filled up with all possible number of questions. The higher number of submitted queries, the higher cache hit rate. On the other hand, the higher number of submitted quires, the lower response time. As the cache size increase, the response time decreases.

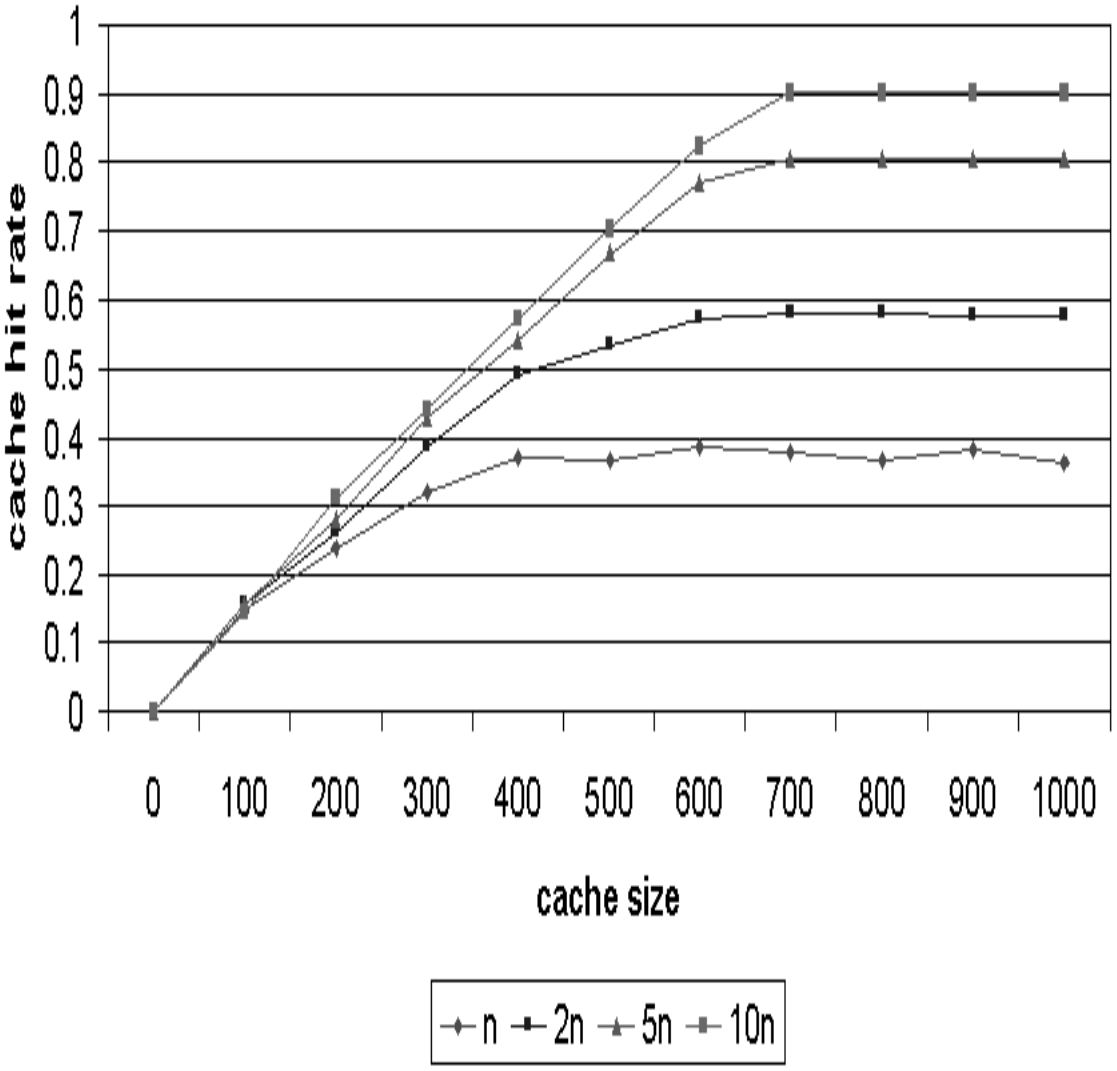

Figure 7.20: Cache hit rate v.s. cache size, where $n = 693$

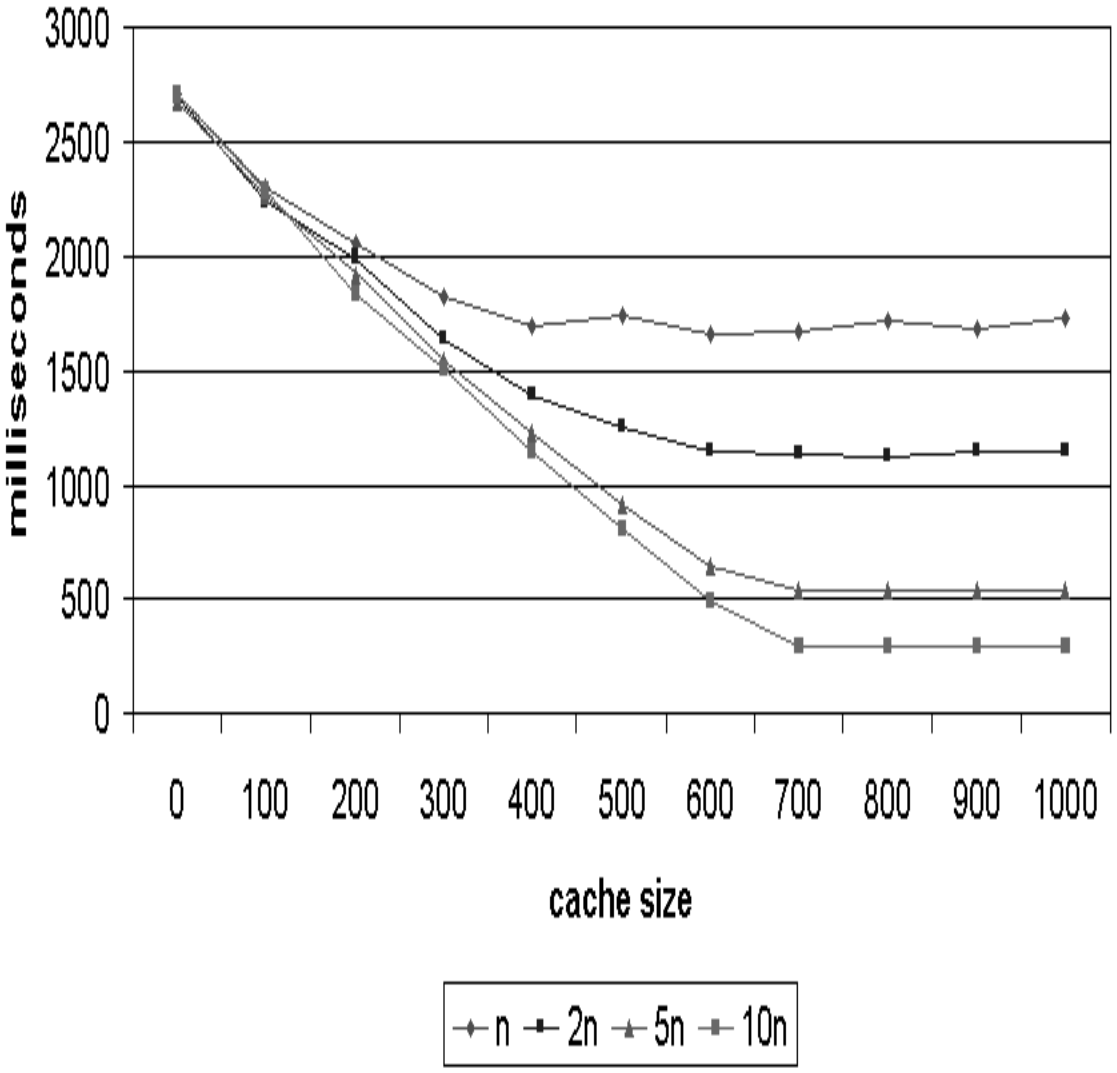

Figure 7.21: Response time v.s. cache size, where $n = 693$

Chapter 8

Conclusion

The thesis addresses difficulties for finding information on the Web. It provides an extensive background by investigating the current Web search approaches. The keyword search (i.e., search engine and metasearcher) approach is currently the most popular way of Web searching. It allows users to submit a keyword expression and return a list of relevant pages. However, the keyword expression can not fully represent the user's intent and the keyword search approach is not able to return the direct answers. The category search approach divides indexed Web pages into categories, but not all Web pages can be classified. The database view approach integrates different structured data in Web sources and combines information into a view format. However, the number of Web sources that are well structured is limited. The semistructured data querying approach uses semistructured data query languages for querying semistructured data, but it requires a technique to transform Web data to semistructured data, which is not efficient. The Web query language approach provides languages for querying the Web. Although, this approach ex-

ploits the content of the Web by searching keywords, but the primary focuses is to exploit the structure of Web documents. The approach does not "understand" the content, so it is not able to return direct answers. The learning-based approach allows users to submit keyword expressions and learns the users' query requirements. There are few problems. The approach returns Web pages, not direct answers. It requires the users to rate Web pages for other users, but the rating can be inaccurate. The question-answering approach allows users to pose natural language questions and returns direct answers. However, very few of them use the Web as data sources and they are limited to factual questions. One exception is Ask Jeeves, but it returns relevant pages, not direct answers.

This thesis introduces a Web query system called WebQA, which follows the question-answering approach. It accepts a short factual natural language question and returns a list of possible direct answers, not a list of hyperlinks. It is an open system with a wrapper/mediator architecture and has the ability to query different Web sources such as search engines, data sources (e.g. CIA's World Factbook) and Web sites (weather sites). It provides an uniform interface for users to query the Web effectively. It provides a solid architecture for future modification. The system has a textual interface that is used internally, and a graphical user interface (a Web page).

The core part of the thesis is the discussion of the WebQA engine. When the engine receives an query from an interface, it passes the query to its Query Parser component. The QP component categorized the query and returned a WebQAL expression. After that, the engine passed the WebQAL expression to its Summary

Retriever component. The SR component retrieves relevant records from Web sources. The relevant records and the WebQAL expression are passed to the Answer Extractor component. The AE component identified all the possible candidates from the relevant records and returns the best candidates as the best answers to the interface.

The QP component takes a query as an input. If the query is a WebQAL expression, then the QP passes it to the WebQAL Checker subcomponent, which confirms that the WebQAL expression is valid. If the query is written in natural language (i.e English), then the QP determines the category by calling the Categorizer subcomponent. The category and the question are both submitted to the WebQAL Generator subcomponent, which generates a WebQAL expression. Finally, the QP returns the WebQAL expression.

