Spring 5-31-2001

WAQS : a web-based approximate query system

George Jyh-Shian Chang
New Jersey Institute of Technology

ABSTRACT

WAQS: A Web-based Approximate Query System

by
George Jyh-Shian Chang

The Web is often viewed as a gigantic database holding vast stores of information and provides ubiquitous accessibility to end-users. Since its inception, the Internet has experienced explosive growth both in the number of users and the amount of content available on it. However, searching for information on the Web has become increasingly difficult. Although query languages have long been part of database management systems, the standard query language being the Structural Query Language is not suitable for the Web content retrieval.

In this dissertation, a new technique for document retrieval on the Web is presented. This technique is designed to allow a detailed retrieval and hence reduce the amount of matches returned by typical search engines. The main objective of this technique is to allow the query to be based on not just keywords but also the location of the keywords within the logical structure of a document. In addition, the technique also provides approximate search capabilities based on the notion of *Distance* and *Variable Length Don't Cares.* The proposed techniques have been implemented in a system, called *Web-Based Approximate Query System* which contains an SQL-like query language called *Web-Based Approximate Query Language.*

Web-Based Approximate Query Language has also been integrated with EnviroDaemon, an environmental domain specific search engine. It provides EnviroDaemon with more detailed searching capabilities than just keyword-based search. Implementation details, technical results and future work are presented in this dissertation.

WAQS: A WEB-BASED APPROXIMATE QUERY SYSTEM

by
George Jyh-Shian Chang

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer and Information Science

Department of Computer and Information Science

May 2001

APPROVAL PAGE

WAQS: A Web-based Approximate Query System

George Jyh-Shian Chang

5/3/01

Dr. Jason T. L. Wang, Dissertation Advisor — Date
Professor of Computer and Information Science Department
New Jersey Institute of Technology, Newark, NJ

5/3/01

Dr. James A. McHugh, Dissertation Co-Advisor — Date
Professor of Computer and Information Science Department
New Jersey Institute of Technology, Newark, NJ

5/3/01

Dr. Byoung-Kee Yi, Committee Member — Date
Assistant Professor of Computer and Information Science Department
New Jersey Institute of Technology, Newark, NJ

5/3/01

Dr. Michael Halper, Committee Member — Date
Associate Professor of Mathematics and Computer Science Department
Kean University, Union, NJ

5/3/01

Dr. Qicheng Ma, Committee Member — Date
Senior Scientist in Functional Genomes Department
Novartis Pharmaceuticals Corporation, Summit, NJ

BIOGRAPHICAL SKETCH

Author: George Jyh-Shian Chang

Degree: Doctor of Philosophy

Date: May 2001

Education:

- Doctor of Philosophy in Computer and Information Science
  New Jersey Institute of Technology
  Newark, New Jersey, May 2001.
- Master of Science in Computer and Information Science
  New Jersey Institute of Technology
  Newark, New Jersey, May 1995.
- Bachelor of Science in Computer Science
  Bachelor of Science in Applied Mathematics and Statistics
  State University of New York at Stony Brook
  Stony Brook, New York, May 1994.

Major: Computer and Information Science

Book Publication:

"Mining the World Wide Web: An Information Search Approach," with M. J. Healey, J. A. M. McHugh and J. T. L. Wang. *Kluwer Academic Publishers, 2001.*

Paper Publications:

"Finding Approximate Patterns in Undirected Acyclic Graphs: Algorithms and Applications," with J. T. L. Wang, K. Zhang, Z. Weinberg and D. Shasha. *Pattern Recognition, accepted.*

"Precise Environmental Search: EnviroDaemon with Hierarchical Information Search," with M. J. Healey. and J. T. L. Wang. *Journal of Environmental Quality Management*, Vol. 9, No. 1, 1999, pp. 51–62.

"Change Detection of Websites: A Case Study in Environmental Engineering Domain," with J. T. L. Wang and M. J. Healey. *Proceedings of the Fourth Joint Conference on Information Science Conferences, the Second International Workshop on Intelligent Control*, Research Triangle Park, NC. October 1998.

"A Graphical Environment for Change Detection in Structured Documents," with G. Patel, L. Relihan and J. T. L. Wang. *Proceedings of the Twenty-First Annual International Computer Software and Application Conference*, Washington, D.C., pp. 536–541, August 1997.

"Structural Matching and Discovery in Document Databases," with J. T. L. Wang, D. Shasha, L. Relihan, K. Zhang, and G. Patel. *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona, pp. 560–563, May 1997.

"A Visualization Tool for Pattern Matching and Discovery in Scientific Databases," with J. T. L. Wang, G. W. Chirn and C. Y. Chang. *Proceedings of the Eighth International Conference on Software Engineering and Knowledge Engineering*, Lake Tahoe, Nevada, pp. 563–570, June 1996.

"An Integrated Toolkit for Pattern Matching and Pattern Discovery in Scientific, Program and Document Databases," with J. T. L. Wang, G. W. Chirn, C. Y. Chang, A. Noriega and K. Pysniak, *Proceedings of the 7th International Conference on Software Engineering and Knowledge Engineering*, Rockville, Maryland, pp. 497, June 1995.

Exhibits and Demonstrations:

"The 4th Annual Coalition for National Science Funding Exhibition for members of Congress and their staffs", Washington, D.C., May 20, 1998. Pattern Matching and Discovery Toolkit for Scientific Databases. Structural Matching and Discovery in Document Databases.

"Annual Industrial Advisory Committee Meeting of the Board of Trustees," NJIT CIS Conference Room. April 21, 1998. Structural Matching and Discovery in Document Databases.

"1997 ACM SIGMOD International Conference on Management of Data," Tucson, Arizona, May 11-15, 1997. Structural Matching and Discovery in Document Databases.

"The 7th International Conference on Software Engineering and Knowledge Engineering," Rockville, Maryland, June 22-24, 1995. An Integrated Toolkit for Pattern Matching and Pattern Discovery in Scientific Databases.

To my parents,
Ching-Hui Chang and Meei-Lan Sheu

ACKNOWLEDGMENT

I am indebted to my advisor, Professor Jason T. L. Wang and co-advisor, Professor James A. M. McHugh for their guidance, patience and encouragement throughout this work.

I would like to express my appreciation to Dr. Marcus J. Healey for his suggestions and supports in environmental science domain and thank Professor Byoung-Kee Yi and Professor Michael Halper and Dr. Qicheng Ma for being committee members.

Special thanks to the faculty of Computer and Information Science Department at New Jersey Institute of Technology, Computer Science Department at State University of New York at Stony Brook, and Mathematics Department at Forest Hills High School for teaching me the foundations of computer science.

I would also like to thank friends who I have worked with or supported me, directly or indirectly, in the completion of my dissertation: Frank Brophy, Chia-Yo Chang, Michael Chang, Jeff Cheng, Sophia Cheng, Gung-Wei Chirn, George Chu, Jeanie Hsiang, Yvonne Huang, Kuo Kuo, Limin Liu, Yufen Liu, Qicheng Ma, Girish Patel, Iris Rang, Ren Shen, Janey Tsai, Sophia Tzeng, Xiong Wang, Zhiyuan Wang, and Weiping Zhang.

Most of all, I want to express my deepest appreciation to my father Ching-Hui Chang, my mother Meei-Lan Sheu, and my brother David Chang. My accomplishment would not be possible without their continuous support, encouragement, and love.

TABLE OF CONTENTS

Chapter **Page**

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS
(continued)

Chapter **Page**

TABLE OF CONTENTS
(continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The World Wide Web (Web), which emerged in the early 1990s, has made great strides in the late 1990s. Its explosive growth is expected to continue into the next millennium. The contributing factors to this explosive growth include the widespread use of microcomputers, advances in hardware (microprocessors, memory and storage) technologies, increased ease of use in computer software packages, and most importantly – tremendous opportunities the Web offers for all businesses.

The consequence of the popularity of the Web as a global information system is that it has flooded us with a large amount of data and information. The Web has achieved an extraordinary amount of information content and provided ubiquitous accessibility to end-users, yet it has at the same time become very difficult to locate specific information. In this sea of data and information, searching for a piece of information is like finding a needle in a haystack.

For example, the most common way to search for a document of interest in this repository is to use navigation-oriented browsers. While browsing is a convenient way of viewing documents, it is not a good search technique because of the following limitations. First, unless the user knows exactly where the document of interest is located, it is very time consuming and difficult to find the target document by browsing. Second, it does not provide a global view as to where the target document might be. Looking for useful information on the Web is often a tedious and frustrating experience. Therefore, new tools and techniques are needed to assist us in intelligently searching for and discovering useful information on the Web.

Due to the limitations of browsing as a search technique, various tools have been developed using information retrieval techniques to speed–up and facilitate

information search on the Web. Important Web search tools are categorized as following:

- *Keyword-Based Search Engines*
- *Query-Based Search Systems*
- *Mediators and Wrappers*
- *Multimedia Search Engines*
- *Web Crawling Agents*

Keyword-based search engines provide a fast way of searching for information on the Web. While many different types of search engines exist on the Web, they usually offer only keyword-based searches and indices. Most search engines index site documents via one of the following methods: by title, by document content, by Uniform Resource Locators (URLs), or a combination of the above methods. Many directories on the Web have also been created and categorize sites by topics of interest. Yahoo is perhaps the best known among the directories.

Though search engines provide a quick and easy way of finding information on the Web, the process of using them is still often tedious and frustrating. Since search engines are typically designed to maximize recall, there is little or no attempt to filter the information intelligently, so manual browsing of the original documents is required to find relevant information. Another shortcoming of the current Web search engines is the lack of a high-level querying facility to automate the information retrieval process. The compelling need for a high-level query language has led to the developments of query-based search systems. such as WebSQL [104], WebLog [82], W3QS [78], which allow the user to do queries on Web information using an SQL-like query language.

Unlike query-based search systems that use search engines as backends, mediators and wrappers have been developed which are based on a Database Management Systems (DBMS). Mediators and wrappers can take advantage of the underlying DBMS querying and data storing facilities. Hence, they can provide extensive backend analytical processing.

Web content has transformed the Web from a textual to a multimedia-based repository. As sounds and images proliferate on the Web, multimedia search engines have become increasingly important. Multimedia search engines are designed to handle non-textual based content. Hence, having scalable and responsive system is critical in its design.

Web crawling agents is another essential category of Web search tools. These agents are responsible for discovering new content that can be indexed by the search systems. Hence, its performance is critical to the overall accuracy of a search tool.

In this dissertation, a new query-based system called *Web-Wide Approximate Query System* (*WAQS*), for the Web, is presented. WAQS examines the explicit embedded semantic structure associated with each hypertext document using the *Document Type Definition Model* (*DTDM*). In addition, inspired by the various query languages designed for the Web, an SQL-like query language, *Web-Based Approximate Query Language* (*WAQL*), has been designed and implemented to provide querying capability for WAQS. WAQL extends the power of the existing search engines by exploiting the hierarchy of the underlying HTML tags.

The rest of the dissertation is organized as following: Chapter 2 discusses keyword-based search engines; Chapter 3 discusses query-based search systems along with the syntax and semantics of WAQL that were developed; Chapter 4 discusses mediators and wrappers; Chapter 5 discusses multimedia search engines; Chapter 6 discusses Web crawling agents related work that were used in building the system; Chapter 7 discusses the integration of WAQL with a domain specific search engine

prototype called EnviroDaemon. This search engine uses tools freely available on the Web to gather environmental science related information on the Web. Chapter 8 discusses future research that can be extended from this dissertation.

CHAPTER 2
KEYWORD-BASED SEARCH ENGINES

The World Wide Web (WWW), also known as the Web, was introduced in 1992 at the Center for European Nuclear Research (CERN) in Switzerland [17]. What began as a means of facilitating data sharing in different formats among physicists at CERN is today a mammoth, heterogeneous, non-administered, distributed, global information system that is revolutionizing the information age. The Web is organized as a set of hypertext documents interconnected by hyperlinks, used in the *Hypertext Markup Language* (HTML) to construct links between documents. The many potential benefits the Web augurs have spurred research in information search/filtering [30, 81], Web/database integration [23, 90], Web querying systems [1, 78, 82, 104], and data mining [37, 138]. The Web has also brought together researchers from areas as diverse as communications, electronic publishing, language processing, and databases, as well as from multiple scientific and business domains.

Prior to the Web, the Internet was only a massive interconnected network of computers, which was text oriented and used primarily by large corporations and research institutes for sharing information. Since the inception of the Web, the Internet has experienced explosive growth both in the number of users and the amount of content available on it. The Web has added the interconnection between documents with different contents. The contents include images, graphics, sound and video, in addition to text. Millions of knowledgeable Internet users have turned the Web into a remarkable repository of information. Indeed, the ability of the Web to collect and to disseminate information has in a few short years arguably transcended what television broadcasting took 50 years to accomplish.