The SR component takes the WebQAL expression as an input. It passes the WebQAL expression to the Keyword Generator and Source Ranker subcomponents. The Keyword Generator generates a keyword expression for submitting to the Web sources. The Source Ranker checks the category in the WebQAL expression and returns a source ranking for retrieving records. Both the list of keywords and the source ranking are passed to the Record Consolidator/Ranker subcomponent. The Record Consolidator/Ranker instantiates a record retriever for each source in the source ranking. All record retrievers are running concurrently. Depending on the source, each record retriever retrieves records from the assigned wrapper by calling an interface, Wrapper API. Each wrapper is able to submit the keyword expression to its Web source and convert the result back to record format. Once all record

retrievers finish retrieving records, the Record Consolidator/Ranker consolidates all the records and ranks them using its ranking algorithm. At the end, it returns the ranked records.

The AE component takes the WebQAL expression and the ranked records as inputs. It calls its Candidate Identifier subcomponent for identifying all the possible candidates. Each candidate consists the name (possible answer in string format) and a score. A high score implies that the candidate would likely be the correct answer. Then, the AE passes the list of candidates to the Best Candidate Retriever for retrieving the best n candidates, where n is specified by an user. The list of best candidates is passed to the Rearranger, which rearranges the list of best candidates. At the end, the list of rearranged candidates is passed to the Output Converter, which converts the list of rearranged candidates to the result format.

A number of experiments have been run to test WebQA effectiveness and performance. We have used a list of TREC-9 questions as sample data. Our categorization accuracy is 98.99%. As for accuracy, WebQA has a TREC-9 score of 0.359. The average response time of WebQA is 2991 milliseconds. If we compare WebQA with other search engines, WebQA returns, on the average a smaller number of characters in an answer, has a higher TREC-9 score, but has a longer response time. If we compare WebQA with traditional QA systems, WebQA is ranked second out of other 22 systems. With these significant results, WebQA provides a more effective way for searching Web data.

8.1 Differences between Mulder and WebQA

The only other system in literature that is sufficiently similar to WebQA is Mulder. Both systems accept short factual NL questions as inputs and both returns the exact answers as outputs. Both systems use search engines as their knowledge base. And both systems have similar main components which is common in traditional QA systems. However, there are major differences within each component of the systems. For query parsing, Mulder requires heavy NL parsing and WebQA takes a light weight approach. Mulder classifies questions to only 3 categories while WebQA has 7 categories (with the ability to increase more in the future). With more specific categories, we can increase accuracy by extracting answers with specific type. For example, WebQA has place category. If a user asks *"Which states has the most population?"*, WebQA verifies that the returned answer must be one of a valid state. Another issue we need to be concerned with is the accuracy of question classification. We have tried to compare our categorizer with Mulder's. The accuracy of our categorizer is close to 100% and we have more categories. Unfortunately, [KEW01] does not state the performance of the question classification. As for passage retrieval, Mulder only relies on a single search engine. But WebQA searches multiple search engines for efficient and effective reasons. If one search engine fails, there are still results from other search engines. This gives WebQA fault-tolerance that is not available in Mulder. In addition, WebQA also queries relevant information sources that might contain answers to the question. Generally, these sources are more accurate than search engines.

8.2 Future Work

WebQA provides a framework that enables users to submit short factorial questions and returns lists of possible answers. It provides many future opportunities in research and development, so others can extend it to make WebQA more powerful. A true QA system should be able to answer any type of questions including procedural questions (e.g., "How to make a cheese cake?"). One future improvement is to allow WebQA to handle other types of question.

At this stage, WebQA can handle only English queries and data that are written in English. A future direction is to internationalize WebQA, so that it can handle other written languages, such as Chinese and Japanese.

Natural language questions introduce ambiguity, a better way to propose a query is to use a standard query language. A challenge future work is to design and implement the standard query language for WebQA. Therefore, WebQA will handle complex query without using heavy natural language processing.

TREC-9 and other QA evaluation systems are designed for evaluating QA systems that search off-line data. One of major problems is that there is no evaluation system that specifically evaluate QA systems that search online data (i.e., the Web). The new evaluation system should be able to measure online QA systems' accuracy and efficiency. Since the data on the Web changes dynamically, the evaluation system should handle up-to-date answers.

Each wrapper of WebQA is written manually. A possible future work is to build an automatic wrapper generator that generates a wrapper given a Web site. This way, we can save a lot of development time.

Furthermore, we can add more Web sources for WebQA, so it can cover more resources and answer more queries.

There are only 7 categories in WebQA. We can add more categories such as stock quote, so we don't have to use the *Other* category. This way, it would provide higher accuracy.

Appendix C lists words that can identify each category. A future work is to add more words to all the term lists to improve the question categorization.

Finally, we can improve our simple source and record ranking algorithms for improvement of WebQA's accuracy.

Appendix A

Sample Outputs

Here is a list of sample outputs that WebQA returns correct answers in the first rank. The number in front of each answer is the ranking of the answer. This appendix provides an idea of a sample question and a sample answer look like. It does not serve the purpose of demonstrating the WebQA's accuracy.

Q: Who is the inventor of telephone?
A: 1. Alexander Graham Bell

Q: Who invented electricity?
A: 1. Thomas Edison

Q: Who invented the paper clip?
A: 1. Johan Vaaler

Q: Who is the chairman of Microsoft?
A: 1. Bill Gates

Q: What is the capital city of Canada?
A: 1. Ottawa

Q: What is the capital city of United States?
A: 1. Washington, DC

Q: What is the capital city of China?
A: 1. Beijing

Q: What is the capital city of France?
A: 1. Paris

Q: What river is the longest river in the World?
A: 1. Nile

Q: What is the primary language of Philippines? (TREC-9)
A: 1. English

Q: Who is the Greek God of the Sea? (TREC-9)
A: 1. Poseidon

Q: What is IBM stands for?
A: 1. International Business Machines

Q: What is ABED stands for?
A: 1. Arab Bank for Economic Development in Africa

Q: What is the temperature in Ottawa, Canada?
A: 1. City: Ottawa, Canada
Temperature: 63 F
Observed Time: 03:00 PM EDT on August 28, 2001

Appendix B

Stoplist

Here is a list of stoplist that is used by WebQA:

a	about	all
also	an	and
another	are	as
at	be	because
been	being	by
could	did	do
does	each	for
from	further	had
has	have	having
he	her	his
how	i	if
in	into	is
it	its	many
me	much	of
on	onto	or
other	our	said
she	should	since
so	some	such
than	that	the
their	them	then
there	thereby	therefore
thereof	thereto	these
they	this	those
thus	to	untitled
use	was	we
were	what	when
where	which	while

who	will	with
would	you	

Appendix C

Term List

The followings shows a list of words that can identify for each category. These lists can be expanded for a future work. Note that there are no words for identifying Other category.

C.1 Name Term List

actor	actors	actress
actresses	astronaut	astronauts
author	authors	building
buildings	CEO	CEOs
chairman	chairmen	companies
company	counterpart	counterparts
creator	creators	director
directors	drink	drinks
god	gods	king
kings	mascot	mascots
monarch	monarchs	model
models	movie	movies
name	named	names
newspaper	newspapers	nickname
nicknamed	nicknames	park
parks	president	presidents
queen	queens	scientist
scientists	skater	skaters
soda	sodas	speaker
speakers	star	stars

C.2 Place Term List

birthplace	birthplaces	capital
capitals	cities	city

college
colleges
continent
continents
countries
country
location
locations
nationalities
nationality
ocean
oceans
place
places
province
provinces
state
states
territories
territory
universities
university

C.3 Time Term List

birthday
birthday
birthdays
date
dates
day
days
month
months
year
years

C.4 Quantity Term List

amount
amounts
income
incomes
number
numbers
period
periods
population
populations
salaries
salary
speed
speeds
wage
wages

C.5 Abbreviation Term List

abbreviate
abbreviates
abbreviated
abbreviation
acronym

C.6 Weather Term List

barometer
barometers
conditions
dewpoint
dewpoints
humidities
humidity
moonrise
moonset
sunrise
sunrises
sunset
sunsets
temperature
temperatures
weather
weathers
wind
winds
visibilities
visibility

C.7 How-quantity Term List

big
close
cold
cool
far
fast
high
hot
large
long
many
much
old
short
slow
small
tall
wide
young

Appendix D

Verb-to-noun Conversion Table

The followings shows a verb-to-noun conversion table.

create	creator
created	creator
creates	creator
invent	inventor
invented	inventor
invents	inventor
locate	location
located	location
locates	location
own	owner
owned	owner
owns	owner
sang	singer
skate	skater
skated	skater
skates	skater
sing	singer
sings	singer
write	author
writes	author
wrote	author

Appendix E

TREC-9 Question List

Here is a list of TREC-9 questions with question IDs. It is not written in TREC-9 question file format. Note that the question ID starts with 201

201. What was the name of the first Russian astronaut to do a spacewalk?
202. Where is Belize located?
203. How much folic acid should an expectant mother get daily?
204. What type of bridge is the Golden Gate Bridge?
205. What is the population of the Bahamas?
206. How far away is the moon?
207. What is Francis Scott Key best known for?
208. What state has the most Indians?
209. Who invented the paper clip?
210. How many dogs pull a sled in the Iditarod?
211. Where did bocci originate?
212. Who invented the electric guitar?
213. Name a flying mammal.
214. How many hexagons are on a soccer ball?
215. Who is the leader of India?
216. What is the primary language of the Philippines?
217. What is the habitat of the chickadee?
218. Who was Whitcomb Judson?
219. What is the population of Japan?
220. Who is the prime minister of Australia?
221. Who killed Martin Luther King?
222. Who is Anubis?
223. Where's Montenegro?
224. What does laser stand for?
225. Who is the Greek God of the Sea?
226. Where is the Danube?
227. Where does dew come from?
228. What is platinum?
229. Who is the fastest swimmer in the world?
230. When did the vesuvius last erupt?

231. Who was the president of Vichy France?
232. Who invented television?
233. Who made the first airplane?
234. Who made the first airplane that could fly?
235. How many astronauts have been on the moon?
236. Who is Coronado?
237. Name one of the major gods of Hinduism?
238. What does the abbreviation OAS stand for?
239. Who is Barbara Jordan?
240. How many years ago did the ship Titanic sink?
241. What is a caldera?
242. What was the name of the famous battle in 1836 between Texas and Mexico?
243. Where did the ukulele originate?
244. Who invented baseball?
245. Where can you find the Venus flytrap?
246. What did Vasco da Gama discover?
247. Who won the Battle of Gettysburg?
248. What is the largest snake in the world?
249. Where is the Valley of the Kings?
250. Where did the Maya people live?
251. How many people live in Chile?
252. When was the first flush toilet invented?
253. Who is William Wordsworth?
254. What is California's state bird?
255. Who thought of teaching people to tie their shoe laces?
256. Who is buried in the great pyramid of Giza?
257. What do penguins eat?
258. Where do lobsters like to live?
259. What are birds descendents of?
260. What does NAFTA stand for?
261. What company sells the most greeting cards?
262. What is the name of the longest ruling dynasty of Japan?
263. When was Babe Ruth born?
264. Who wrote the Farmer's Almanac?
265. What's the farthest planet from the sun?
266. Where was Pythagoras born?
267. What is the name for clouds that produce rain?
268. Who killed Caesar?
269. Who was Picasso?
270. Where is the Orinoco?
271. How tall is the giraffe?
272. Where are there aborigines?
273. Who was the first U.S. president ever to resign?
274. Who invented the game Scrabble?
275. About how many soldiers died in World War II?
276. How much money does the Sultan of Brunei have?
277. How large is Missouri's population?
278. What was the death toll at the eruption of Mount Pinatubo?

279. Who was Lacan?
280. What's the tallest building in New York City?
281. When did Geraldine Ferraro run for vice president?
282. What do ladybugs eat?
283. Where is Ayer's rock?
284. What is the life expectancy of an elephant?
285. When was the first railroad from the east coast to the west coast completed?
286. What is the nickname of Pennsylvania?
287. Who is Desmond Tutu?
288. How fast can a Corvette go?
289. What are John C. Calhoun and Henry Clay known as?
290. When was Hurricane Hugo?
291. When did the Carolingian period begin?
292. How big is Australia?
293. Who found Hawaii?
294. Who is the richest person in the world?
295. How many films did Ingmar Bergman make?
296. What is the federal minimum wage?
297. What did brontosauruses eat?
298. What is California's state tree?
299. How many types of lemurs are there?
300. What is leukemia?
301. Who was the first coach of the Cleveland Browns?
302. How many people die from snakebite poisoning in the U.S. per year?
303. Who is the prophet of the religion of Islam?
304. Where is Tornado Alley?
305. What is molybdenum?
306. Where do hyenas live?
307. Who is Peter Weir?
308. How many home runs did Babe Ruth hit in his lifetime?
309. Who was Buffalo Bill?
310. Where is the bridge over the river Kwai?
311. How many Superbowls have the 49ers won?
312. Who was the architect of Central Park?
313. Who invented paper?
314. What is Alice Cooper's real name?
315. Why can't ostriches fly?
316. Name a tiger that is extinct?
317. Where is Guam?
318. Where did Bill Gates go to college?
319. How many continents are there?
320. Where is Romania located?
321. When was the De Beers company founded?
322. Who was the first king of England?
323. Who is the richest woman in the world?
324. What is California's capital?
325. What is the size of Argentina?
326. What do manatees eat?

327. When was the San Francisco fire?
328. What was the man's name who was killed in a duel with Aaron Burr?
329. What is the population of Mexico?
330. When was the slinky invented?
331. How hot is the core of the earth?
332. How long would it take to get from Earth to Mars?
333. What is the name of the second space shuttle?
334. How old is the sun?
335. What is the wingspan of a condor?
336. When was Microsoft established?
337. What's the average salary of a professional baseball player?
338. Who invented basketball?
339. What was the ball game of ancient Mayans called?
340. Who is Zebulon Pike?
341. How wide is the Atlantic Ocean?
342. What effect does a prism have on light?
343. What's the longest river in the world?
344. Who was considered to be the father of psychology?
345. What is the population of Kansas?
346. Who is Langston Hughes?
347. Who was Monet?
348. Who built the first pyramid?
349. What is the best-selling book of all time?
350. How many Stradivarius violins were ever made?
351. Who is Charles Lindbergh?
352. Who invented the game bowling?
353. The numbering system we use today was introduced to the western world by what culture?
354. What is a nematode?
355. What is the most expensive car in the world?
356. Where does chocolate come from?
357. What state in the United States covers the largest area?
358. What is a meerkat?
359. Where is Melbourne?
360. How much in miles is a ten K run?
361. How hot does the inside of an active volcano get?
362. What is the capital of Burkina Faso?
363. What is the capital of Haiti?
364. How many people lived in Nebraska in the mid 1980s?
365. What is the population of Mozambique?
366. Who won the Superbowl in 1982?
367. What is Martin Luther King Jr.'s real birthday?
368. Where is Trinidad?
369. Where did the Inuits live?
370. What is the most common cancer?
371. A corgi is a kind of what?
372. When was the Triangle Shirtwaist fire?
373. Where is the Kalahari desert?

374. What is porphyria?
375. What ocean did the Titanic sink in?
376. Who was the 33rd president of the United States?
377. At what speed does the Earth revolve around the sun?
378. Who is the emperor of Japan?
379. How big is our galaxy in diameter?
380. What language is mostly spoken in Brazil?
381. Who assassinated President McKinley?
382. When did Muhammad live?
383. What is the largest variety of cactus?
384. Who invented the radio?
385. Where are zebras most likely found?
386. What is anorexia nervosa?
387. What year did Montana become a state?
388. What were the names of the three ships used by Columbus?
389. Who was the 21st U.S. President?
390. Where was John Adams born?
391. Who painted Olympia?
392. Who was Quetzalcoatl?
393. Where is your corpus callosum?
394. What is the longest word in the English language?
395. What is saltpeter?
396. Who invented silly putty?
397. When was the Brandenburg Gate in Berlin built?
398. When is Boxing Day?
399. What is the exchange rate between England and the U.S.?
400. What is the name of the Jewish alphabet?
401. Who was Maria Theresa?
402. What nationality was Jackson Pollock?
403. Tell me what city the Kentucky Horse Park is near?
404. What is the state nickname of Mississippi?
405. Who used to make cars with rotary engines?
406. What is the tallest mountain?
407. What is Black Hills, South Dakota most famous for?
408. What kind of animal was Winnie the Pooh?
409. What's another name for aspartame?
410. What does hazmat stand for?
411. What tourist attractions are there in Reims?
412. Name a film in which Jude Law acted.
413. Where are the U.S. headquarters for Procter & Gamble?
414. What's the formal name for Lou Gehrig's disease?
415. What does CNN stand for?
416. When was CNN's first broadcast?
417. Who owns CNN?
418. What is the name of a Salt Lake City newspaper?
419. Who was Jane Goodall?
420. What is pandoro?
421. What is thalassemia?

422. When did Princess Diana and Prince Charles get married?
423. What soft drink contains the largest amount of caffeine?
424. What do you call a group of geese?
425. How many months does a normal human pregnancy last?
426. What format was VHS's main competition?
427. What culture developed the idea of potlatch?
428. Where is Logan International located?
429. What university was Woodrow Wilson President of?
430. Where is Basque country located?
431. What does CPR stand for?
432. What state produces the best lobster to eat?
433. Who was Darth Vader's son?
434. What is a nanometer?
435. How did Bob Marley die?
436. What instrument is Ray Charles best known for playing?
437. What is Dick Clark's birthday?
438. What is titanium?
439. What is "Nine Inch Nails"?
440. Where was Poe born?
441. What king was forced to agree to the Magna Carta?
442. What's the name of Pittsburgh's baseball team?
443. What is the chemical formula/name for napalm?
444. Where is the location of the Orange Bowl?
445. When was the last major eruption of Mount St. Helens?
446. What is the abbreviation for Original Equipment Manufacturer?
447. What is anise?
448. Where is Rider College located?
449. What does Nicholas Cage do for a living?
450. What does caliente mean (in English)?
451. Where is McCarren Airport?
452. Who created "The Muppets"?
453. When is Bastille Day?
454. What is the Islamic counterpart to the Red Cross?
455. What is Colin Powell best known for?
456. What is the busiest air travel season?
457. Where is Webster University?
458. What's the name of a golf course in Myrtle Beach?
459. When was John D. Rockefeller born?
460. Name a Gaelic language.
461. Who was the author of the book about computer hackers called "The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage"?
462. What is measured in curies?
463. What is a stratocaster?
464. Where are the headquarters of Eli Lilly?
465. Where did Hillary Clinton graduate college?
466. Where is Glasgow?
467. Who was Samuel Johnsons's friend and biographer?
468. What is tyvek?

469. Who coined the term "cyberspace" in his novel "Neuromancer"?
470. Who is the president of Bolivia?
471. What year did Hitler die?
472. When did the American Civil War end?
473. Who created the character of Scrooge?
474. Who first broke the sound barrier?
475. What is the salary of a U.S. Representative?
476. Name one of the Seven Wonders of the Ancient World.
477. What are the birth dates for scorpios?
478. What is the airport code for Los Angeles International?
479. Who provides telephone service in Orange County, California?
480. What is the zip code for Fremont, CA?
481. Who shot Billy the Kid?
482. Who is the monarch of the United Kingdom?
483. What is sake?
484. What is the name of the Lion King's son in the movie, "The Lion King"?
485. Where is Amsterdam?
486. How many states have a "lemon law" for new automobiles?
487. What was the name of the movie that starred Sharon Stone and Arnold Schwarzenegger?
488. What continent is Bolivia on?
489. What are the Poconos?
490. Where did guinea pigs originate?
491. Where did Woodstock take place?
492. What is the name of the vaccine for chicken pox?
493. What is Betsy Ross famous for?
494. Who wrote the book, "The Grinch Who Stole Christmas"?
495. When did Aldous Huxley write, "Brave New World"?
496. Who wrote the book, "Song of Solomon"?
497. Who portrayed Jake in the television show, "Jake and the Fatman"?
498. Who portrayed Fatman in the television show, "Jake and the Fatman"?
499. Where is Venezuela?
500. What city in Florida is Sea World in?
501. What is the population of Ohio?
502. What is one of the cities that the University of Minnesota is located in?
503. What kind of sports team is the Buffalo Sabres?
504. Who is the founder of the Wal-Mart stores?
505. What city is Massachusetts General Hospital located in?
506. Who reports the weather on the "Good Morning America" television show?
507. When did the California lottery begin?
508. Where is Los Vegas?
509. When was Beethoven born?
510. What's the name of the tiger that advertises for Frosted Flakes cereal?
511. Where is Tufts University?
512. What movie did Madilyn Kahn star in with Gene Wilder?
513. When did the royal wedding of Prince Andrew and Fergie take place?
514. What cereal goes "snap, crackle, pop"?
515. For what disease is the drug Sinemet used as a treatment?