The availability of user-friendly, graphics-oriented interfaces has contributed substantially to the growth and usefulness of the Web . Browsers, such as Netscape Navigator and Internet Explorer, and multimedia players such as RealPlayer, have

greatly simplified the use of the Internet and expanded its appeal. However, the Web's rapid growth has exacerbated the burden of sifting and winnowing the information that can be accessed on the Web. In many respects, the Web is similar to a library with a limited catalog system: full of valuable information, but confusing and time-consuming to search.

As the number of computers connected to the Internet grows, so does the volume of content on the Web. Ironically, the same hyperlink organization that gives the Web its power, also makes its organization remarkably chaotic. Searching for specific information has become increasingly difficult in this dynamic, distributed, heterogeneous, and unstructured repository. Using browsing as a search method is often a problematic option for finding wanted information, frequently like looking for a needle in a haystack.

In this chapter, the focus is on keyword-based search engines and related techniques. Chapter 3 addresses issues in query based search. Chapter 4 discusses mediators and wrappers. Chapter 4 describes multimedia search engines. The techniques presented are not all new to the information retrieval and database community. Rather, the emergence of the Web has promoted the refinement and perfection of existing techniques in hardware and software to meet the demands of the Internet.

2.1 Search Engines

Web search engines, also called *Web indexes*, *index servers*, or simply *search engines*, have become the most visited Web sites on the Internet. Indeed, the most common method used today to search the Web consists of sending information requests to these servers, which index as many documents as they can find by navigating the Web. A salient problem is that an informed user must be aware of the various index servers, their strengths, weaknesses, and the peculiarities of their query interfaces.

The need for querying information on the Web has led to the development of a number of tools that search these indexes based on keywords specified by users.

Several such tools, e.g., AltaVista [5] and WebCrawler [145], have become the gateway to the Web for the neophyte and information oracles for the experienced. These tools serve the information needs of users by finding useful information in the sea of Internet data. While early search engines were research prototypes, today's search engines represent billions of dollars of invested capital. They are maintained by commercial firms to promote services primarily related to e-commerce.

Search engines are similar to ordinary library search systems in that they allow users to type in subjects of interest as keywords, and return a set of results satisfying given conditions. However, search engines seek, update, and index documents on a far more massive scale than library systems. In addition, the Web is a far more dynamic domain than a library, making it harder to maintain the currency of indexes.

According to a recent survey [70], the most popular search engines include AltaVista [5], Excite [52] (Magellan and WebCrawler have been acquired by Excite), HotBot [73], Infoseek [63], Lycos [92], Yahoo! [151]. These search engines are multi-domain oriented and span the entire Web. AltaVista acts like the Yellow Pages for the Web. It indexes the full-text of documents. Excite uses artificial intelligence, employing concept-driven or "fuzzy" search. HotBot's Slurp spider is the most powerful of all the Web "crawlers", capable of indexing the entire Web in about a week, an ability which translates into fewer out-of-date links. Infoseek is the most user-friendly search engine, with a clean, intuitive interface. Lycos is like a bibliographic database service except that its abstracts are generated by programs called Web crawlers, rather than human indexers. Yahoo! is actually not a search engine, but rather a directory of the Web compiled manually by human indexers. Some search engines focus on single domains [61]. Others are meta-search ones that harness the power of multiple existing search engines. Still others employ user profiles

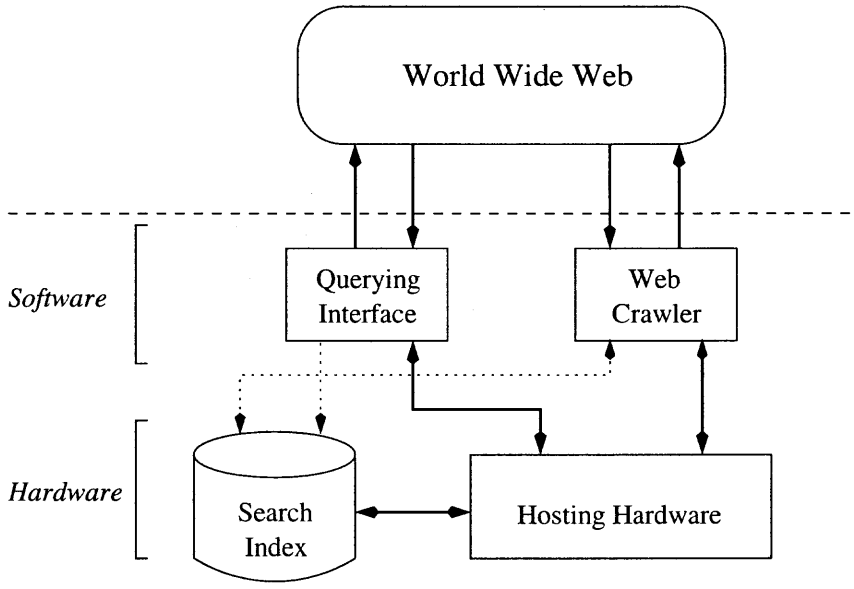


Figure 2.1 The architecture of a search engine.

known, in Web parlance, as "cookies", and server logs to determine the most popular sites.

Another very popular search engine, Google [67], has recently announced that its searches encompass an average of more than one billion (10^9) pages. Its index consists of 560 million full-text and 500 million partially indexed pages. This news has placed Google as the largest index on the Internet.

Search Engine Watch [128], a portal on search engine news and resources, categorizes search engines as: *Major*, *Kids*, *Meta*, *Multimedia*, *News*, *Regional*, and *Specialized*. Typically, a search engine has four major components: *Querying Interface*, *Search Index*, *Web Crawler*, and *Hosting Hardware*, as illustrated in Figure 2.1. The solid lines in this figure represent direct interactions; dotted lines are indirect interactions between components. The first three components are described in detail below. Hardware issues are beyond the scope of this dissertation. Detailed discussion on the hardware environments that powers the AltaVista search engine can be found in [119].

2.1.1 Querying Interface

The querying interface is the software component that searches through indexes to find matches based on search criteria provided in a *query*. Retrieved documents are ranked according to a predefined relevance matrix. In a keyword-based query, combinations of keywords can be formulated to search for documents containing such keywords.

Such an interface is simple, intuitive, and easy to use, even for naive users. Its more effective use requires experience and more complex combinations of keywords and operators. In this subsection, some of the most popular ways of formulating keyword-based queries for search engines are discussed.

2.1.1.1 Single-Word Query This is the most elementary querying interface for a search engine. In this model, a keyword is used as an input to the search engine. The result of the search is the set of documents containing at least one occurrence of the input keyword. Depending on the model used by the search engine index, the keyword could occur as a word or as part of a word in the document. A semantically enhanced version of keyword search, called *field* search, is provided by some systems to focus the search on specific structural aspects of a Web page. For example, a query with the term, `title:"information"`, would return only documents with `information` in the title, using the HTML tags of a document to identify document components such as titles. Table 2.1 summarizes a list of commonly used search fields.

2.1.1.2 Multiple-Term Query In this model, the querying keyword is not restricted to a single word or phrase. Multiple terms can be used in formulating query criteria. The terms can be basic terms (words) or Boolean expressions built on

Field	*Field Location*	*Example*
text:	Body	text:information
title:	Title	title:database
link:	Hyperlink	link:kluwer.nl
anchor:	Visual Part of a Hyperlink	anchor:mining
url:	URL	url:www.xyz.com
host:	Computer Name	host:xyz.com
domain:	Specific Domain	domain:edu
image:	Image Name	image:map.gif
applet:	Applet Name	applet:tetris
object:	Object Name	object:game

Table 2.1 Searchable Web page fields.

the basic terms. Most search engines minimally provide the following user-selected methods of expressing multiple-term queries:

- Resulting documents must contain *all* the keywords.
- Resulting documents must contain *any* of the keywords.

For example, a multiple-term query such as "information **and** database" is used to find all documents containing both of the words "information" and "database". A query with expressions such as "((data **or** Web) **and** mining)" identifies all documents containing either "data **and** mining" or "Web **and** mining". In general, one can use Boolean operators as connectives to connect two terms. Let e_1 and e_2 be two terms. The most commonly used Boolean operators are:

- **and** – for combining query terms. For example, the query (e_1 **and** e_2) selects all documents that contain both e_1 and e_2.
- **or** – for including either the first or the second term. For example, the query (e_1 **or** e_2) selects all documents that contain e_1 or e_2.

- **not** – for excluding documents with a query term. For example, the query (**not** e_1) excludes documents with e_1 from the result.

Clearly, these three operators can only be performed on documents indexed by a search engine. Duplicates are removed from the result.

2.1.1.3 Context Based Query When the context of the querying keyword is known, a query can be formulated using phrase and proximity querying. These two methods can improve the search efforts by eliminating documents that do not satisfy this stricter form of multiple-term query.

A phrase is defined as any set of words that appear in a specific order. In a multiple-term query, the order of the querying keywords is not important because search engines evaluate each keyword individually. In contrast, in a phrase search, the order of the querying keywords is important. Typically, querying phrases are enclosed inside quotation marks (" "). They can be used to search for particular sentences, for example, "I have a dream."

In proximity querying, the user can specify a sequence of terms (words or phrases) and a maximum allowed *distance* between any two terms. This distance can be measured in characters or words, depending on the index. For example, the AltaVista search engine uses an adjacency operator near to denote the textual closeness of two terms. The retrieved documents must contain the two terms, which must be within ten words of each other. Proximity querying is helpful in, for example, searching for names because first and last names may be separated by middle names and initials. In general, near is useful when the proximity between two words reflects or captures some semantic connection.

2.1.1.4 Natural Language Query A major problem search services face is the complexity of the querying interface, which is often too complicated for naive users.

Formulating a Boolean query is difficult and frustrating for many people. As a result, systems such as AskJeeves [9] and ElectricMonk [49] have developed natural language interfaces. These services do an impressive job of getting people to find what they want by prompting users to form their own questions. Questions like: "Where can I find a digital camera?" and "Which models of cars are most popular?" can be posed and very accurate responses provided.

The secret to the accuracy of AskJeeves is human intervention. A team of developers creates the knowledge base of questions that powers the search engine. Currently, there are more than seven million questions in the AskJeeves knowledge base. Popular questions that are very frequently asked have prebuilt answers targeted for those questions including lists of subsequent questions useful to narrow the search. If AskJeeves cannot find a match of the asked question in its knowledge base of questions, it falls back on its meta-search engine component to retrieve various search engine results as a backup.

Natural language query is the trend of future information retrieval systems since it is more intuitive for casual users to formulate search criteria in natural language. Strictly speaking, search engines with such a capability should not be categorized as keyword-based ones; instead they should be considered as natural language search engines.

2.1.1.5 Pattern Matching Query In a pattern matching query, the objective is to retrieve all documents that match a given pattern. In this case, it is often difficult to rank results because the given pattern might not contain exact words.

Search engines like AltaVista provide a way to broaden a search by using *wildcards*. Wildcards are simply placeholders for missing characters. The main idea is to allow the missing characters, represented by an asterisk ($*$), to match an arbitrary

character(s), in order to increase the number of related hits. For example, `colo*r` will match both *color*, the American spelling, and *colour*, the British spelling.

Restrictions are placed on wildcards because they can broaden a search excessively. AltaVista implements the following two guidelines on wildcards usage:

- Use wildcards only after three or more characters.
- Use wildcards as placeholders for up to 5 unknown characters.

Some systems allow searching using regular expressions [72]. Enhanced regular expressions [150] extend the expressive power of regular expressions. They incorporate constructs like character class, conditional expression, wildcards, and exact and approximate matching operations to make patterns more powerful.

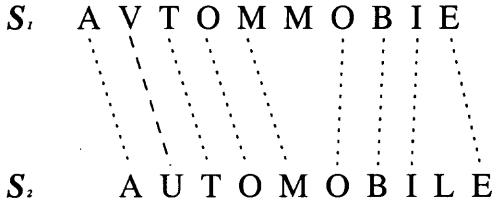


Figure 2.2 String edit operations.

One definition of approximate matching is based on the concept of edit distance. Given two strings S_1 and S_2 the *edit distance* between S_1 and S_2 is defined as the minimum number of *edit operations* (*delete*, *insert* and *relabel*) needed to transform one to the other as illustrated in Figure 2.2. In this figure, three edit operations are required to transform S_1 to S_2 (via relabeling V to U, deleting M, and inserting L). Algorithms for finding the edit distance between two strings can be found in [131]. Approximate text matching over hypertext, on the Web, can be found in [6, 110, 150].

Since matching regular and extended regular expressions is computationally expensive, it is not implemented on a large scale for the Web and is not widely

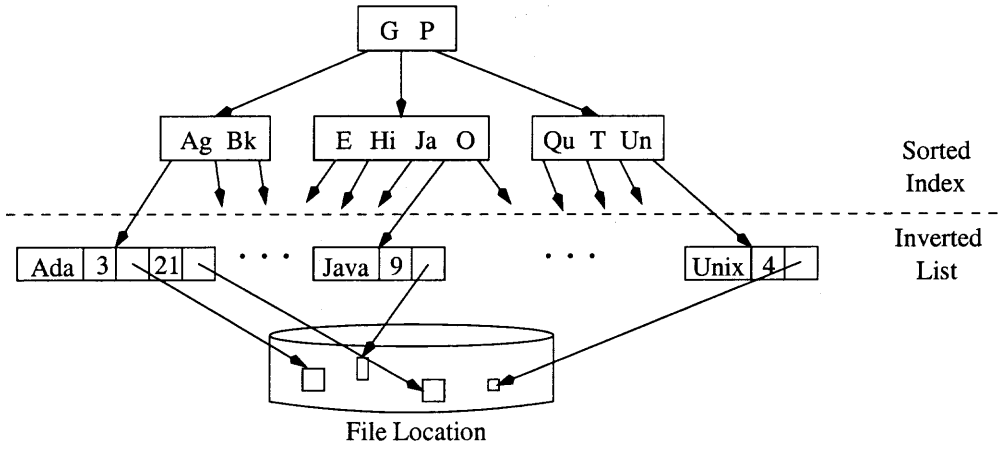


Figure 2.3 An inverted file built based on a B-tree.

adopted by major search engines. However, approximate matching is implemented in WebGlimpse [96], which is based on Glimpse [97].

2.1.2 Search Index

The *search index*, or simply *index*, is the heart of a search engine. The index is typically a list of keywords; each keyword is associated with a list of corresponding documents that contain the keyword. In order to provide fast response time for thousands of concurrent users and be able to store large volumes of data, most search engines use inverted file indexing.

The search index is used because sequential string search algorithms [4, 20, 76, 77] are impractical when the text collection is large, especially if searching is a very frequently performed operation, as it is in systems such as database servers and library information systems. The most widely used method to speed up text search is to prebuild an index. Building and maintaining an index when the amount of text is large and dynamic is not an easy task. Therefore, various indexing techniques, including inverted files [8], suffix arrays [66, 95], suffix trees [4, 101, 144] and signature files [53, 54], have been developed and studied. Most search engines use inverted files because they are easier to maintain and to implement.

An inverted file is an enormous, word-oriented, look-up table. It contains two major parts: a *sorted index* (list) of keywords (vocabulary) and an *inverted list* that stores a list of pointers to all occurrences of a keyword as illustrated in Figure 2.3. In this figure, the internal nodes represent the index structure, while the leaf nodes contain the indexed words with their occurrence position within the file and pointers to different file locations. In the case in which the inverted index is used for the Web, the locations referred to are URLs. Inverted files can be implemented using a variety of data structures. The sorted index (list) can be built using sorted arrays, hash tables, B-trees [14, 15], tries [21], or a combination of these.

Different search engines use different inverted file indexing schemes. The *granularity* of an index is defined as the accuracy to which the index identifies the location of a search keyword. In general, indexes can be classified into the following three categories:

- *Coarse-grained* – able to identify a set of documents based on a search keyword.
- *Medium-grained* – able to identify a specific document based on a search keyword.
- *Fine-grained* – able to identify a sentence or word location in a specific document based on a search keyword.

Each granularity classification has different storage requirements and precision, as summarized in Table 2.2.

Type	*Space Requirement*	*Access Speed*	*Accuracy*
coarse-grained	low	low	low
medium-grained	medium	medium	high
fine-grained	very high	very fast	exact

Table 2.2 Granularity of inverted file indexes.

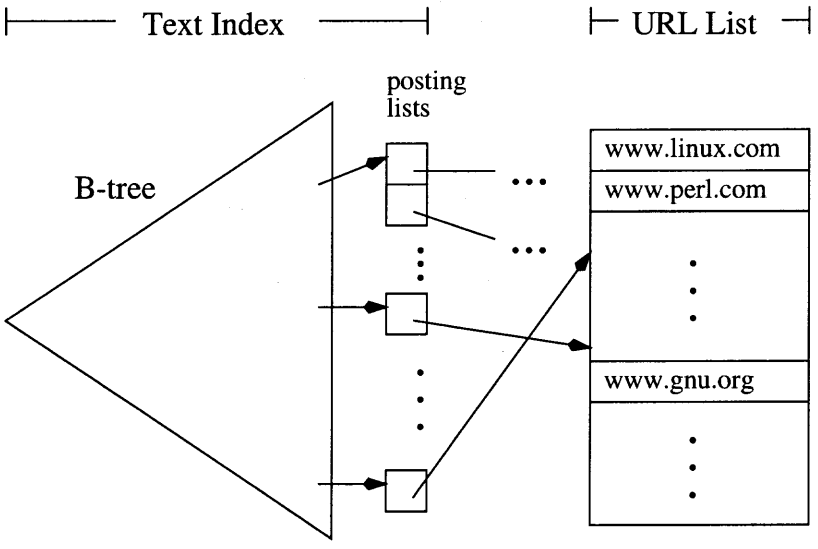


Figure 2.4 Index structure of a Web search engine.

In the case of Web search engines, instead of storing the locations of keyword occurrences, the search index stores the URLs of the occurrences, as illustrated in Figure 2.4. The triangular shape in Figure 2.4 represents a B-tree index. The listing of URLs indexed by the B-tree is shown in a table on the right of the figure. Each posting list refers to a list of pointers to those URLs where a particular keyword occurs, with one posting list per keyword. The order of the URLs in a posting list for a keyword depends on an internal ranking mechanism based on criteria such as word frequency count and word weight, which are used to measure the importance of a word within a document.

The Web has become a global information resource, with obvious consequences for the size of the index. In order to reduce the size of the index, the following kinds of transformation techniques are used in the index building process. Index compression techniques are also applied to represent indexes more compactly.

- *Case folding* – converts everything to lower case. For example, `Data Mining` becomes `data mining`.

- *Stemming* – reduces words to their morphological root. For example, `compression` and `compressed` become `compress`.
- *Stop word removal* – removes common or semantically insignificant words. For example, the definite article, `the`, and indefinite articles, `a` and `an` in English, are removed.
- *Text compression* – reduces the inverted file size.

Different search engines apply the transformation methods differently. The methods are *domain* and *language* specific. For example, "SAT" and "Sat", and "US" and "us" have different meanings in English. Therefore, modifications to these methods are made to reflect different domains and languages. More detailed information on document compressing and indexing is discussed in [148].

2.1.3 Web Crawlers

Web crawlers, also known as *agents*, *robots* or *spiders*, are programs that work continuously behind the scene, having the essential role of locating information on the Web and retrieving it for indexing.

Crawlers run continuously to ensure an index is kept as up to date as possible and to achieve the broadest possible coverage of the Web. However, since the Web is constantly changing and expanding, no search engine can feasibly cover the whole Web. Indeed, many studies that have been conducted to estimate the coverage of search engines employing crawlers, show that coverage ranges between only 5% to 30% [86] and the union of 11 major search engines covers less than 50% of the Web [87].

Claiming broader coverage of the Web is one way of demonstrating the superiority of an index. Search engine firms tend to use the extent of their coverage

to boast about their indexing technology. However, broader coverage does not by itself guarantee higher accuracy. Most search engines attempt to maximize *recall*, a figure of merit used in information retrieval, defined as:

$$recall = \frac{N}{T},$$

where N denotes the number of retrieved documents that are relevant to a search query and T denotes the number of potentially retrievable Web documents that are relevant to the search query. Observe that this is not the ratio of retrieved to relevant documents, which could be considerably greater than 100% since a large number of irrelevant documents could be included in what are retrieved.

Another measure of the effectiveness of search engines is *precision*, defined as:

$$precision = \frac{N}{R},$$

where R denotes the number of documents retrieved in response to the search query. Low precision would indicate that many irrelevant or superfluous documents are retrieved, while low recall would indicate that the fraction of potentially relevant documents retrieved is low. Thus, recall and precision tend to be inversely related: when recall is high, precision tends to be low, as illustrated in Figure 2.5.

Although there are many publicly available search engines, the details of how specific indexes are organized remain a strategic business secret. The in-depth coverage on crawlers will be discussed in Chapter 6.

2.2 Web Directories

Search engines such as Northern Light [113], Direct Hit [44], Inktomi [74], FAST Search [56], create their index automatically using crawlers. On the other hand, Yahoo! [151] depends on humans to create its listing, called a *directory*. Directories are created by using Website descriptions submitted by various sites or generated by human editors. Table 2.3 illustrates directory categories listed on

• Arts & Humanities	• News & Media
• Business & Economy	• Recreation & Sports
• Computer & Internet	• Reference
• Education	• Regional
• Entertainment	• Science
• Government	• Social Science
• Health	• Society & Culture

Table 2.3 Yahoo! directory categories.

Yahoo! directory main page. Similar category structures can be found at other search engines, like Excite [52], Netscape [111], and Go [63]. Although directories and indexes work differently on the backend, the querying interface frontends work similarly.

2.3 Meta-Search Engines

Search engines provide fast retrieval of information of interest from the Web. However, the problem of knowing where search engines are and how to use them poses some difficulties. Furthermore, empirical results indicate that only 45% of relevant results will likely be returned by a single search engine [129], that is, each

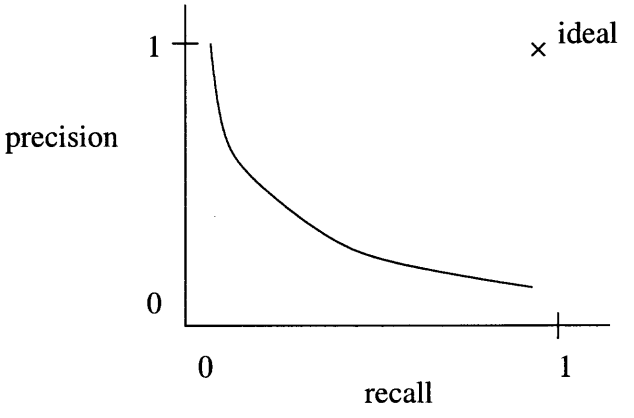


Figure 2.5 Recall-precision curve.

has a recall rate of 45%. This limitation is compounded by the fact that the coverage of a typical search engine is between only 5% – 30% of the Web [86].

Meta-search engines are designed to mitigate such problems by accessing multiple individual search engines. The principle behind meta-search engines like MetaCrawler [105], SherlockHound [132], SavvySearch [125], and Inquirus [85], is simple: "A dozen search engines is better than one". The system architecture of MetaCrawler is discussed in [46, 130] and an improved meta-search architecture has been presented and studied in [62]. Figure 2.6 illustrates the system architecture of a meta-search engine that contains the following components:

- Query Interface Module – responsible for getting user's query input.
- Dispatch Module – responsible for determining to which search engines a specific query is sent.
- Knowledge-base Module – used by the Dispatch Module to perform decision-making (optional).
- Interface Agents Module – responsible for interacting with different search engines using different query formats.
- Evaluation Module – responsible for ranking results according to some predefined evaluation methods (optional).
- Display Module – responsible for displaying results.

2.4 Information Filtering

An important tool which is complementary to the search engine approach is information filtering [13]. Filtering refers to the process of determining whether a document is relevant to search criteria or not, and eliminating irrelevant documents.