516. Who is Henry Butler?
517. What province is Edmonton located in?
518. In what area of the world was the Six Day War fought?
519. What is the zip code for Parsippany, NJ?
520. When was the first Barbie produced?
521. What was the distinguishing mark on the "Little Rascals" dog?
522. What does EKG stand for?
523. What is Chiricahua the name of?
524. Where did Wicca first develop?
525. Name a symptom of mononucleosis. 526. Where are diamonds mined?
527. When was the NFL established?
528. What are geckos?
529. Who is Terrence Malick?
530. What other name were the "Little Rascals" known as?
531. What was the name of the "Little Rascals" dog?
532. What breed of dog was the "Little Rascals" dog?
533. Who won the rugby world cup in 1987?
534. Where is Windsor Castle?
535. Who portrayed "Rosanne Rosanna-Dana" on the television show "Saturday Night Live"?
536. What is the population of the United States?
537. What animal do buffalo wings come from?
538. When was the bar-code invented?
539. What is witch hazel?
540. What's the abbreviation for limited partnership?
541. What was the purpose of the Manhattan project?
542. What is the most common kind of skin cancer in the U.S.?
543. Name a civil war battlefield. 544. What are pomegranates?
545. Who wrote the song, "Silent Night"?
546. Who portrayed "the man without a face" in the movie of the same name?
547. When was the Hoover Dam constructed?
548. What's the most famous tourist attraction in Rome?
549. At Christmas time, what is the traditional thing to do under the mistletoe?
550. What is the Pennsylvania state income tax rate?
551. What is the name of the Michelangelo painting that shows two hands with fingers touching?
552. What caused the Lynmouth floods?
553. What is the name of the company that manufactures the "American Girl" doll collection?
554. How many zip codes are there in the U.S.?
555. What was the name of the Titanic's captain?
556. How many copies of an album must be sold for it to be a gold album?
557. When is the Tulip Festival in Michigan?
558. What is Giorgio Vasari famous for?
559. Who created the character James Bond?
560. What store does Martha Stewart advertise for?
561. Name an American made motorcycle.
562. What Cruise Line does Kathie Lee Gifford advertise for?

563. Name a movie that the actress, Sandra Bullock, had a role in.
564. Where is the Thomas Edison Museum?
565. What is TCI?
566. What state does MO stand for?
567. Name a female figure skater.
568. What state is the Filenes store located in?
569. How much calcium should an adult female have daily?
570. What hockey team did Wayne Gretzky play for?
571. What hair color can I use to just cover a little gray?
572. How long would it take for a $50 savings bond to mature?
573. How many home runs did Lou Gehrig have during his career?
574. How many states have a lottery?
575. What kind of a sports team is the Wisconsin Badgers?
576. What is the name of Joan Jett's band?
577. Can you give me the name of a clock maker in London, England?
578. What was the name of the sitcom that Alyssa Milano starred in with Tony Danza?
579. What is the real name of the singer, Madonna?
580. Where did yoga originate?
581. What flower did Vincent Van Gogh paint?
582. What radio station did Paul Harvey work for?
583. What's the name of the song Will Smith sings about parents?
584. What does NASA stand for?
585. What famous model was married to Billy Joel?
586. What is the chemical symbol for nitrogen?
587. In what book can I find the story of Aladdin?
588. When did World War I start?
589. What state is Niagra Falls located in?
590. What's the name of the actress who starred in the movie, "Silence of the Lambs"?
591. Who owns the St. Louis Rams?
592. Where could I go to take a ride on a steam locomotive?
593. When was the movie, Caligula, made?
594. Name an American war plane?
595. Where is Burma?
596. How many highway miles to the gallon can you get with the Ford Fiesta?
597. Name a novel written by John Steinbeck.
598. Who wrote the song, "Boys of Summer"?
599. What is the mascot for Notre Dame University?
600. What is typhoid fever?
601. When did the neanderthal man live?
602. Who manufactures the software, "PhotoShop"?
603. How many casinos are in Atlantic City, NJ?
604. What state does Martha Stewart live in?
605. Who was the first woman in space?
606. What are Cushman and Wakefield known for?
607. What is D.B. Cooper known for?
608. When was Nostradamus born?

609. When was "the Great Depression"?
610. What is Archimedes famous for?
611. What is the medical condition of hypertension?
612. Who created the comic strip, "Garfield"?
613. Where is the Isle of Man?
614. Who wrote the book, "Huckleberry Finn"?
615. What day is known as the "national day of prayer"?
616. What is the purpose of a car bra?
617. What are chloroplasts?
618. What is the name of the art of growing miniature trees?
619. What is the telephone number for the University of Kentucky?
620. Who wrote "The Scarlet Letter"?
621. Name an art gallery in New York.
622. What type of hunting are retrievers used for?
623. What party was Winston Churchill a member of?
624. How was Teddy Roosevelt related to FDR?
625. When did the Chernobyl nuclear accident occur?
626. What does SIDS stand for?
627. What's the name of the star of the cooking show, "Galloping Gourmet"?
628. What city is 94.5 KDGE Radio located in?
629. What President became Chief Justice after his presidency?
630. How tall is kilamanjaro?
631. Who won the nobel prize in literature in 1988?
632. What's the name of the Tampa newspaper?
633. How long do hermit crabs live?
634. What is Dr. Ruth's last name?
635. What is cribbage?
636. Italy is the largest producer of what?
637. What wrestling star became "The Incredible Hulk"?
638. Who wrote "The Pit and the Pendulum"?
639. Who manufacturers Magic Chef applicances?
640. When was the first Wall Street Journal published?
641. What is the normal resting heart rate of a healthy adult?
642. Who's the lead singer of the Led Zeppelin band?
643. Who wrote "An Ideal Husband"?
644. What is ouzo?
645. When did the Dow first reach 2000?
646. Who was Charles Lindbergh's wife?
647. How many miles is it from London, England to Plymouth, England?
648. Who is Secretary-General of the United Nations?
649. Who played the teacher in Dead Poet's Society?
650. How many counties are in Indiana?
651. What actress starred in "The Lion in Winter"?
652. Where was Tesla born?
653. What's the name of a hotel in Indianapolis?
654. What U.S. Government agency registers trademarks?
655. Who started the Dominos Pizza chain?
656. What is the hair style called that new military recruits receive?

657. In what country is a stuck-out tongue a friendly greeting?
658. Where is Kings Canyon?
659. Where is the Mayo Clinic?
660. How big is the Electoral College?
661. How much does one ton of cement cost?
662. Where does Mother Angelica live?
663. How many people watch network television?
664. What is the name of a Greek god?
665. What year was the first automobile manufactured?
666. What's the population of Mississippi?
667. What was the name of Jacques Cousteau's ship?
668. What are the names of Jacques Cousteau's two sons?
669. What is Java?
670. What does Final Four refer to in the sports world?
671. What does Knight Ridder publish?
672. What task does the Bouvier breed of dog perform?
673. What sport do the Cleaveland Cavaliers play?
674. What year was the Avery Dennison company founded?
675. What's the population of Biloxi, Mississippi?
676. Name a ballet company Mikhail Baryshnikov has danced for?
677. What was the name of the television show, staring Karl Malden, that had San Francisco in the title?
678. Who was the founding member of the Pink Floyd band?
679. What did Delilah do to Samson's hair?
680. What's the name of the Tokyo Stock Exchange?
681. What actor first portrayed James Bond?
682. What type of horses appear on the Budweiser commercials?
683. What do river otters eat?
684. When did the art of quilting begin?
685. When did Amtrak begin operations?
686. Where is the Smithsonian Institute located?
687. What year did the Vietnam War end?
688. What country are Godiva chocolates from?
689. How many islands does Fiji have?
690. What card company sells Christmas ornaments?
691. Who manufactures synthroid?
692. What year was Desmond Mpilo Tutu awarded the Nobel Peace Prize?
693. What city did the Flintstones live in?
694. Who was the oldest U.S. president?
695. Who was the tallest U.S. president?
696. Where is Santa Lucia?
697. Which U.S. President is buried in Washington, D.C.?
698. Where is Ocho Rios?
699. What year was Janet Jackson's first album released?
700. What is another name for nearsightedness?
701. Winnie the Pooh is what kind of animal?
702. What species was Winnie the Pooh?
703. Winnie the Pooh is an imitation of which animal?

704. What was the species of Winnie the Pooh?
705. Aspartame is also known as what?
706. What is a synonym for aspartame?
707. Aspartame is known by what other name?
708. Aspartame is also called what?
709. Hazmat stands for what?
710. What is the definition of hazmat?
711. What are the names of the tourist attractions in Reims?
712. What do most tourists visit in Reims?
713. What attracts tourists to Reims?
714. What are tourist attractions in Reims?
715. What could I see in Reims?
716. What is worth seeing in Reims?
717. What can one see in Reims?
718. Jude Law was in what movie?
719. Jude Law acted in which film?
720. What is a film starring Jude Law?
721. What film was Jude Law in?
722. What film or films has Jude Law appeared in?
723. What city houses the U.S. headquarters of Procter and Gamble?
724. Where is Procter & Gamble headquartered in the U.S.?
725. What is the U.S. location of Procter & Gamble corporate offices?
726. Procter & Gamble is headquartered in which U.S. city?
727. Where is Procter & Gamble based in the U.S.?
728. What is the recommended daily requirement for folic acid for pregnant women?
729. How much folic acid should a pregnant woman get each day?
730. What is the daily requirement of folic acid for an expectant mother?
731. What amount of folic acid should an expectant mother take daily?
732. Name the first Russian astronaut to do a spacewalk.
733. Who was the first Russian astronaut to walk in space?
734. Who was the first Russian to do a spacewalk?
735. CNN is the abbreviation for what?
736. CNN is an acronym for what?
737. What was the date of CNN's first broadcast?
738. CNN began broadcasting in what year?
739. CNN's first broadcast occurred on what date?
740. When did CNN begin broadcasting?
741. When did CNN go on the air?
742. Who is the owner of CNN?
743. CNN is owned by whom?
744. What newspaper serves Salt Lake City?
745. Name a Salt Lake City newspaper.
746. What is Jane Goodall famous for?
747. What is Jane Goodall known for?
748. Why is Jane Goodall famous?
749. What made Jane Goodall famous?
750. Define thalassemia. 751. What is the meaning of thalassemia?
752. How is thalassemia defined?

753. What soft drink is most heavily caffeinated?
754. What is the most heavily caffeinated soft drink?
755. To get the most caffeine, what soda should I drink?
756. Which type of soda has the greatest amount of caffeine?
757. What soft drink would provide me with the biggest intake of caffeine?
758. What is the collective term for geese?
759. What is the collective noun for geese?
760. What is the term for a group of geese?
761. What is the name given to a group of geese?
762. What is the gestation period for human pregnancies?
763. How long is human gestation?
764. What is the gestation period for humans?
765. A normal human pregnancy lasts how many months?
766. What was the alternate to VHS?
767. What video format was an alternative to VHS?
768. What format was the major competition of VHS?
769. What ethnic group introduced the idea of potlatch?
770. What is the cultural origin of the ceremony of potlatch?
771. Who developed potlatch?
772. Where is Logan Airport?
773. What city is Logan Airport in?
774. Logan International serves what city?
775. Logan International is located in what city?
776. What city's airport is named Logan International?
777. What city is served by Logan International Airport?
778. Woodrow Wilson was president of which university?
779. Name the university of which Woodrow Wilson was president.
780. Woodrow Wilson served as president of what university?
781. What does the acronym CPR mean?
782. What do the initials CPR stand for?
783. CPR is the abbreviation for what?
784. What is the meaning of "CPR"?
785. What was the name of Darth Vader's son?
786. What was Darth Vader's son named?
787. What caused the death of Bob Marley?
788. What killed Bob Marley?
789. What was the cause of Bob Marley's death?
790. What instrument does Ray Charles play?
791. Musician Ray Charles plays what instrument?
792. Ray Charles plays which instrument?
793. Ray Charles is best known for playing what instrument?
794. When was Dick Clark born?
795. When is Dick Clark's birthday?
796. What is Dick Clark's date of birth?
797. What was Poe's birthplace?
798. What was the birthplace of Edgar Allen Poe?
799. Where is Poe's birthplace?
800. What monarch signed the Magna Carta?