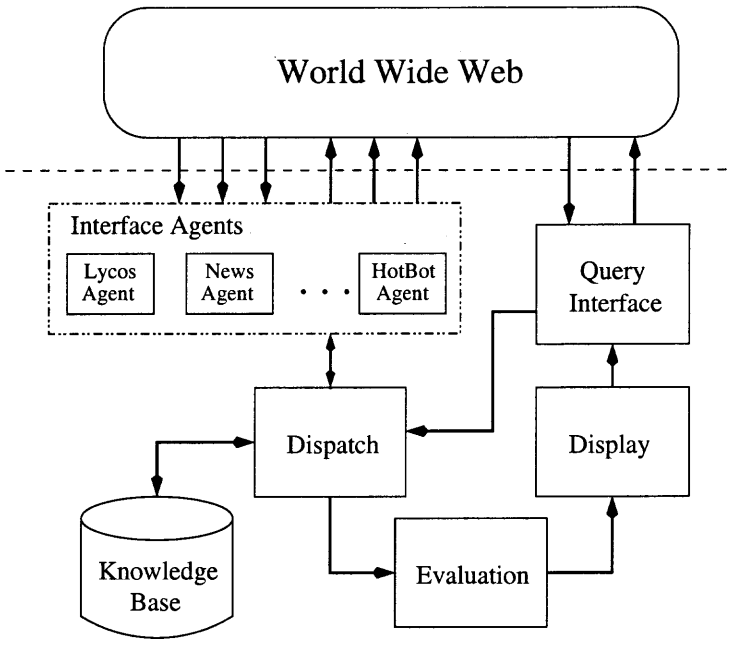


Figure 2.6 The architecture of a meta-search engine.

Filtering applications usually involve a stream of incoming data such as newswire and email. Filtering is also used to describe the process of accessing and searching for information on remote servers using intelligent agents [16].

Information filtering, which is based on a combination of machine learning and information retrieval techniques [30, 39, 40, 59, 83, 100, 112, 126], has been employed in many specialized search engines. Information filters can be viewed as mediators between information sources and target systems. They help systems eliminate irrelevant information using intelligent decision-making techniques like Bayesian classifiers, term-frequency analysis, k-neighbors, neural networks, rule-learning, etc.

Information filtering tools can be used to build topic-specific search engines. General-purpose search engines, such as HotBot [73], offer high levels of Web coverage for general information search on the Web. However, topic-specific search engines are growing in popularity because they offer an increased precision/recall rate. Examples of topic-specific search engines are: DejaNews [41], which specializes in Usenet news

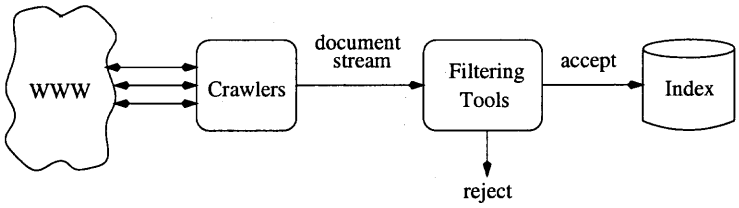


Figure 2.7 The building process of a topic-specific search engine.

articles; BioCrawler [18], which is a directory and search engine for biological information search; and Cora [38], which allows one to search for computer science research papers in PostScript format from universities and labs all over the world.

A general process for building a topic-specific search engine is illustrated in Figure 2.7. In this figure, the agents which are responsible for retrieving documents from the Web are crawlers. Documents returned by crawlers are streamed to information filtering tools, which decide the relevance of the documents to the topic domain. Accepted documents are added to the search index, while rejected documents are discarded.

CHAPTER 3

QUERY-BASED SEARCH SYSTEMS

A shortcoming of the keyword-based search tools discussed in Chapter 2 is the lack of high-level querying facilities available to facilitate information retrieval on the Web. Query languages have long been part of database management systems (DBMSs), the standard DBMS query language being the Structural Query Language (SQL) [60, 79, 118]. Such query languages not only provide a structural way of accessing the data stored in a database, but also hide details of the database structure from the user. Since the Web is often viewed as a gigantic database holding vast stores of information, some Web-oriented query systems have been developed. However, unlike the highly structured data found in a DBMS, information on the Web is stored mainly as files. The files can be generally categorized as:

- *Structured*, such as flat databases and BibTeX files.
- *Semistructured*, such as HTML, XML, and LaTeX files.
- *Unstructured*, such as sound, image, pure text and executable files.

Structured files have a strict inner structure. For example, files like BibTeX are highly structured. The grammar of BibTeX precisely defines the syntax and semantics of the data. In this case, the grammar is like the schema of a database.

Semistructured files are text files that contain formatting codes, often called *tags*. Such files include LaTeX, HTML, and XML files. Although tags can be used to specify the semantic information within documents, most of the semantic information is not coded in a formal way. Furthermore, even if the semantics were formally coded, they would not be at a fine-grained level of specification like in a database schema. For example, the `\paragraph` tag in LaTeX and the `<P>` tag in HTML

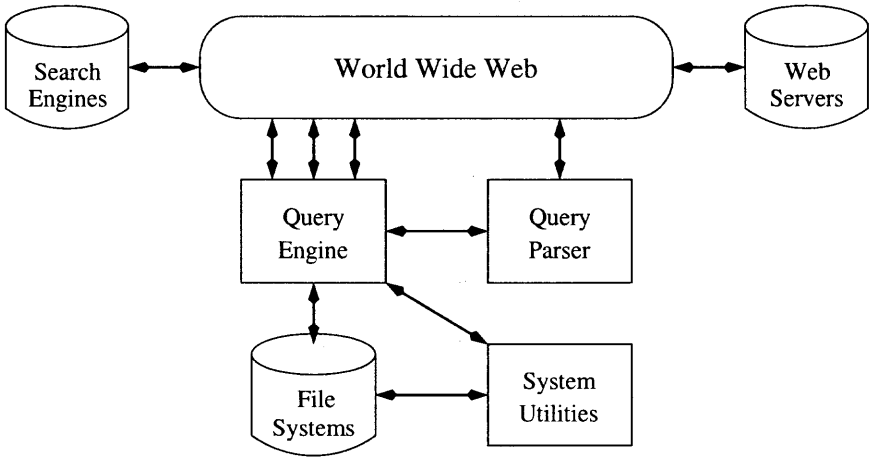


Figure 3.1 Web querying system architecture.

documents are general tags but say nothing about the content of a paragraph. XML provides a finer grained specification than HTML by allowing user-defined tags, but the lack of data type support and querying facilities has led to the further development of XML-data [88], XML-QL [42, 57], and the Niagara Internet Query System (NiagaraCQ) [33].

Unstructured files include sound, image and executable files that are not text based. This makes it difficult to apply database techniques such as querying and indexing to them.

Web query languages that have been developed and that allow high-level querying facility on the Web include W3QS/W3QL [78], WebSQL [104], WAQL [70], and WebLog [82]. These systems allow the user to pose SQL-like queries on the Web, with the exception of WebLog which uses a logic-like query format. Unlike the query facilities provided in mediators to be described in Chapter 4, these query systems interact directly with the Web with minimal interaction with a local DBMS or a file system. The general architecture of these query systems is illustrated in Figure 3.1. It consists of a Query Parser module, a Query Engine module, and system utilities. Query Parser takes a query and sends it to Query Engine. Query Engine interacts with search engines, Web servers, system utilities and file systems.

W3QS/W3QL, WebSQL and WAQL are discussed in this chapter, and detailed discussion on WebLog can be found in [82].

3.1 W3QS/W3QL

W3QS [78] was one of the first Web querying systems. The system was designed to provide:

- a high-level SQL-like query language,
- a simple interface to external programs and Unix tools,
- a graphical user interface for the information gathered by queries, and
- a higher view maintenance facility than robot-maintained indexes.

A query language called W3QL was subsequently designed as part of the W3QS system. W3QL is an SQL-like language that allows querying based on the organization and HTML content of the hypertext. It emphasizes extensibility and the ability to interact with user-written programs, Unix utilities, and tools for managing Web forms. Its ability to interact with external programs makes W3QL like a command-driven query language.

W3QL views the Web as a directed graph. Each URL or document corresponds to a node of the graph. Each node has associated properties according to its document type and content. For example, a node corresponding to an HTML file has an HTML format, a URL, a Title, etc. On the other hand, a node corresponding to a LaTeX file is of LaTeX format and might have Author and Title associated with it. A directed edge is drawn from node a to node b if node a is a node with HTML or XML format and the document contains at least one anchor (hyperlink) to node b.

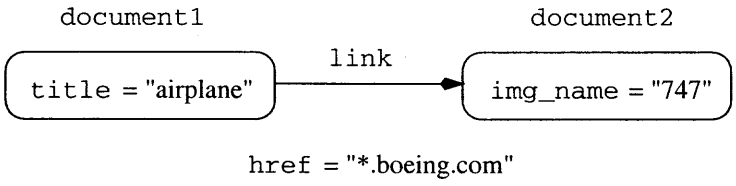


Figure 3.2 Structure-based query.

Like nodes, edges also have associated properties. For example, consider Figure 3.2 which illustrates a structure-based path pattern query. This query is to find documents with "airplane" in their title and with a hyperlink to Boeing's Website with an image on 747. The `<A>` tag which specifies the hyperlink corresponds to a directed edge with attributes such as `HREF` and `REV`. Since there may be more than one hyperlink between a pair of nodes, the Web actually corresponds to a labelled directed multigraph.

W3QL provides two types of hypertext query: *content*-based and *structure*-based. In content-based queries, W3QL uses a program called `SQLCOND` to evaluate Boolean expressions in the query. This is similar to the `WHERE` clause of an SQL query. Using `SQLCOND`, a user can select nodes from the Web that satisfy certain conditions. For example, the conditions might be:

```
(node.format = HTML) and (node.title = "database").
```

In a structure-based query, the query can be specified using a *path pattern* (*path expression*). A *path* is a set of nodes $\{n_1,\ldots,n_k\}$, and node pairs (n_i, n_{i+1}), $1 \leq i \leq k-1$, where (n_i, n_{i+1}) are edges of the graph.

The query patterns are a set of subgraphs of the Web in graph representation. Each subgraph must satisfy conditions in the query pattern including node and edge conditions specified by the query pattern.

W3QS/W3QL has three main modules - a Query Processor, a Remote Search Program (RSP), and a Format Library. The Query Processor takes a query and invokes

programs in the RSP library to interact with search engines. At the end of the search, the Format Library is used to display the result.

The semantics of a W3QL query is defined as follows:

SELECT	α
FROM	β
WHERE	ω
USING	ψ
EVALUATED EVERY	γ

where α specifies how results should be processed; β specifies the pattern graph of the search domain; ω specifies a set of conditions on the nodes and edges specified in β; ψ and γ optionally specify the search methods and search interval, respectively.

3.1.1 Select Clause

The SELECT clause is an optional part of the query and defines the form in which the query results should be processed. It has the following form:

```
SELECT [CONTINUOUSLY] statements.
```
(3.1)

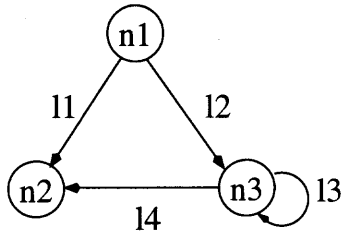
The `statements` are UNIX programs and arguments that are invoked after the results are returned from the RSP. This clause may also involve running a UNIX program to process the result. It can also act as a filter to reformat the result.

3.1.2 From Clause

The FROM clause describes the virtual pattern graph the RSP uses to search for results. Patterns are described using a set of paths according to the following format:

```
FROM path_expressions.
```
(3.2)

The `path_expressions` specify the search pattern to be used by the RSP. A pattern graph is a directed graph $G(V, E)$ where there is at most one edge between any two nodes and at most one self-loop. Figure 3.3 illustrates a pattern graph with its corresponding text description.



n1 l1 n2
n1 l2 (n3 l3)
(n3 l3) l4 n2

or

n1 l1 n2
n1 l2 (n3 l3) l4 n2

Figure 3.3 Example of a pattern graph.

3.1.3 Where Clause

The `WHERE` clause is used to specify search predicates that nodes and edges in the `FROM` clause must satisfy. Only nodes satisfying the `WHERE` conditions will be returned as part of the query results. The `WHERE` clause has the following form:

```
WHERE condition_clauses.
```
(3.3)

The `condition_clauses` impose search restrictions on the nodes and edges specified in the `FROM` clause. It can also provide RSP navigation directives. The following are conditions that can be specified:

- `unix_program args = reg_expression.` This will invoke external UNIX programs.
- `node_name IN file_name.` This will check whether the node name `node_name` is in the file `file_name`.
- `FILL node_name AS IN file_name WITH assignments.` This will automatically fill in the form in `node_name` with data found in `file_name` and `assignments`.
- `RUN unix_program IF node_name UNKNOWN IN file_name.` This will automatically invoke an external program if an unknown form is encountered during the search process.

3.1.4 Using clause

The `USING` clause is optionally used to specify the search algorithm to be performed by the RSP. Search methods like depth first search and breadth first search can be specified. If the algorithm is not specified in a query, a default search algorithm is used.

3.1.5 Evaluated every clause

The `EVALUATED EVERY` clause is optionally used to specify the updating interval for the query. Intervals such as daily and weekly can be specified for automatic updates. This clause is mainly used for dynamically maintained views.