801. Which king signed the Magna Carta?
802. Who was the king who was forced to agree to the Magna Carta?
803. What king signed the Magna Carta?
804. Who was the king who signed the Magna Carta?
805. Where is one's corpus callosum found?
806. What part of your body contains the corpus callosum?
807. The corpus callosum is in what part of the body?
808. What English word has the most letters?
809. What English word contains the most letters?
810. What is the longest English word?
811. What is the name of the inventor of silly putty?
812. Silly putty was invented by whom?
813. Who was the inventor of silly putty?
814. When was Berlin's Brandenburg gate erected?
815. What is the date of Boxing Day?
816. What date is Boxing Day?
817. Boxing Day is celebrated on what date?
818. What city does McCarren Airport serve?
819. What city is served by McCarren Airport?
820. McCarren Airport is located in what city?
821. What is the location of McCarren Airport?
822. Where is McCarren Airport located?
823. Who invented "The Muppets"?
824. What was the name of "The Muppets" creator?
825. "The Muppets" was created by whom?
826. Name the creator of "The Muppets".
827. Who is the creator of "The Muppets"?
828. What is the date of Bastille Day?
829. Bastille Day occurs on which date?
830. What is the equivalent of the Red Cross in the Middle East?
831. What is the name of the Islamic counterpart to the Red Cross?
832. Name the Islamic counterpart to the Red Cross.
833. What is the Islamic equivalent of the Red Cross?
834. What is the name given to the Islamic counterpart of the Red Cross?
835. Colin Powell is most famous for what?
836. Colin Powell is best known for what achievement?
837. Who is Colin Powell?
838. Colin Powell is famous for what?
839. What time of year do most people fly?
840. What time of year has the most air travel?
841. What time of year is air travel the heaviest?
842. At what time of year is air travel at a peak?
843. Name a golf course in Myrtle Beach.
844. What is Pittsburg's baseball team called?
845. The major league baseball team in Pittsburgh is called what?
846. Name Pittsburgh's baseball team. 847. What city is the Orange Bowl in?
848. The Orange Bowl is in what city?
849. The Orange Bowl is located in what city?

850. Where is the Orange Bowl?
851. When did Mount St. Helens last erupt?
852. When did Mount St. Helen last have a major eruption?
853. When did Mount St. Helen last have a significant eruption?
854. How do you abbreviate "Original Equipment Manufacturer"?
855. How is "Original Equipment Manufacturer" abbreviated?
856. Where can one find Rider College?
857. What is the location of Rider College?
858. Rider College is located in what city?
859. Where is Rider College?
860. What is the occupation of Nicholas Cage?
861. What is Nicholas Cage's profession?
862. What is Nicholas Cage's occupation?
863. What does caliente translate to in English?
864. What is the English meaning of caliente?
865. What is the meaning of caliente (in English)?
866. What is the English translation for the word "caliente"?
867. What is the Jewish alphabet called?
868. The Jewish alphabet is called what?
869. The Jewish alphabet is known as what?
870. Jackson Pollock was a native of what country?
871. Jackson Pollock is of what nationality?
872. What was the nationality of Jackson Pollock?
873. The Kentucky Horse Park is close to which American city?
874. Where is the Kentucky Horse Park located?
875. Where is the Kentucky Horse Park?
876. What city is the Kentucky Horse Park near?
877. The Kentucky Horse Park is located near what city?
878. What is a nickname for Mississippi?
879. Mississippi is nicknamed what?
880. Mississippi has what name for a state nickname?
881. What is the nickname for the state of Mississippi?
882. What is the nickname of the state of Mississippi?
883. Rotary engines were manufactured by which company?
884. Who made the rotary engine automobile?
885. Rotary engine cars were made by what company?
886. Rotary engines used to be made by whom?
887. What company produced rotary engine vehicles?
888. What is the world's highest peak?
889. What is the highest mountain in the world?
890. Name the highest mountain.
891. What is the name of the tallest mountain in the world?
892. What makes Black Hills, South Dakota a tourist attraction?
893. What are the Black Hills known for?

Appendix F

TREC-9 Score File

Here is the TREC-9 score file for WebQA that uses the source ranking as shown in Table 7.3. The score file shows the TREC-9 score for each question.

Question 201: No correct answer found.
Question 202: Correct answer found at rank 1 (1.00).
Question 203: Correct answer found at rank 1 (1.00).
Question 204: Correct answer found at rank 3 (0.33).
Question 205: No correct answer found.
Question 206: No correct answer found.
Question 207: No correct answer found.
Question 208: No correct answer found.
Question 209: No correct answer found.
Question 210: Correct answer found at rank 4 (0.25).
Question 211: No correct answer found.
Question 212: Correct answer found at rank 1 (1.00).
Question 213: Correct answer found at rank 1 (1.00).
Question 214: Correct answer found at rank 1 (1.00).
Question 215: No correct answer found.
Question 216: Correct answer found at rank 1 (1.00).
Question 217: No correct answer found.
Question 218: Correct answer found at rank 1 (1.00).
Question 219: Correct answer found at rank 1 (1.00).
Question 220: Correct answer found at rank 1 (1.00).
Question 221: Correct answer found at rank 3 (0.33).
Question 222: No correct answer found.
Question 223: Correct answer found at rank 2 (0.50).
Question 224: No correct answer found.
Question 225: Correct answer found at rank 1 (1.00).
Question 226: Correct answer found at rank 4 (0.25).
Question 227: No correct answer found.
Question 228: No correct answer found.
Question 229: No correct answer found.

Question 230: Correct answer found at rank 1 (1.00).
Question 231: No correct answer found.
Question 232: Correct answer found at rank 1 (1.00).
Question 233: Correct answer found at rank 1 (1.00).
Question 234: Correct answer found at rank 1 (1.00).
Question 235: Correct answer found at rank 1 (1.00).
Question 236: No correct answer found.
Question 237: Correct answer found at rank 4 (0.25).
Question 238: Correct answer found at rank 1 (1.00).
Question 239: No correct answer found.
Question 240: No correct answer found.
Question 241: No correct answer found.
Question 242: Correct answer found at rank 1 (1.00).
Question 243: No correct answer found.
Question 244: Correct answer found at rank 1 (1.00).
Question 245: No correct answer found.
Question 246: No correct answer found.
Question 247: Correct answer found at rank 4 (0.25).
Question 248: No correct answer found.
Question 249: Correct answer found at rank 1 (1.00).
Question 250: Correct answer found at rank 1 (1.00).
Question 251: Correct answer found at rank 1 (1.00).
Question 252: No correct answer found.
Question 253: Correct answer found at rank 1 (1.00).
Question 254: Correct answer found at rank 1 (1.00).
Question 255: No correct answer found.
Question 256: No correct answer found.
Question 257: Correct answer found at rank 3 (0.33).
Question 258: No correct answer found.
Question 259: Correct answer found at rank 1 (1.00).
Question 260: Correct answer found at rank 1 (1.00).
Question 261: No correct answer found.
Question 262: No correct answer found.
Question 263: Correct answer found at rank 1 (1.00).
Question 264: No correct answer found.
Question 265: Correct answer found at rank 1 (1.00).
Question 266: Correct answer found at rank 2 (0.50).
Question 267: No correct answer found.
Question 268: Correct answer found at rank 1 (1.00).
Question 269: No correct answer found.
Question 270: Correct answer found at rank 4 (0.25).
Question 271: No correct answer found.
Question 272: Correct answer found at rank 1 (1.00).
Question 273: Correct answer found at rank 2 (0.50).
Question 274: Correct answer found at rank 1 (1.00).
Question 275: No correct answer found.
Question 276: No correct answer found.

Question 277: No correct answer found.
Question 278: No correct answer found.
Question 279: No correct answer found.
Question 280: No correct answer found.
Question 281: Correct answer found at rank 1 (1.00).
Question 282: Correct answer found at rank 1 (1.00).
Question 283: Correct answer found at rank 1 (1.00).
Question 284: No correct answer found.
Question 285: Correct answer found at rank 1 (1.00).
Question 286: Correct answer found at rank 2 (0.50).
Question 287: Correct answer found at rank 1 (1.00).
Question 288: No correct answer found.
Question 289: No correct answer found.
Question 290: Correct answer found at rank 1 (1.00).
Question 291: No correct answer found.
Question 292: Correct answer found at rank 1 (1.00).
Question 293: No correct answer found.
Question 294: Correct answer found at rank 1 (1.00).
Question 295: No correct answer found.
Question 296: No correct answer found.
Question 297: No correct answer found.
Question 298: Correct answer found at rank 1 (1.00).
Question 299: No correct answer found.
Question 300: No correct answer found.
Question 301: No correct answer found.
Question 302: Correct answer found at rank 5 (0.20).
Question 303: Correct answer found at rank 1 (1.00).
Question 304: No correct answer found.
Question 305: No correct answer found.
Question 306: Correct answer found at rank 3 (0.33).
Question 307: No correct answer found.
Question 308: Correct answer found at rank 1 (1.00).
Question 309: Correct answer found at rank 1 (1.00).
Question 310: Correct answer found at rank 1 (1.00).
Question 311: No correct answer found.
Question 312: Correct answer found at rank 3 (0.33).
Question 313: No correct answer found.
Question 314: Correct answer found at rank 1 (1.00).
Question 315: No correct answer found.
Question 316: Correct answer found at rank 4 (0.25).
Question 317: No correct answer found.
Question 318: No correct answer found.
Question 319: Correct answer found at rank 2 (0.50).
Question 320: Correct answer found at rank 1 (1.00).
Question 321: Correct answer found at rank 3 (0.33).
Question 323: No correct answer found.
Question 324: Correct answer found at rank 1 (1.00).