3.1.6 Examples

Example 1. This query searches for HTML documents that contain hyperlinks to images in GIF format located in `www.xyz.com`. Here `n1` and `n2` are nodes and `l1` is the link used to connect `n1` with `n2`. The `select` statement is a command that copies the content of `n1` to a file called result.

```
select  cp n1/* result;
from    n1, l1, n2;
where   SQLCOND (n1.format = HTML) AND
        (l1.href = "www.xyz.com") AND
        (n2.name = "*.gif");
```

Example 2. This query defines a view which maintains a list of pointers to LaTeX articles with "Data Mining" in the title. `URLlist.url` specifies a built-in list of important search indexes. `URLlist.fil` specifies how to fill in the keyword "Data Mining" in HTML forms required by search indexes in `URLlist.url`. The `select` statement specifies that the URL of `n2` should be printed continuously.

```
select  CONTINUOUSLY SQLPRINT n2.URL;
from    n1, l1, n2;
where   n1 IN URLlist.url;
        FILL n1.form AS IN URLlist.fil WITH keyword="Data Mining";
        SQLCOND (n2.format = "LaTeX") AND
        (n2.title = "Data Mining");
```

3.2 WebSQL

WebSQL [104] is another SQL-like query language in the spirit of W3QL which represents an effort towards greater formalization. It introduces some interesting ideas including:

- a high-level SQL-like query language,
- a querying interface implemented in Java,
- a new theory of query cost based on "query locality,"
- the ability to use multiple index servers without explicit user knowledge, and
- the ability for Web maintenance.

WebSQL is also based on a graph model of a document network. It views the Web as a "virtual graph" whose nodes are documents and whose directed edges are hyperlinks. To find a document in this graph, one navigates starting from known nodes or with the assistance of index servers. Given a URL u, an agent can be used to fetch all nodes reachable from u by examining anchors in the document contents. Conversely, nodes that have links to u can be found by querying index servers for nodes that contain links to node u, using `link:url` search format in the AltaVista [5] search engine.

The semantics of a WebSQL query is defined using, for example, selections and projections. A query has the form:

$$\begin{array}{ll} \texttt{SELECT} & \alpha \\ \texttt{FROM} & \beta \\ \texttt{WHERE} & \omega \end{array}$$

where α specifies attributes of documents that satisfy a given predicate; β specifies the domain conditions of the attributes; and ω specifies a set of Boolean expressions that a query must satisfy.

3.2.1 Select Clause

The SELECT clause is used to list the attributes desired for the result of a query. It has the form:

$$\texttt{SELECT Ident.}attrib_1\texttt{, ...,Ident.}attrib_n, \tag{3.4}$$

where $\texttt{Ident.}attrib_i$ are identifer and attribute pairs. The attributes can be any combination of the following:

- `url` – for retrieving the Uniform Resource Locator.
- `title` – for retrieving the title of the document.
- `text` – for retrieving the actual hypertext.
- `type` – for retrieving specific document format.
- `length` – for retrieving the length of a document.
- `modif` – for retrieving the last modification date.

3.2.2 From Clause

The `FROM` clause is used to list the domain conditions for the attributes desired for the result of a query. It has the following form:

```
FROM DocType Ident SUCH THAT DomainCondition,          (3.5)
```

where `DocType` specifies the document or structure type such as *document, anchor, sound, image*; `Ident` is a tuple variable; and `DomainCondition` specifies the condition of the domain based on the following forms:

- Node *PathRegExp* TableVariable
- TableVariable MENTIONS StringConstant
- Attribute = Node

PathRegExp is used to specify a path regular expression based on hyperlink structures, where a hyperlink can be one of the following types:

- *Interior* – if its target is within the source document;
- *Local* – if its target is a different document on the same server;
- *Global* – if its target is located on a different server.

Arrow-like symbols are used to denote the three link types; thus, $\mapsto$ denotes an interior link, $\rightarrow$ denotes a local link, and $\Rightarrow$ denotes a global link. In addition, $=$ denotes the empty path. Using these three link types, one can build path regular expressions using concatenation, alternation (|) and repetition (*).

3.2.3 Where Clause

The `WHERE` clause is used to specify search predicates that documents returned from the `FROM` clause must satisfy. Only documents satisfying the `WHERE` conditions will

be returned as part of the query results. It has the following form:

$$\texttt{WHERE } Term_1 \ \mathcal{LC} \ \ldots \ \mathcal{LC} \ Term_n. \tag{3.6}$$

The search predicates are Boolean expressions consisting of terms connected by logical connectives ($\mathcal{LC}$), with *and* denoting intersection and *or* denoting union.

3.2.4 Examples

Example 1. This query is used to search for HTML documents about "mining". It returns the URL of the documents in the tuple d.

```
select  d.url
from    Document d
        SUCH THAT d MENTIONS "mining"
where   d.type = "text/html";
```

Example 2. This query is used to search for documents about "mining" that contain a hyperlink to www.kdd.org. It returns the URL and title of the documents in the tuple d.

```
select  d.url, d.title
from    Document d
        SUCH THAT d MENTIONS "mining",
        Anchor y SUCH THAT base = d
where   y.href = "www.kdd.org";
```

Example 3. This query is used to search for documents with "Web mining" in the title, which are linked from a hyperlink path originating at www.kdd.org, of length two or less, and located on the local server. It returns the URL and title of the documents in the tuple d.

```
select  d.url, d.title
from    Document d
        SUCH THAT "http://www.kdd.org" = | → | → → d
where   d.title = "Web mining";
```

3.3 WAQL

WAQL [70] (Web-based Approximate Query Language) is an SQL-like language similar to W3QL and WebSQL. It can interact with both search engines and Websites and automate the search/gathering process. It acts like an agent that controls the information searching process according to the user's specification. It was designed to provide:

- a high-level SQL-like query language,
- a structural search mechanism based on document structure,
- approximate search based on edit distance, and
- approximate search based on variable length don't cares.

The semantics of a WAQL query has the form:

```
SELECT  α
FROM    β
USING   γ
WHERE   ω
```

where α specifies attributes of documents that satisfy a given predicate; β specifies which Website to search; γ specifies which index servers should be used; and ω specifies a set of Boolean expressions that a query result must satisfy.

The BNF specification of the language syntax is given in Figure 3.4. Expressions inclosed between '{' and '}' means zero or more repetitions of a construct. Expressions inclosed between '[' and ']' are optional. All bold and lower case words are reserved words. Words with first character capitalized are terminals:

<query>	::=	**select** <attribs> [**from** <website_list>] [**using** <index_list>] **where** <boolean_exp>;
<attribs>	::=	<attrib> {, <attrib>}
<attrib>	::=	<doc_ptr>.<field> {.<field>}
<field>	::=	**url** | **title** | **size** | **text** | **modif** | **dist** | ϵ
<website_list>	::=	<urls> <doc_ptr> <traversal_method>
<urls>	::=	<url> {, <url>}
<index_list>	::=	<search_engine_list> <doc_ptr> [<target_range>]
<search_engine_list>	::=	<engine_name> {, <engine_name>}
<engine_name>	::=	***AltaVista*** | ***Excite*** | ***Infoseek*** | ***Hotbot***
<traversal_method>	::=	<link_type> <depth_level>
	|	<link_type> <target_range>
	|	<link_type> <depth_level> <target_range> | ϵ
<boolean_exp>	::=	<boolean_exp> { **and** <boolean_exp> }
	|	<boolean_exp> { **or** <boolean_exp> }
	|	(<boolean_exp>)
	|	<containment_exp>
	|	<link_constraint>
<containment_exp>	::=	<doc_ptr> **mentions** "<string>" [<dist_exp>] {, "<string>" [<dist_exp>] }
	|	<doc_ptr> **contains** "<reg_exp>" {, "<reg_exp>" }
	|	<doc_ptr> **has** <tree> [<dist_exp>]
<link_constraint>	::=	<doc_ptr> -> <doc_ptr> { -> <doc_ptr> } | ϵ
<dist_exp>	::=	**with dist** <op> *Integer* | ϵ
<link_type>	::=	-> | => | +> | |> | ||> | |||>
<op>	::=	= | > | < | >= | <=
<target_range>	::=	*Integer* | *Integer* .. *Integer*
<reg_exp>	::=	*Reg_Exp*
<string>	::=	*String*
<tree>	::=	*Tree*
<url>	::=	*URL*
<depth_level>	::=	*Integer*
<doc_ptr>	::=	*Ident*

Figure 3.4 Grammar for WAQL.

Ident - Identifier (ex: a, b, doc1,...).

Integer - Integer (ex: 0, 1, 2,...).

Url - Uniform Resource Locator (ex: http://www.ibm.com, http://www.linux.org,...).

String - String (ex: "+database", "-object",...).

Reg_Exp - Extended Regular Expression (ex: "relational.*object").

Tree - Hierarchical query tree structure.

3.3.1 Select Clause

The `SELECT` clause is used to list the attributes desired as the result of a query. It has the following form:

$$\texttt{SELECT Ident.}attrib_1\texttt{, ...,Ident.}attrib_n, \tag{3.7}$$

where `Ident.`$attrib_i$ are identifer and attribute pairs. The attributes can be any combination of the following:

- `url` – for retrieving the Uniform Resource Locator.
- `title` – for retrieving the title of the document.
- `size` – for retrieving the size of the document.
- `text` – for retrieving the actual hypertext.
- `modif` – for retrieving the last modification date.
- `dist` – for specifying the distance between query and actual document.

3.3.2 From Clause

The `FROM` clause is used to specify which Website to contact and retrieve documents from. It has the following form:

```
FROM URL {,URL} Ident traversal_method [target_range].        (3.8)
```

The `FROM` clause is used to perform a site-oriented search that traverses the entire Website. The `URL` field specifies the address of a target site. The `Ident` field is a pointer to each document returned by the traversal. Let I_1 and I_n denote integer type. When the `target_range` field has the form I_n, it informs the query processor to process the 1st to the I_nth URL; when it has the form $I_1..I_n$, it informs the query processor to process the I_1th to the I_nth URL.

When performing a Website oriented search, hyperlinks (URLs) encountered within hypertext documents during the navigation are of the following types:

- *Local-links*, which are hyperlinks that link to the same domain.
- *External-links*, which are hyperlinks that link to different domains.
- *All-links*, which include both local and external links.

The traversal methods, which can be performed using any specified combinations of these different hyperlink types, are:

- *Depth first* – Documents are retrieved in the depth first search order with respect to the hyperlinks.
- *Breadth first* – Documents are retrieved in the breadth first search order with respect to the hyperlinks.

Table 3.1 shows a list of operators that specify hyperlink types and their respective traversal methods.

Operator	*Definition of Operator*
$->$	Breadth first traversal only on local-link.
$=>$	Breadth first traversal only on external-link.
$+>$	Breadth first traversal on all-links.
$\|>$	Depth first traversal only on local-link.
$\|\|>$	Depth first traversal only on external-link.
$\|\|\|>$	Depth first traversal on all-links.

Table 3.1 Operators specifying the traversal method on a specific link type.

3.3.3 Using Clause

The `USING` clause is used to specify which index server to contact for an initial search. It has the following form:

```
USING search_engine Ident [target_range].
```
(3.9)

The `search_engine` field specifies which index server to request for the target documents. WAQL provides interfaces to five publicly available search engines: AltaVista, Excite, Hotbot, Infoseek, and Lycos.

`Ident` is a tuple variable that acts like a pointer to each document returned by the search engines. The `target_range` field has the same interpretation as for the `FROM` clause.

3.3.4 Where Clause

The `WHERE` clause is used to specify search predicates that documents returned from the `FROM` clause must satisfy. Only documents satisfying the `WHERE` conditions will be returned as part of the query results. It has the following form:

$$\texttt{WHERE}\ Term_1\ \mathcal{LC}\ \ldots\ \mathcal{LC}\ Term_n. \tag{3.10}$$

The predicates have the same overall syntax as in WebSQL. Let `String` denote string, `Reg_Exp` denote a regular expression, and `Tree` denote the query tree structure in preorder string format (WAQL represents each document by a tree structure).

A term has one of the following four forms:

$$D \texttt{ mentions String [with dist } OP \ k \texttt{]}, \tag{3.11}$$

$$D \texttt{ contains Reg\_Exp}, \tag{3.12}$$

$$D \texttt{ has Tree [with dist } OP \ k \texttt{]}, \tag{3.13}$$

and

$$D_1 \texttt{ -> } D_2 \texttt{ -> } \ldots \texttt{ -> } D_n. \tag{3.14}$$

Terms of the form (3.11) are used to specify the exact and approximate string matching requirements that resulting documents must satisfy. D is a document pointer that will contain the location of the actual document where string matching will be applied. OP is a comparison operator ($>$, $<$, $=$), and k is an integer that specifies the distance allowed for approximate string matching. If k is not specified, a default value of 0 is assumed, which means exact string matching is performed.