Question 325: Correct answer found at rank 1 (1.00).
Question 326: No correct answer found.
Question 327: Correct answer found at rank 1 (1.00).
Question 328: Correct answer found at rank 1 (1.00).
Question 329: No correct answer found.
Question 330: No correct answer found.
Question 331: No correct answer found.
Question 332: No correct answer found.
Question 334: No correct answer found.
Question 335: No correct answer found.
Question 336: No correct answer found.
Question 337: No correct answer found.
Question 338: Correct answer found at rank 1 (1.00).
Question 340: Correct answer found at rank 2 (0.50).
Question 341: No correct answer found.
Question 342: No correct answer found.
Question 343: Correct answer found at rank 1 (1.00).
Question 344: Correct answer found at rank 5 (0.20).
Question 345: No correct answer found.
Question 346: Correct answer found at rank 2 (0.50).
Question 347: No correct answer found.
Question 348: No correct answer found.
Question 349: No correct answer found.
Question 350: Correct answer found at rank 3 (0.33).
Question 351: No correct answer found.
Question 352: No correct answer found.
Question 353: No correct answer found.
Question 354: No correct answer found.
Question 355: No correct answer found.
Question 356: No correct answer found.
Question 357: No correct answer found.
Question 358: No correct answer found.
Question 359: Correct answer found at rank 2 (0.50).
Question 360: No correct answer found.
Question 361: No correct answer found.
Question 362: Correct answer found at rank 1 (1.00).
Question 363: Correct answer found at rank 1 (1.00).
Question 364: No correct answer found.
Question 365: Correct answer found at rank 1 (1.00).
Question 366: Correct answer found at rank 2 (0.50).
Question 367: Correct answer found at rank 4 (0.25).
Question 368: Correct answer found at rank 3 (0.33).
Question 369: Correct answer found at rank 2 (0.50).
Question 370: Correct answer found at rank 1 (1.00).
Question 371: Correct answer found at rank 2 (0.50).
Question 372: Correct answer found at rank 1 (1.00).
Question 373: Correct answer found at rank 1 (1.00).

Question 374: No correct answer found.
Question 375: Correct answer found at rank 3 (0.33).
Question 376: Correct answer found at rank 1 (1.00).
Question 377: No correct answer found.
Question 378: Correct answer found at rank 1 (1.00).
Question 379: No correct answer found.
Question 380: Correct answer found at rank 1 (1.00).
Question 381: Correct answer found at rank 5 (0.20).
Question 382: No correct answer found.
Question 383: No correct answer found.
Question 384: Correct answer found at rank 1 (1.00).
Question 385: Correct answer found at rank 4 (0.25).
Question 386: Correct answer found at rank 1 (1.00).
Question 387: No correct answer found.
Question 388: No correct answer found.
Question 389: Correct answer found at rank 5 (0.20).
Question 390: No correct answer found.
Question 391: Correct answer found at rank 1 (1.00).
Question 392: No correct answer found.
Question 393: Correct answer found at rank 2 (0.50).
Question 394: No correct answer found.
Question 395: Correct answer found at rank 2 (0.50).
Question 396: No correct answer found.
Question 397: Correct answer found at rank 3 (0.33).
Question 398: Correct answer found at rank 1 (1.00).
Question 399: No correct answer found.
Question 400: No correct answer found.
Question 401: Correct answer found at rank 4 (0.25).
Question 402: Correct answer found at rank 2 (0.50).
Question 403: No correct answer found.
Question 404: Correct answer found at rank 1 (1.00).
Question 405: Correct answer found at rank 2 (0.50).
Question 406: Correct answer found at rank 1 (1.00).
Question 407: Correct answer found at rank 1 (1.00).
Question 408: Correct answer found at rank 2 (0.50).
Question 409: Correct answer found at rank 1 (1.00).
Question 410: Correct answer found at rank 1 (1.00).
Question 411: No correct answer found.
Question 412: No correct answer found.
Question 413: Correct answer found at rank 1 (1.00).
Question 414: Correct answer found at rank 3 (0.33).
Question 415: Correct answer found at rank 1 (1.00).
Question 416: Correct answer found at rank 1 (1.00).
Question 417: Correct answer found at rank 1 (1.00).
Question 418: Correct answer found at rank 4 (0.25).
Question 419: No correct answer found.

Question 420: No correct answer found.
Question 421: No correct answer found.
Question 422: Correct answer found at rank 1 (1.00).
Question 423: No correct answer found.
Question 424: No correct answer found.
Question 425: No correct answer found.
Question 426: No correct answer found.
Question 427: No correct answer found.
Question 428: Correct answer found at rank 2 (0.50).
Question 429: Correct answer found at rank 1 (1.00).
Question 430: Correct answer found at rank 1 (1.00).
Question 431: Correct answer found at rank 1 (1.00).
Question 432: Correct answer found at rank 1 (1.00).
Question 433: Correct answer found at rank 4 (0.25).
Question 434: No correct answer found.
Question 435: No correct answer found.
Question 436: No correct answer found.
Question 437: Correct answer found at rank 1 (1.00).
Question 438: Correct answer found at rank 5 (0.20).
Question 439: No correct answer found.
Question 440: No correct answer found.
Question 441: No correct answer found.
Question 442: Correct answer found at rank 1 (1.00).
Question 444: Correct answer found at rank 1 (1.00).
Question 445: Correct answer found at rank 1 (1.00).
Question 446: Correct answer found at rank 1 (1.00).
Question 447: No correct answer found.
Question 448: No correct answer found.
Question 449: No correct answer found.
Question 450: Correct answer found at rank 2 (0.50).
Question 451: Correct answer found at rank 1 (1.00).
Question 452: Correct answer found at rank 2 (0.50).
Question 453: Correct answer found at rank 1 (1.00).
Question 454: No correct answer found.
Question 455: No correct answer found.
Question 456: Correct answer found at rank 5 (0.20).
Question 457: No correct answer found.
Question 458: No correct answer found.
Question 459: Correct answer found at rank 1 (1.00).
Question 460: Correct answer found at rank 1 (1.00).
Question 461: No correct answer found.
Question 462: Correct answer found at rank 5 (0.20).
Question 463: Correct answer found at rank 2 (0.50).
Question 464: Correct answer found at rank 2 (0.50).
Question 465: No correct answer found.
Question 466: Correct answer found at rank 3 (0.33).

Question 467: No correct answer found.
Question 468: No correct answer found.
Question 469: Correct answer found at rank 2 (0.50).
Question 470: No correct answer found.
Question 471: No correct answer found.
Question 472: No correct answer found.
Question 473: No correct answer found.
Question 474: Correct answer found at rank 1 (1.00).
Question 475: No correct answer found.
Question 476: No correct answer found.
Question 477: No correct answer found.
Question 478: Correct answer found at rank 1 (1.00).
Question 479: No correct answer found.
Question 480: Correct answer found at rank 4 (0.25).
Question 481: Correct answer found at rank 1 (1.00).
Question 482: No correct answer found.
Question 483: No correct answer found.
Question 484: Correct answer found at rank 1 (1.00).
Question 485: Correct answer found at rank 1 (1.00).
Question 486: No correct answer found.
Question 487: No correct answer found.
Question 488: No correct answer found.
Question 489: Correct answer found at rank 1 (1.00).
Question 490: No correct answer found.
Question 491: Correct answer found at rank 2 (0.50).
Question 492: No correct answer found.
Question 493: Correct answer found at rank 1 (1.00).
Question 494: Correct answer found at rank 2 (0.50).
Question 495: Correct answer found at rank 1 (1.00).
Question 496: Correct answer found at rank 3 (0.33).
Question 497: No correct answer found.
Question 498: No correct answer found.
Question 499: No correct answer found.
Question 500: Correct answer found at rank 1 (1.00).
Question 501: Correct answer found at rank 3 (0.33).
Question 502: No correct answer found.
Question 503: Correct answer found at rank 2 (0.50).
Question 504: Correct answer found at rank 3 (0.33).
Question 505: Correct answer found at rank 1 (1.00).
Question 506: No correct answer found.
Question 507: No correct answer found.
Question 508: No correct answer found.
Question 509: Correct answer found at rank 1 (1.00).
Question 510: No correct answer found.
Question 511: No correct answer found.
Question 512: No correct answer found.
Question 513: Correct answer found at rank 4 (0.25).

Question 514: No correct answer found.
Question 515: Correct answer found at rank 1 (1.00).
Question 516: No correct answer found.
Question 517: Correct answer found at rank 1 (1.00).
Question 518: No correct answer found.
Question 519: Correct answer found at rank 1 (1.00).
Question 520: Correct answer found at rank 1 (1.00).
Question 521: No correct answer found.
Question 522: No correct answer found.
Question 523: No correct answer found.
Question 524: No correct answer found.
Question 525: Correct answer found at rank 5 (0.20).
Question 526: No correct answer found.
Question 527: No correct answer found.
Question 528: No correct answer found.
Question 529: No correct answer found.
Question 530: No correct answer found.
Question 531: No correct answer found.
Question 532: No correct answer found.
Question 533: Correct answer found at rank 1 (1.00).
Question 534: Correct answer found at rank 1 (1.00).
Question 535: No correct answer found.
Question 536: No correct answer found.
Question 537: Correct answer found at rank 1 (1.00).
Question 538: No correct answer found.
Question 539: No correct answer found.
Question 540: No correct answer found.
Question 541: Correct answer found at rank 1 (1.00).
Question 542: No correct answer found.
Question 543: No correct answer found.
Question 544: No correct answer found.
Question 545: No correct answer found.
Question 546: Correct answer found at rank 5 (0.20).
Question 547: Correct answer found at rank 2 (0.50).
Question 548: No correct answer found.
Question 549: Correct answer found at rank 2 (0.50).
Question 550: No correct answer found.
Question 551: No correct answer found.
Question 552: No correct answer found.
Question 553: No correct answer found.
Question 554: No correct answer found.
Question 555: Correct answer found at rank 1 (1.00).
Question 556: No correct answer found.
Question 557: No correct answer found.
Question 558: Correct answer found at rank 1 (1.00).
Question 559: Correct answer found at rank 1 (1.00).
Question 560: Correct answer found at rank 2 (0.50).

Question 561: No correct answer found.
Question 562: Correct answer found at rank 2 (0.50).
Question 563: No correct answer found.
Question 564: No correct answer found.
Question 565: No correct answer found.
Question 566: Correct answer found at rank 1 (1.00).
Question 567: No correct answer found.
Question 568: Correct answer found at rank 1 (1.00).
Question 569: No correct answer found.
Question 570: No correct answer found.
Question 571: No correct answer found.
Question 572: No correct answer found.
Question 573: No correct answer found.
Question 574: No correct answer found.
Question 575: No correct answer found.
Question 576: Correct answer found at rank 1 (1.00).
Question 577: No correct answer found.
Question 578: No correct answer found.
Question 579: No correct answer found.
Question 580: No correct answer found.
Question 581: No correct answer found.
Question 582: No correct answer found.
Question 583: No correct answer found.
Question 584: Correct answer found at rank 1 (1.00).
Question 585: Correct answer found at rank 1 (1.00).
Question 586: Correct answer found at rank 1 (1.00).
Question 587: No correct answer found.
Question 588: Correct answer found at rank 4 (0.25).
Question 589: Correct answer found at rank 1 (1.00).
Question 590: Correct answer found at rank 4 (0.25).
Question 592: No correct answer found.
Question 593: Correct answer found at rank 2 (0.50).
Question 594: No correct answer found.
Question 595: No correct answer found.
Question 596: No correct answer found.
Question 597: No correct answer found.
Question 599: No correct answer found.
Question 600: No correct answer found.
Question 601: No correct answer found.
Question 602: Correct answer found at rank 1 (1.00).
Question 603: Correct answer found at rank 2 (0.50).
Question 604: No correct answer found.
Question 605: Correct answer found at rank 2 (0.50).
Question 606: No correct answer found.
Question 607: Correct answer found at rank 2 (0.50).
Question 608: Correct answer found at rank 1 (1.00).
Question 609: No correct answer found.

Question 610: No correct answer found.
Question 611: No correct answer found.
Question 612: Correct answer found at rank 2 (0.50).
Question 613: No correct answer found.
Question 614: No correct answer found.
Question 615: No correct answer found.
Question 616: No correct answer found.
Question 617: No correct answer found.
Question 618: Correct answer found at rank 1 (1.00).
Question 619: No correct answer found.
Question 620: Correct answer found at rank 2 (0.50).
Question 621: No correct answer found.
Question 622: No correct answer found.
Question 623: Correct answer found at rank 1 (1.00).
Question 624: No correct answer found.
Question 625: Correct answer found at rank 1 (1.00).
Question 626: Correct answer found at rank 1 (1.00).
Question 627: Correct answer found at rank 4 (0.25).
Question 628: Correct answer found at rank 1 (1.00).
Question 629: Correct answer found at rank 1 (1.00).
Question 630: No correct answer found.
Question 631: Correct answer found at rank 1 (1.00).
Question 632: Correct answer found at rank 1 (1.00).
Question 633: No correct answer found.
Question 634: No correct answer found.
Question 635: No correct answer found.
Question 636: Correct answer found at rank 1 (1.00).
Question 637: No correct answer found.
Question 638: Correct answer found at rank 2 (0.50).
Question 639: No correct answer found.
Question 640: No correct answer found.
Question 641: No correct answer found.
Question 642: Correct answer found at rank 1 (1.00).
Question 643: Correct answer found at rank 2 (0.50).
Question 644: No correct answer found.
Question 645: No correct answer found.
Question 646: Correct answer found at rank 1 (1.00).
Question 647: No correct answer found.
Question 648: No correct answer found.
Question 649: Correct answer found at rank 1 (1.00).
Question 650: Correct answer found at rank 1 (1.00).
Question 651: Correct answer found at rank 3 (0.33).
Question 652: Correct answer found at rank 2 (0.50).
Question 653: No correct answer found.
Question 654: No correct answer found.
Question 655: No correct answer found.
Question 657: No correct answer found.

Question 658: No correct answer found.
Question 659: Correct answer found at rank 2 (0.50).
Question 660: No correct answer found.
Question 661: No correct answer found.
Question 662: No correct answer found.
Question 664: Correct answer found at rank 2 (0.50).
Question 665: No correct answer found.
Question 666: No correct answer found.
Question 667: Correct answer found at rank 1 (1.00).
Question 668: No correct answer found.
Question 669: No correct answer found.
Question 670: No correct answer found.
Question 671: No correct answer found.
Question 672: No correct answer found.
Question 673: No correct answer found.
Question 674: No correct answer found.
Question 675: No correct answer found.
Question 676: No correct answer found.
Question 677: No correct answer found.
Question 678: No correct answer found.
Question 679: No correct answer found.
Question 680: No correct answer found.
Question 681: Correct answer found at rank 1 (1.00).
Question 682: No correct answer found.
Question 683: Correct answer found at rank 1 (1.00).
Question 684: No correct answer found.
Question 685: Correct answer found at rank 5 (0.20).
Question 686: Correct answer found at rank 1 (1.00).
Question 687: No correct answer found.
Question 688: Correct answer found at rank 1 (1.00).
Question 689: Correct answer found at rank 1 (1.00).
Question 690: No correct answer found.
Question 691: Correct answer found at rank 1 (1.00).
Question 692: Correct answer found at rank 1 (1.00).
Question 693: No correct answer found.
Question 694: No correct answer found.
Question 695: No correct answer found.
Question 696: No correct answer found.
Question 697: No correct answer found.
Question 698: Correct answer found at rank 1 (1.00).
Question 699: Correct answer found at rank 3 (0.33).
Question 700: Correct answer found at rank 2 (0.50).
Question 701: Correct answer found at rank 2 (0.50).
Question 702: No correct answer found.
Question 703: No correct answer found.
Question 704: No correct answer found.
Question 705: Correct answer found at rank 1 (1.00).

Question 706: No correct answer found.
Question 707: Correct answer found at rank 1 (1.00).
Question 708: Correct answer found at rank 1 (1.00).
Question 709: Correct answer found at rank 1 (1.00).
Question 710: Correct answer found at rank 1 (1.00).
Question 711: No correct answer found.
Question 712: Correct answer found at rank 5 (0.20).
Question 713: No correct answer found.
Question 714: No correct answer found.
Question 715: Correct answer found at rank 2 (0.50).
Question 716: No correct answer found.
Question 717: Correct answer found at rank 4 (0.25).
Question 718: No correct answer found.
Question 719: No correct answer found.
Question 720: No correct answer found.
Question 721: No correct answer found.
Question 722: No correct answer found.
Question 723: Correct answer found at rank 4 (0.25).
Question 724: Correct answer found at rank 2 (0.50).
Question 725: No correct answer found.
Question 726: Correct answer found at rank 1 (1.00).
Question 727: Correct answer found at rank 5 (0.20).
Question 728: Correct answer found at rank 1 (1.00).
Question 729: Correct answer found at rank 1 (1.00).
Question 730: No correct answer found.
Question 731: Correct answer found at rank 1 (1.00).
Question 732: No correct answer found.
Question 733: No correct answer found.
Question 734: No correct answer found.
Question 735: Correct answer found at rank 1 (1.00).
Question 736: Correct answer found at rank 1 (1.00).
Question 737: No correct answer found.
Question 738: No correct answer found.
Question 739: No correct answer found.
Question 740: No correct answer found.
Question 741: No correct answer found.
Question 742: No correct answer found.
Question 743: Correct answer found at rank 1 (1.00).
Question 744: No correct answer found.
Question 745: Correct answer found at rank 4 (0.25).
Question 746: No correct answer found.
Question 747: No correct answer found.
Question 748: No correct answer found.
Question 749: No correct answer found.
Question 750: No correct answer found.
Question 751: No correct answer found.
Question 752: No correct answer found.

Question 753: No correct answer found.
Question 754: No correct answer found.
Question 755: No correct answer found.
Question 756: No correct answer found.
Question 757: No correct answer found.
Question 758: Correct answer found at rank 2 (0.50).
Question 759: Correct answer found at rank 2 (0.50).
Question 760: Correct answer found at rank 3 (0.33).
Question 761: Correct answer found at rank 2 (0.50).
Question 762: No correct answer found.
Question 763: No correct answer found.
Question 764: Correct answer found at rank 1 (1.00).
Question 765: Correct answer found at rank 3 (0.33).
Question 766: No correct answer found.
Question 767: No correct answer found.
Question 768: No correct answer found.
Question 769: No correct answer found.
Question 770: No correct answer found.
Question 771: No correct answer found.
Question 772: Correct answer found at rank 1 (1.00).
Question 773: Correct answer found at rank 2 (0.50).
Question 774: Correct answer found at rank 2 (0.50).
Question 775: Correct answer found at rank 2 (0.50).
Question 776: Correct answer found at rank 2 (0.50).
Question 777: Correct answer found at rank 2 (0.50).
Question 778: Correct answer found at rank 1 (1.00).
Question 779: Correct answer found at rank 1 (1.00).
Question 780: Correct answer found at rank 1 (1.00).
Question 781: Correct answer found at rank 1 (1.00).
Question 782: Correct answer found at rank 1 (1.00).
Question 783: Correct answer found at rank 1 (1.00).
Question 784: No correct answer found.
Question 785: Correct answer found at rank 2 (0.50).
Question 786: Correct answer found at rank 4 (0.25).
Question 787: No correct answer found.
Question 788: No correct answer found.
Question 789: No correct answer found.
Question 790: No correct answer found.
Question 791: No correct answer found.
Question 792: No correct answer found.
Question 793: No correct answer found.
Question 795: Correct answer found at rank 1 (1.00).
Question 797: No correct answer found.
Question 798: No correct answer found.
Question 799: No correct answer found.
Question 800: Correct answer found at rank 1 (1.00).

Question 801: Correct answer found at rank 1 (1.00).
Question 802: Correct answer found at rank 1 (1.00).
Question 803: Correct answer found at rank 1 (1.00).
Question 804: Correct answer found at rank 1 (1.00).
Question 805: Correct answer found at rank 2 (0.50).
Question 806: Correct answer found at rank 1 (1.00).
Question 807: Correct answer found at rank 1 (1.00).
Question 808: No correct answer found.
Question 809: No correct answer found.
Question 810: No correct answer found.
Question 812: No correct answer found.
Question 813: No correct answer found.
Question 814: No correct answer found.
Question 815: Correct answer found at rank 1 (1.00).
Question 816: Correct answer found at rank 1 (1.00).
Question 817: Correct answer found at rank 1 (1.00).
Question 818: Correct answer found at rank 1 (1.00).
Question 819: Correct answer found at rank 1 (1.00).
Question 820: Correct answer found at rank 1 (1.00).
Question 821: Correct answer found at rank 1 (1.00).
Question 822: Correct answer found at rank 1 (1.00).
Question 823: Correct answer found at rank 2 (0.50).
Question 824: Correct answer found at rank 2 (0.50).
Question 825: Correct answer found at rank 2 (0.50).
Question 826: Correct answer found at rank 2 (0.50).
Question 827: Correct answer found at rank 2 (0.50).
Question 828: Correct answer found at rank 1 (1.00).
Question 829: Correct answer found at rank 1 (1.00).
Question 830: No correct answer found.
Question 831: No correct answer found.
Question 832: No correct answer found.
Question 833: No correct answer found.
Question 834: No correct answer found.
Question 835: No correct answer found.
Question 836: No correct answer found.
Question 837: Correct answer found at rank 1 (1.00).
Question 838: No correct answer found.
Question 839: No correct answer found.
Question 840: No correct answer found.
Question 841: No correct answer found.
Question 842: No correct answer found.
Question 843: No correct answer found.
Question 844: Correct answer found at rank 3 (0.33).
Question 845: Correct answer found at rank 1 (1.00).
Question 846: Correct answer found at rank 1 (1.00).