Terms of the form (3.12) are used to specify the regular expression matching requirements that resulting documents must satisfy. D is a document pointer containing the location of the actual document where regular expression matching will be applied.

Terms of the form (3.13) are used to specify a hierarchical search query. Once again, D is a document pointer to the location of the actual document where regular expression matching will be applied. OP is a comparison operator, and k is an integer value that specifies the distance allowed for the approximate tree matching with variable length don't cares (VLDCs). That k equals 0 corresponds to exact-match retrieval, which means the query tree structure must be totally embedded in the document tree structure. That is, a `Tree` T of distance k away from document D is embedded in D if and only if k equals 0. When k is positive, the retrieval is considered to be approximate.

Terms of the form (3.14) are used to specify link constraints. For example, if a term A–>B–>C is specified then the query requires that document A has a hyperlink to document B and document B has a hyperlink to document C.

In general, terms are not limited to just these four types, but can be any function that resulting documents must satisfy, such as the result of the application of an image or voice recognition algorithm. Thus the query language is flexible and can be extended to incorporate new functions.

3.3.5 Examples

Example 1. This query is to find documents containing "database", "oracle", and "relational" using Altavista as the index server. The resulting documents' URLs are returned.

```
select  d.url
using   Altavista d
where   d mentions "database", "oracle", "relational";
```

Example 2. This query is to find documents containing "database" in an H1 tag and a regular expression "object.*relational" in a paragraph using Excite as the index server. This regular expression means the word "object" followed by the word "relational" with a VLDC between these two specified words. Thus, the * matches a string of characters of arbitrary length. The resulting documents' URLs are returned.

```
select  d.url
using   Excite d
where   d has (*(H1("database")(P(*("object.*relational")))));
```

Example 3. This query is to find documents that have the word "database" in an H1 tag, followed by a paragraph containing "oracle", using Infoseek as the index server, and examines only the 23rd to 69th URLs from the list of returned URLs. Most importantly, each matched document must be at most zero or unit distance away

from this hierarchical query pattern in the edit distance sense defined in [153]. The resulting documents' URLs are returned.

```
select  d.url
using   Infoseek d [23..69]
where   d has (*(H1("database"))(P(*("oracle")))) with dist <= 1;
```

Example 4. This query is to find documents that have the word "RS6000", and have "deep blue" in the title followed by "kasparov" in a paragraph, where the search domain is www.ibm.com. The method of search will be depth first search down to the third level in the Web tree structure.

```
select  d.url
from    http://www.ibm.com d ||> 3
where   d mentions "RS6000" and
        d has (*(title("deep blue"))(P("kasparov")));
```

Example 5. This query is to find books that mention "Ontos", have "database" in the book title, and put an emphasis on "object database". The author field in the resulting documents is returned. This query is actually targeting XML documents because user-defined tags are used.

```
select  d.author
using   index_server d
where   d mentions "Ontos" and
        d has (Book(title("database"))emph("object database"));
```

3.3.6 Performance Evaluation

In order to test the effectiveness of WAQL some experiments are performed on approximate query. All queries have been tested with four major index servers (AltaVista, Excite, Hotbot, and Infoseek.) There were several thousand URLs returned by each search engine. To restrict the test set to a manageable number, only the first one hundred URLs returned by each search engine are proceeded by WAQS. Distance between query pattern and HTML documents ranging from 0 to 4 is recorded for each query, where distance 0 indicates an exact match. Column u of

table 3.2 indicates the number of URLs that was unable to retrieve, parsing error, and other network problems not uncommon on the Web.

Test 1. Search for documents containing "database" in H1 tag as illustrated in Figure 3.5. The result of this query is summarized in Table 3.2.

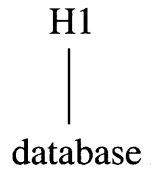


Figure 3.5 H1 contains word "database".

Search Engine	*Dist 0*	*Dist 1*	*Dist 2*	*Dist 3*	*Dist 4*	*u*
AltaVista	21	64	1	0	0	14
Excite	9	82	1	0	0	8
Hotbot	6	82	1	0	0	11
Infoseek	10	70	3	0	0	17

Table 3.2 Results from Test Query Pattern 1.

Test 2. Search for documents containing "database" in P (paragraph) tag as illustrated in Figure 3.6. The result of this query is summarized in Table 3.3.

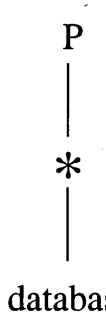


Figure 3.6 P contains word "database".

Search Engine	*Dist 0*	*Dist 1*	*Dist 2*	*Dist 3*	*Dist 4*	*u*
AltaVista	58	26	1	0	0	15
Excite	80	11	1	0	0	8
Hotbot	56	33	1	0	0	10
Infoseek	55	26	3	0	0	16

Table 3.3 Results from Test Query Pattern 2.

Test 3. Search for documents containing "database" in TITLE tag as illustrated in Figure 3.7. The result of this query is summarized in Table 3.4.

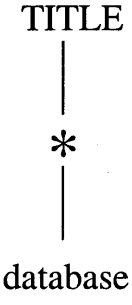


Figure 3.7 TITLE contains word "database".

Search Engine	*Dist 0*	*Dist 1*	*Dist 2*	*Dist 3*	*Dist 4*	*u*
AltaVista	81	4	1	0	0	14
Excite	42	50	1	0	0	7
Hotbot	54	34	1	0	0	11
Infoseek	40	38	5	0	0	17

Table 3.4 Results from Test Query Pattern 3.

Test 4. Search for documents containing "object" followed by "relational" in P tag as illustrated in Figure 3.8. The result of this query is summarized in Table 3.5.

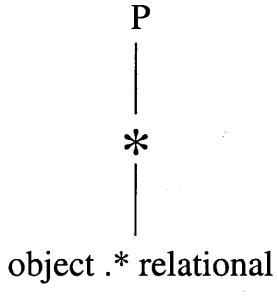


Figure 3.8 P contains word "object" followed by "relational".

Search Engine	*Dist 0*	*Dist 1*	*Dist 2*	*Dist 3*	*Dist 4*	*u*
AltaVista	42	35	0	0	0	23
Excite	71	14	0	0	0	15
Hotbot	41	42	1	0	0	16
Infoseek	55	9	0	0	0	36

Table 3.5 Results from Test Query Pattern 4.

Test 5. Search for documents containing "database" in H1 than "object" followed by "relational" in P tag as illustrated in Figure 3.9. The result of this query is summarized in Table 3.6.

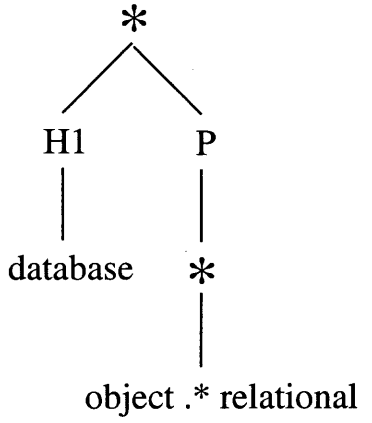


Figure 3.9 H1 contains word "database" and than P contains "object" followed by "relational".

Search Engine	*Dist 0*	*Dist 1*	*Dist 2*	*Dist 3*	*Dist 4*	*u*
AltaVista	3	24	33	20	1	19
Excite	6	43	38	7	0	6
Hotbot	4	18	42	23	3	10
Infoseek	8	32	23	3	1	33

Table 3.6 Results from Test Query Pattern 5.

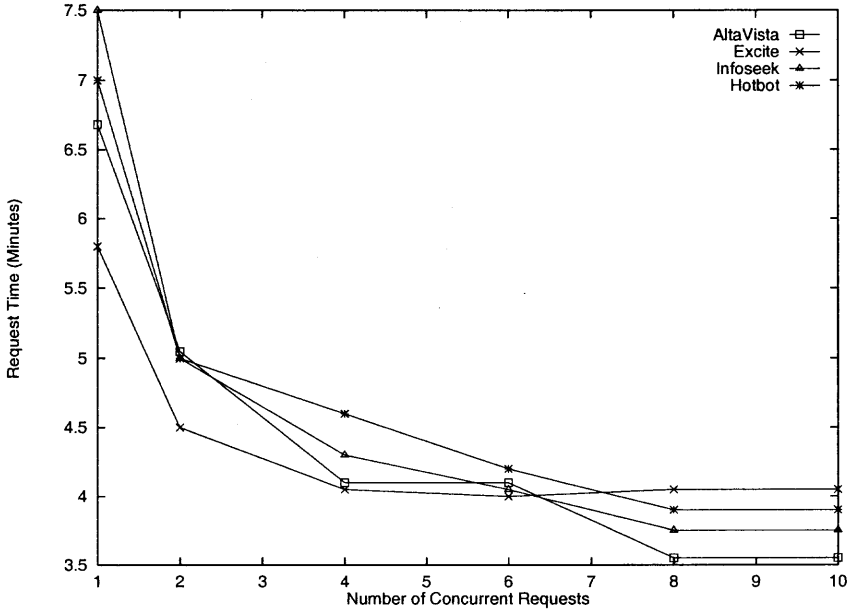


Figure 3.10 Number of Concurrent Requests vs. Request Time.

An experiment on the effectiveness of having concurrent HTTP requests versus a single HTTP request was conducted. The query pattern was the one in *Test 5*. The tests were conducted during normal hours (between 10AM-5PM EST) on four indices. In this experiment, only the first hundred URLs returned by each index server are processed. The experiment was performed using a Intel Pentium II-300MHz with 128MB RAM, running Linux, and connected to a local area network (100Mbp) that has a T1 connection to the Internet. The results are summarized in Figure 3.10.

From Figure 3.10, one can see a sharp drop in total request time from having just a single request to two concurrent requests. The performance gains were flat while system resources were wasted after having more than eight concurrent requests.

CHAPTER 4
MEDIATORS AND WRAPPERS

Search engines and directories, as described in Chapter 2, provide Internet users with rapid retrieval of information, but do not provide a database-like query language to retrieve information based, for example, on the underlying structure of the HTML documents. The lack of such query languages is due largely to the semistructured nature of Web data [2], which is unsuitable for retrieval and storage in a relational database form. Systems based on mediators and data warehouses have been introduced to overcome the inability to query Web data using a full-fledged database query language. In contrast to the approaches described in Chapter 3, which employ search engines as backends, mediators and data warehouses are based on a database management system (DBMS). Thus, the query languages in Chapter 3 would be implemented using search engines and directories as backends. Such a query system would generate, from the user's query, a query or a set of queries that can be executed on the search engines. The responses returned from the search engines would then be compiled and supplied to the user. In the mediator or data warehouse approach, on the other hand, the user interacts with the DBMS, which in turn interacts with the Web.

In the client/server DBMS model, a data server maintains the database. A client sends requests to the server and the server responds by returning a result. Figure 4.1 illustrates a client/server DBMS architecture. Architectures such as mediators and data warehouses have been introduced for a distributed environment where many servers are available. In the data warehouse architecture illustrated in Figure 4.2, data are collected from different servers, which may be scattered over a heterogenous wide area network, and integrated into a large data warehouse. The role of the data warehouse is to provide a centralized location to store data and process queries.

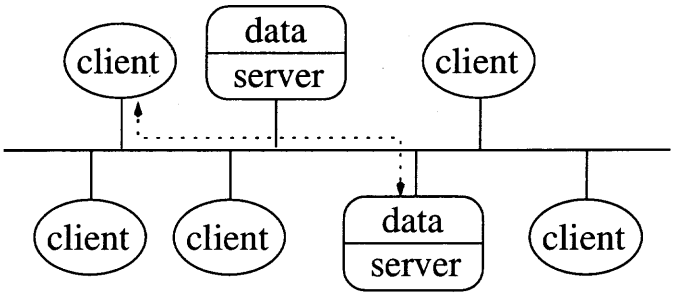


Figure 4.1 A client/server architecture.

Unlike data warehouses, mediators store only a minimal amount of data from various data sources. The goal of a mediator is to provide a centralized location for querying, as opposed to both centralized storage and querying as is done in the case of a data warehouse system.

Both data warehouse and mediator systems use software components, called *wrappers*, to extract data from the Web. The purpose of a wrapper is to filter and transform the Web data into suitable formats. A mediator architecture with wrappers is illustrated in Figure 4.3.

In addition, data warehouse and mediator systems use schemas to model structured data or object models to represent unstructured or semistructured data. The backend of each system is a DBMS which users can query instead of going directly to the Web. The advantage of using a DBMS instead of a Web search

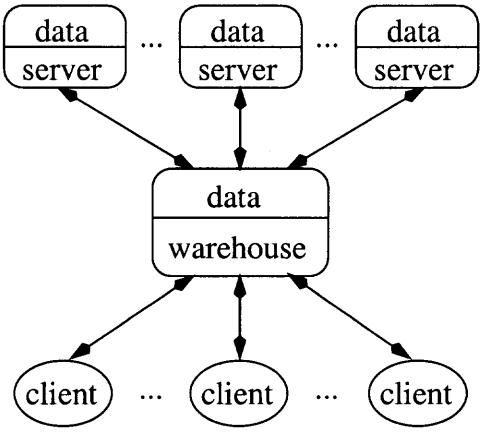


Figure 4.2 A data warehouse architecture.

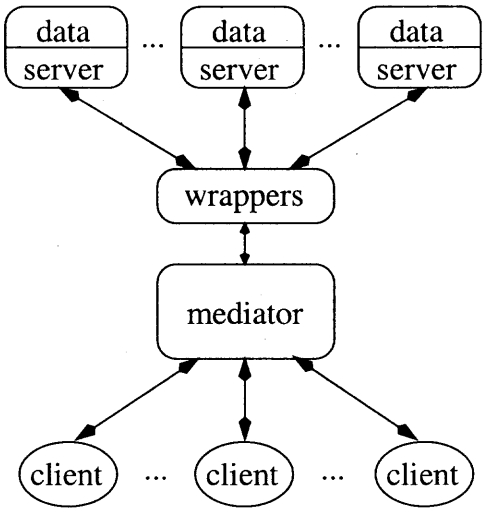


Figure 4.3 A mediator architecture.

engine is that the DBMS has benefits like security and querying facilities. The disadvantage of using a DBMS to store data from the Web is that the domain is limited, because specific programs (wrappers) must be implemented to extract data from different sources. Consequently, users see only that narrow view of the Web which the DBMS is intended to facilitate access to.

In this chapter, systems that use mediators and/or wrappers to integrate databases with the Web are discussed. Readers interested in XML should refer to [2, 64]. XML is a new language standard adopted by the World Wide Web Consortium (W3C) [149] that also can handle semistructured data, but unlike mediators and data warehouses, XML does not constitute a system. *WSQ/DSQ* [65] describe another approach that combines the query facilities of a traditional database system with existing search engines on the Web. WSQ, stands for *Web-Supported (Database) Queries*, uses search engine results to enhance SQL queries. DSQ, stands for *Database-Supported (Web) Queries*, uses a database to enhance Web queries. [65] focuses on WSQ.

4.1 LORE

Lore (Lightweight Object REpository) [102] is a DBMS designed specifically for managing semistructured information. Unlike traditional database systems that adhere to an explicitly specified schema, Lore can adapt to irregular data with a dynamic schema. Lore includes features such as dynamic structural summaries and seamless access to data from external sources. It uses a query language called Lorel derived by adapting the Object Query Language (OQL) [28] to permit querying semistructured data. Lore uses extensive type coercion and path expressions to query semistructured data effectively.

4.1.1 Object Exchange Model

The data model used by Lore is a self-describing, nested object model called Object Exchange Model (OEM) [3], introduced originally in the TSIMMIS [32] project, a system for integrating heterogeneous data sources. The notion of a fixed schema does not exist in OEM, designed for semistructured data. Data in this model is self-describing in nature and represented by a labeled directed data graph. Schematic information is dynamically embedded on labels assigned to the edges of the data graph.

Figure 4.4 illustrates a simple semistructured database based on the OEM model. The vertices in the graph are objects described by quadruples (`label`, `obj_id`, `type`, `value`) where `label` is a character string; `obj_id` is a unique object identifier; `type` is simple or complex. Simple objects, i.e., atomic objects that have no outgoing edges, contain a `value` from one of the basic atomic types such as `integer`, `float`, `string`, `jpg`, `audio`, etc. Complex objects have outgoing edges with `values` which are either a set or list of `obj_ids`. Special labels, called names, serve as aliases and entry points to the database. Objects not accessible by a path from some name are deleted.

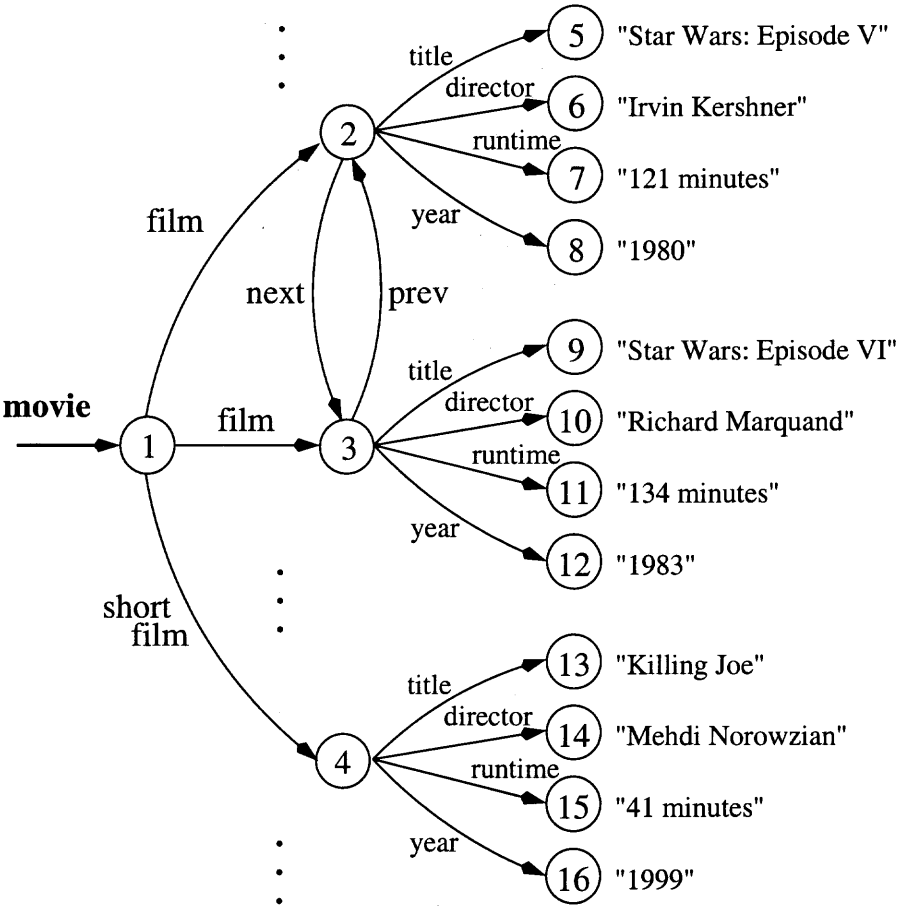


Figure 4.4 An OEM database.

4.1.2 Lorel

The Lorel query language is the core language of Lore. Lorel is similar to UnQL [24, 25] and uses pattern matching based on the syntax of the semistructured data format. The distinguishing feature of Lorel and UnQL is their ability to search using the data graph. Path expressions are used to specify the search criteria to an arbitrary depth in the data graph. Lorel uses path expressions to return a subset of nodes in the database. Lorel does not construct new nodes, which are the equivalent of a join operation in a relational database, nor test values in the database. The path expression is the basic building block used in finding specific patterns by Lorel.

A path expression is defined as a sequence of edge labels $l_1 . l_2 . \ldots . l_n$. Its result, as a simple query on a given data graph, is a set of nodes. For example, the result of the path expression "`movie.film`" in Figure 4.4 is the set of nodes $\{n_i, n_j\}$, where n_i is the `movie` node and n_j is the `film` node. The result of the path expression "`movie.film.title`" is the set of nodes with the associated set of strings like "{...,"Star Wars: Episode VI","Star Wars: Episode V",...}".

Rather than specify a path explicitly, we can specify it implicitly by imposing desired properties using regular expressions. A property is defined to be either a path property or an edge label. An example of a path/regular expression is "`movie.(film|short_film).title`", which can match either a `film` edge or a `short_film` edge. Let e be a regular expression. An example of a wild card for a regular expression on paths is as follows. Let _ (underscore) denote any edge label, let $e*$ denote a Kleene closure representing an arbitrary number of repetitions of e. Then, _* denotes an arbitrary sequence of edges. The expression `movie._*.director` finds a path that starts with a `movie` label, ends with a `director` label, and has any sequence of edges in between. Two more Lorel examples follow:

Example 1. This query searches for film titles with length greater than 2 hours.

```
select  title:  T
from    movie.film F, F.title T, F.runtime N
where   N > 2:00:00;
```

Example 2. This query searches for all information about films or short films directed by "Steven" after the year 1990.

```
select  row:  X
from    movie.(film|short_film) F
where   F.director = "Steven" and F.year > 1990;
```

4.1.3 Architecture

The basic architecture of the Lore system consists of three layers: an Application Program Interface (API) Layer, a Query Compilation Layer, and a Data Engine Layer as shown in Figure 4.5.

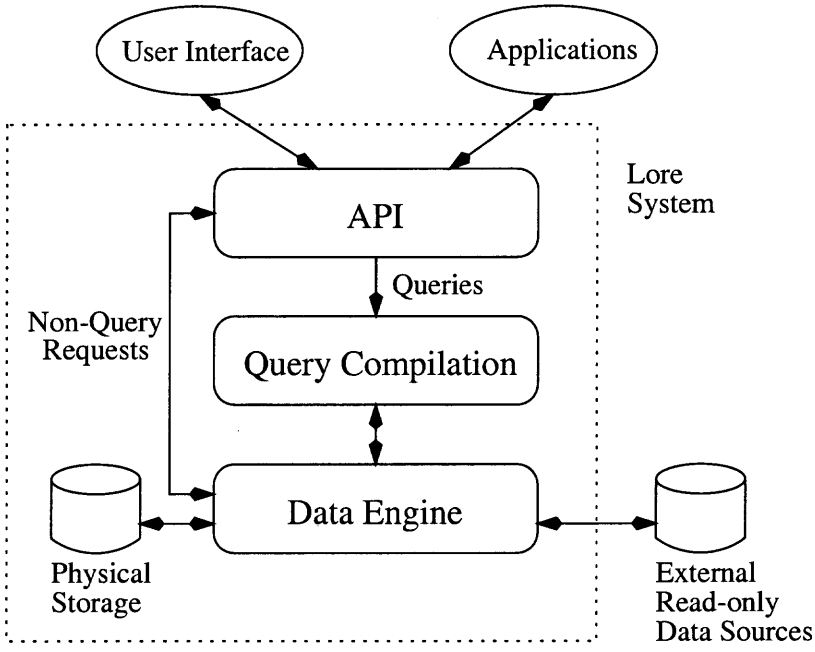


Figure 4.5 Lore architecture.

The API Layer provides access methods to the Lore system. There is a simple textual interface used primarily by system developers. A graphical interface for end users provides tools for browsing through query results, viewing the structure of data and formulating queries.

The Query Compilation Layer of Lore consists of a Parser, a Preprocessor, a Query Plan Generator, and a Query Optimizer. The Parser takes a query and checks whether it conforms with Lorel's grammar. The Preprocessor is responsible for transforming Lorel queries into OQL-like queries that are easier to process. A query plan is then generated from the transformed query by the Query Plan Generator. The query plan is optimized by the Query Optimizer that decides how to use indexes. The optimized

query plan is finally sent to the Data Engine Layer that performs the actual execution of the query.

The Data Engine Layer consists of an Object Manager, Query Operators, various utilities and an External Data Manager. The Object Manager is responsible for translation between the OEM data model and the low-level physical storage model. The Query Operators are in charge of executing the generated query plans. Utilities include a Data Guide Manager, a Loader and an Index Manager. The External Data Manager is responsible for interacting with external read-only data sources.

4.2 ARANEUS

The ARANEUS [11] project presents a set of languages for managing and restructuring data coming from the WWW. The main objective of ARANEUS is to provide a view to the Web framework. This framework has the following three view levels:

- *Structured view* – Data of interest are extracted from sites and given a database structure.
- *Database view* – Further database views can be generated based on traditional database techniques.
- *Derived hypertext view* – An alternative to the original site can be generated.

In this transformation process, Web data goes from a semistructured organization (Web pages) to a very structured organization (database), then back to a Web format (structured Web pages). Figure 4.6 illustrates the data flow of the transformation process.