Question 847: No correct answer found.
Question 848: No correct answer found.
Question 849: No correct answer found.
Question 850: Correct answer found at rank 2 (0.50).
Question 851: Correct answer found at rank 1 (1.00).
Question 852: Correct answer found at rank 1 (1.00).
Question 853: Correct answer found at rank 1 (1.00).
Question 854: No correct answer found.
Question 855: No correct answer found.
Question 856: No correct answer found.
Question 857: No correct answer found.
Question 858: No correct answer found.
Question 859: No correct answer found.
Question 860: Correct answer found at rank 3 (0.33).
Question 861: Correct answer found at rank 4 (0.25).
Question 862: Correct answer found at rank 4 (0.25).
Question 863: Correct answer found at rank 4 (0.25).
Question 864: Correct answer found at rank 2 (0.50).
Question 865: Correct answer found at rank 2 (0.50).
Question 866: Correct answer found at rank 3 (0.33).
Question 867: No correct answer found.
Question 868: No correct answer found.
Question 869: No correct answer found.
Question 870: No correct answer found.
Question 871: Correct answer found at rank 2 (0.50).
Question 872: Correct answer found at rank 2 (0.50).
Question 873: Correct answer found at rank 2 (0.50).
Question 874: Correct answer found at rank 1 (1.00).
Question 875: Correct answer found at rank 1 (1.00).
Question 876: Correct answer found at rank 1 (1.00).
Question 877: Correct answer found at rank 2 (0.50).
Question 878: Correct answer found at rank 1 (1.00).
Question 879: No correct answer found.
Question 880: Correct answer found at rank 1 (1.00).
Question 881: Correct answer found at rank 1 (1.00).
Question 882: Correct answer found at rank 1 (1.00).
Question 883: No correct answer found.
Question 884: Correct answer found at rank 1 (1.00).
Question 885: Correct answer found at rank 1 (1.00).
Question 886: Correct answer found at rank 1 (1.00).
Question 887: Correct answer found at rank 1 (1.00).
Question 888: Correct answer found at rank 2 (0.50).
Question 889: Correct answer found at rank 2 (0.50).
Question 890: No correct answer found.
Question 891: Correct answer found at rank 4 (0.25).

Question 892: No correct answer found.
Question 893: Correct answer found at rank 2 (0.50).

Mean reciprocal rank over 682 questions is 0.360
360 questions had no answers found in top 5 responses.

Bibliography

[ACM93] S. Abiteboul, S. Cluet, and T. Milo. Querying and updating the file. In *Proceedings of 19th International Conference on Very Large Data Bases (VLDB)*, pages 73–84, 1993.

[All02] Alltheweb.com. Main Page: http://www.alltheweb.com/, 2002.

[Alt02] Alta vista. Main Page: http://www.altavista.com/, 2002.

[AM98] G. Arocena and A. Mendelzon. Weboql: Restructuring documents, databases and webs. In *Proceedings of 14th. International Conference on Data Engineering (ICDE)*, pages 24–33, 1998.

[AMM97] P. Atzeni, G. Mecca, and P. Merialdo. To weave the Web. In *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB)*, pages 206–215, 1997.

[AQM+97] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *Journals of Digital Libraries*, 1(1):68–88, 1997.

[Ask02] Ask jeeves. Main Page: http://www.ask.com/, 2002.

[Bal97] M. Balabanovic. An adaptive Web page recommendation service. In *Proceedings of 1st International Conference on Autonomous Agents*, pages 378–385, 1997.

[BDHS96] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 505–516, 1996.

[BLCL+94] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The World Wide Web. *Communication of the ACM*, 37(8):76–82, 1994.

[BP98] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of 7th WWW Conference*, pages 107–117, 1998.

[BYBC+00] Z. Bar-Yossef, A. Berg, S. Chien, J. Fakcharoenphol, and D. Weitz. Approximating aggregate queries about Web pages via random walks. In *Proceedings of 26th International Conference on Very Large Data Bases (VLDB)*, pages 535–544, 2000.

[BYRN99] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, New York, USA, 1999.

[CACS94] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 313–324, 1994.

[Cat94] R. G. G. Cattell. *The Object Database Standard: ODMG-93.* Morgan Kaufmann, San Francisco, California, 1994.

[CDTW00] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: a scalable continuous query system for Internet databases. In *Proceedings of ACM SIGMOD International Conference on Management of data*, pages 379–390, 2000.

[CGMH+94] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS project: integration of heterogeneous information sources. In *Proceedings of 16th Information Processing Society of Japan*, pages 7–18, 1994.

[CM90] M. P. Consens and A. O. Mendelzon. Graphlog: a visual formalism for real life recursion. In *Proceeding of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 404–416, 1990.

[CMW87] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. A graphical query language supporting recursion. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 323–330, 1987.

[CMW88] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. G+: recursive queries without recursion. In *Proceedings of 2nd International Conference on Expert Database Systems*, pages 645–666, 1988.

[Cop02] Copernic. Main Page: http://www.copernic.com/, 2002.

[DBJ99] C. E. Dyreson, M. H. Bohlen, and C. S. Jensen. Capturing and querying multiple aspects of semistructured data. In *Proceedings of 25th International Conference on Very Large Data Bases (VLDB)*, pages 290–301, 1999.

[DLCT00] Y. Diao, H. Lu, S. Chen, and Z. Tian. Toward learning based Web query processing. In *Proceedings of 26th International Conference on Very Large Data Bases (VLDB)*, pages 317–328, 2000.

[dmo02] dmoz. Main Page: http://dmoz.org/, 2002.

[Dog02] Dogpile. Main Page: http://www.dogpile.com/, 2002.

[Env02] Environment canada. Main Page: http://weatheroffice.ec.gc.ca/, 2002.

[Exc02] Excite. Main Page: http://www.excite.com/, 2002.

[FFL97] M. Fernandez, D. Florescu, and A. Levy. A query language for a web-site management system. *SIGMOD Record*, 26(3):4–11, 1997.

[FLM98] D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the World-Wide Web: a survey. *SIGMOD Record, 27(3):59-74*, 27(3):59–74, 1998.

[Gal98] S. Galvin. *Operating System Concepts 5th Edition*. Addison Wesley, California, USA, 1998.

[Goo02a] Google. Main Page: http://www.google.com/, 2002.

[Goo02b] Google protects its search results. Main Page: http://news.com.com/2100-1023-883558.html, 2002.

[Goo02c] Google web apis. Main Page: http://www.google.com/apis/index.html, 2002.

[GW97] R. Goldman and J. Widom. DataGuides: enabling query formulation and optimization in semistructured databases. In *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB)*, pages 436–445, 1997.

[GW00] R. Goldman and J. Widom. WSQ/DSQ: A practical approach for combined querying of databases and the Web. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 285–296, 2000.

[HGMI+95] J. Hammer, H. Garcia-Molina, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. Information translation, mediation, and mosaic-based browsing in the TSIMMIS system. In *Proceedings of ACM SIGMOD International Conference on Management of data*, page 483, 1995.

[Jav02] Java 2 platform, standard edition, v 1.4.0 api specification. Main Page: http://java.sun.com/j2se/1.4/docs/api/index.html, 2002.

[JF97] T. Joachims and D. Freitag. WebWatcher: a tour guide for the World Wide Web. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 770–777, 1997.

[KEW01] C. C. T. Kwok, O. Etzioni, and D. S. Weld. Scaling question answering to the Web. In *Proceedings of 10th International World Wide Web Conference*, pages 150–161, 2001.

[KKS01] L. Kerschberg, W. Kim, and A. Scime. A semantic taxonomy-based personalizable meta-search agent. In *Proceedings of 2nd International Conference on Web Information Systems and Engineering (WISE)*, pages 53–62, 2001.

[Kor97] R. R. Korfhage. *Information Storage and Retrieval.* John Wiley and Sons, New York, USA, 1997.

[KS95] D. Konopnicki and O. Shmueli. W3QS: A query system for the World Wide Web. In *Proceedings of 21th International Conference on Very Large Data Bases (VLDB)*, pages 54–65, 1995.

[Loo02] Looksmart. Main Page: http://www.looksmart.com/, 2002.

[LRO96] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of 22nd International Conference on Very Large Data Bases (VLDB)*, pages 251–262, 1996.

[LSS93] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. On the logical foundations of schema integration and evolution in heterogeneous database systems. In *Proceedings of 3rd International Workshop on Deductive and Object-Oriented Databases (DOOD)*, pages 81–100, 1993.

[LSS96] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. A declarative language for querying and restructuring the Web. In *Proceedings of 6th International Workshop on Research Issues in Data Engineering (RIDE)*, pages 12–21, 1996.

[MAGW97] J. McHugh, S. Abiteboul, R. Goldman, and J. Widom. Lore: a database management system for semistructured data. *ACM SIGMOD Record*, 26(8):54–66, 1997.

[Mam02] Mamma. Main Page: http://www.mamma.com/, 2002.

[MB98] F. Menczer and R. Belew. *Adaptive Retrieval Agents: Internalizing Local Context and Scaling up to the Web. Technical Report CS98-579.* Department of Computer Science, University of California, San Diego, 1998.

[Met02] Metacrawler. Main Page: http://www.metacrawler.com/, 2002.

[MMM97] A. O. Mendelzon, G. A. Mihaila, and T. Milo. Querying the World Wide Web. *Journals on Digital Libraries*, 1(1):54–67, 1997.

[Ove02] Overture. Main Page: http://www.overture.com/, 2002.

[Ozs99] M. T. Ozsu. *Principles of Distributed Database Systems 2nd Ed.* Prentice Hall, Upper Saddle River, New Jersey, 1999.

[PAGM96] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *Proceedings of 22nd International Conference on Very Large Data Bases (VLDB)*, pages 413–424, 1996.

[PdBA+92] J. Paredaens, J. V. den Bussche, M. Andries, M. Gemis, M. Gyssens, I. Thyssens, D. V. Gucht, V. Sarathy, and L. Saxton. An overview of GOOD. *ACM SIGMOD Record 21(1):25-31*, 21(1):25–31, 1992.

[PGMU96] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. Medmaker: A mediation system based on declarative specifications. In *Proceedings of International Conference of Data Engineering (ICDE 1996)*, pages 132–141, 1996.

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proceedings of 11th Data Engineering Conference*, pages 251–260, 1995.

[PMB96] M. Pazzani, J. Muramatsu, and D. Billsus. Syskill & Webert: identifying interesting Web sites. In *Proceedings of 13th National Conference on Artificial Intelligence*, pages 54–61, 1996.

[PTdB93] M. Gemis J. Paredaens, I. Thyssens, and J. V. den Bussche. GOOD: A Graph-Oriented Object Database System. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 505–510, 1993.

[RQZ+01] D. R. Radev, H. Qi, Z. Zheng, S. Blair-Goldensohn, Z. Zhang, W. Fan, and J. Prager. Mining the Web for answers to natural language questions. In *Proceedings of 10th International Conference on Information and Knowledge Management (CIKM)*, pages 143–150, 2001.

[STA02] Start. Main Page: http://www.ai.mit.edu/projects/infolab/, 2002.

[Teo02] Teoma. Main Page: http://www.teoma.com/, 2002.

[TRE01] *Proceedings of Ninth Text REtrieval Conference (TREC-9)*. 2001.

[WFB02] World factbook. Main Page: http://www.cia.gov/cia/publications/factbook/index, 2002.

[WUn02] Weather underground. Main Page: http://www.wunderground.com/, 2002.

[Yah02a] What is google? how is it different from yahoo!? Main Page: http://help.yahoo.com/help/us/ysearch/ysearch-18.html, 2002.

[Yah02b] Yahoo. Main Page: http://www.yahoo.com/, 2002.