In order to achieve this transformation, an interesting data model, called the ARANEUS Data Model (ADM), along with two languages ULIXES and PENELOPE are introduced. ADM is used to describe the scheme for a Web

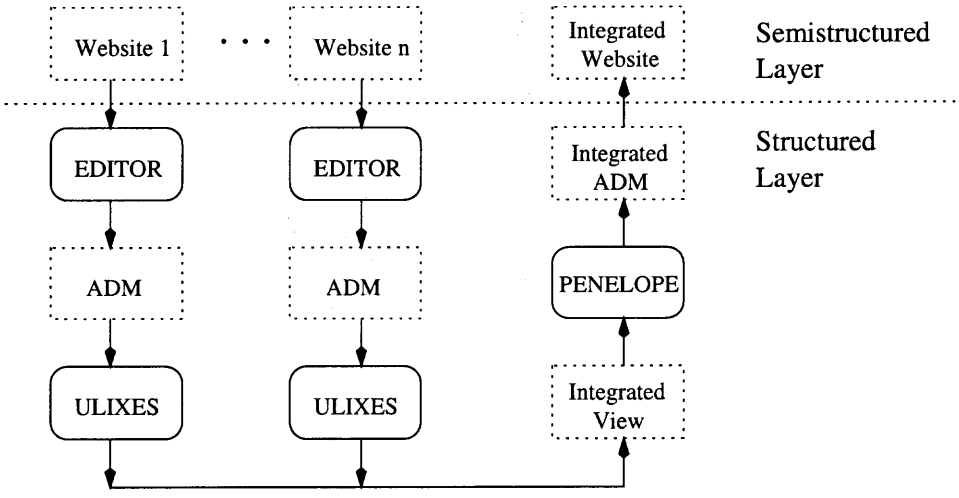


Figure 4.6 ARANEUS data transformation process.

hypertext, in the spirit of a database. The ULIXES language is used to define the database view over a site. The PENELOPE language is used to generate a hypertext derived from the database.

4.2.1 ARANEUS Data Model

ADM is a subset of ODMG [28] based on a page-oriented model which uses a page scheme to extract data from a Website. The notion of a page scheme can be assimilated to the notion of class in ODMG, though list is the only collection type in ADM. ADM also provides a *form* construct for the Web framework, and supports heterogenous union but not inheritance.

The page scheme is used to describe the underlying structure of a set of homogeneous pages in a site. It contains simple and complex attributes. Simple attributes include hyperlinks, images, text and binary data. Complex attributes include lists. The scheme of a Website can be viewed as a collection of connected page schemes. ADM extracts some properties of the pages using this structured Website scheme which provides a high-level description of a Website. The properties are then used as the basis for manipulation.

Text extraction procedures used to extract HTML page information as part of a page scheme, based on the EDITOR language [10], are employed. EDITOR, a language for searching and restructuring text, consists of a set of programs which act as wrappers for extracting and transforming HTML pages from a Website based on page schemes.

An ADM page scheme has the form $P(A_1 : T_1, A_2 : T_2, \ldots, A_n : T_n)$, where P is a page name, each $A_i : T_i$ is an attribute and ADM type pair. An attribute may be labeled *optional* and the page scheme may be labeled as *unique*. Figure 4.7 illustrates a page scheme example [11, 91], declared in the ARANEUS Data Definition Language.

```
PAGE SCHEME AuthorPage
  Name:     TEXT;
  WorkList: LIST OF
      (Authors:    TEXT;
       Title:      TEXT;
       Reference:  TEXT;
       Year:       TEXT;
       ToRefPage:  LINK TO ConferencePage
                      UNION JournalPage;
       AuthorList: LIST OF
             (Name:         TEXT;
              ToAuthorPage: LINK TO
                      AuthorPage OPTIONAL;););
END PAGE SCHEME
```

Figure 4.7 AuthorPage scheme for ARANEUS.

In this figure, `Name` is a uni-valued attribute, and `WordList` is a multi-valued attribute containing a set of nested tuples that describe the list of publications. For each publication, authors, title, reference, year, link to reference page (conference or journal), and an optional link to each corresponding author page are specified in the scheme. A `UNIQUE` keyword is assigned to page-schemes that have a single instance in the site as illustrated in Figure 4.8.

```
PAGE SCHEME PageName UNIQUE
  ...
END PAGE SCHEME
```

Figure 4.8 A unique page-scheme.

4.2.2 ULIXES

ULIXES is a language for defining a relational view over the Web. It is designed for extracting data from the Web based on an ADM scheme. The data extraction is based on navigational expressions, also known as path expressions. A `DEFINE TABLE` statement is used to define a relational view. It has the following form:

`DEFINE TABLE`	γ
`AS`	η
`IN`	ξ
`USING`	α
`WHERE`	ω

where γ specifies a relation $R(a_1, a_2, \ldots, a_n)$, η specifies navigation expressions over ξ; ξ specifies an ADM scheme; α specifies a set of attributes $(A_1, A_2, \ldots, A_n)$ used in ξ that correspond to relation $R(a_1, a_2, \ldots, a_n)$.

As an example, Figure 4.9 illustrates a relational view for VLDB papers used in [11] on the DBLP Bibliography server. In this example, the navigational expression requires that the `Submit` link returns pages according to scheme `AuthorPage`. The resulting table contains authors, titles, and references for all papers by Leonardo da Vinci in VLDB conferences.

4.2.3 PENELOPE

PENELOPE is a language for defining new page-schemes according to which data will be organized. It is used to transform relational views back to hypertexts that do not exist in the current site. The derived site can be specified using a `DEFINE PAGE` statement, which has the following form:

```
DEFINE PAGE  ρ
AS           ξ
FROM         γ
```

where ρ specifies a new page-scheme name and an optional `UNIQUE` keyword is used to indicate the page-scheme to be unique; ξ specifies the page structure; γ specifies a view that returns a relation $R(a_1, a_2, \ldots, a_n)$.

Taking the example from [11] again, suppose one wants to define HTML pages that structure Leonardo da Vinci's papers organized by year as illustrated in Figure 4.10. The structure of the pages can be defined using statements in Figure 4.11. These statements are then used to generate corresponding HTMLs for the new pages. Note that attributes from the source table `DaVinciPapers` are in <...>.

```
DEFINE TABLE  VLDBPapers (Authors, Title, Reference)
AS            AuthorSearchPage.NameForm.Submit →
                  AuthorPage.WorkList
IN            DBLPScheme
USING         AuthorPage.WorkList.Authors,
              AuthorPage.WorkList.Title
              AuthorPage.WorkList.Reference
WHERE         AuthorSearchPage.NameForm.Name =
                  'Leonardo da Vinci'
              AuthorPage.WorkList.Reference
                  LIKE '%VLDB%';
```

Figure 4.9 Relational view on VLDB papers.

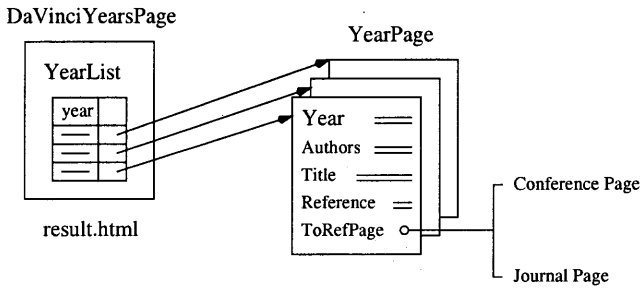


Figure 4.10 Page-schemes to organize papers by year.

```
DEFINE PAGE DaVinciYearPage UNIQUE
AS  URL       'result.html':
    YearList: LIST OF
        (Year:       TEXT <Year>;
         ToYearPage: LINK TO
                     YearPage (URL(<Year>)));
FROM DaVinciPapers;


DEFINE PAGE YearPage
AS  URL       URL(<Year>);
    Year:     TEXT <Year>;
    WorkList: LIST OF
        (Authors:    TEXT  <Authors>;
         Title:      TEXT  <Title>;
         Reference:  TEXT  <Reference>;
         ToRefPage:  LINK TO ConferencePage
                       UNION JournalPage;
                         <ToRefPage>);
FROM DaVinciPapers
```

Figure 4.11 HTML page generating schemes.

4.3 AKIRA

The Web is not a database, though retrieved Web pages are cached for faster retrieval. Such Web caching can be viewed as providing a primitive database that captures a view of the Web. A real DBMS can be used to provide database-supported Web caching. Database-supported caching requires Web pages to be stored as a database-supportable unit. Hence, a transformation process on Web pages is required to break Web pages into small pieces. An attempt using object-oriented databases to model HTML pages has been taken by AKIRA [80]. AKIRA introduces the notion of a *fragment*. HTML pages are stored and retrieved as fragments in a database. This database-supported caching can be viewed as a "smart cache" that populates the retrieved content with enriched information. Users can define their own content structure for delivery.

AKIRA integrates information retrieval, browsing, and database techniques into a flexible system for the user. It assumes zero-knowledge on the content source and a predefined schema is not required. The system also defines a query language, PIQL, which is a simple algebra extended with restructuring primitives in a unified framework.

4.3.1 Fragment Data Model

A fragment is also based on an object model. A fragment corresponds not to a source HTML page, but to a fragment of a page. Thus, an HTML page can be represented by one or more fragments. The structure of the `fragment` class can be represented as shown in Figure 4.12.

```
class Fragment {
  id                        :  Id_type;
  url, content              :  String;
  pred, next, href          :  Fragment;
  href_content, ref_name    :  String;
}
```

Figure 4.12 A `Fragment` class.

Using this definition of a fragment, an HTML page can be analyzed with finer granularity. In addition, by being able to analyze its contents and implicit structure, the semantics of the original page is preserved. Consider the HTML segment shown in Figure 4.13. It can be partitioned into fragments as listed in Table 4.1.

```
<TITLE>Data Mining</TITLE>
...
<A href="#Web mining">Related Sites</A>
Data Mining Related...
<A name="Web mining">Web Mining</A>
...
```

Figure 4.13 A segment of HTML.

ID	URL	CONTENT	PRED	NEXT	HREF	HREF_CONTENT	REF_NAME
1	...	"<TITLE>Data..."	null	2	null	""	""
...	...	...	...	...	...	...	...
5	...	"Related Sites"	4	6	7	"#Web mining"	""
6	...	"Data Mining..."	5	7	null	""	""
7	...	"Related Sites"	6	8	null	""	"Web mining"
...	...	...	...	...	...	...	...

Table 4.1 A list of HTML fragments generated from the HTML segment in Figure 4.13.

The notion of *concept classes* can be superimposed on that of a `Fragment` class as a component of the database. Concept classes can be used to store domain specific knowledge. A concept class is defined as an abstract class with an attribute REFERS_TO of type `Fragment` that refers to objects of class `Fragment`. Relational concepts can also be defined to express relationships between concepts. Two concept classes, `Person` and `MP3` based on the `Fragment` class, are illustrated in Figure 4.14. Concept classes are organized into a hierarchy. New data may be inserted into a new fragment, which is either based on preexisting fragments, or a new instance of

concept classes. Using a database to store concept classes provides a database query language with a rich environment for querying a view of the Web. AKIRA also can specify meta-concepts to express relationships between concepts. In Figure 4.14, concept class MP3 is associated with an instance of class Person.

```
class Person {
  name        : String;
  refers_to   : Fragment;
  ...
}

class MP3 {
  title       : String;
  artist      : Person;
  refers_to   : Fragment;
  ...
}
```

Figure 4.14 Concept classes for Person and MP3.

4.3.2 PIQL

PIQL (Path Identity Query Language) is a high-level, OQL-like query language for querying concept classes in AKIRA. It is reminiscent of OQL due to the adoption of an underlying object-oriented database approach. Similarly to Lorel [102], UnQL [24, 25] and POQL [35], it uses path expressions and supports fuzzy search constructs by allowing wild cards and a fuzzy function. The following are PIQL examples.

Example 1. This query is to find Web pages from World Wide Web Consortium that mention XML and return their URLs. The clause y.context = fuzzy("XML") can be used to search for Web pages similar to XML.

```
select  y.url
from    x in Fragment, y in Fragment
where   x.url = "http://www.w3.org/*"
        x.href = y
        y.content = fuzzy("XML");
```

Example 2. This query is to find the artist's name and the title of an MP3 song.

```
select  s.artist.name, s.title
from    s in MP3
        p in Person
where   s.url = p.url
        p in s.artist.refers_to;
```

4.3.3 Architecture

The AKIRA system consists of five components: Dispatcher, Database System, View Factory, Agent Pool, and Output Formatter. Figure 4.15 illustrates the architecture of AKIRA.

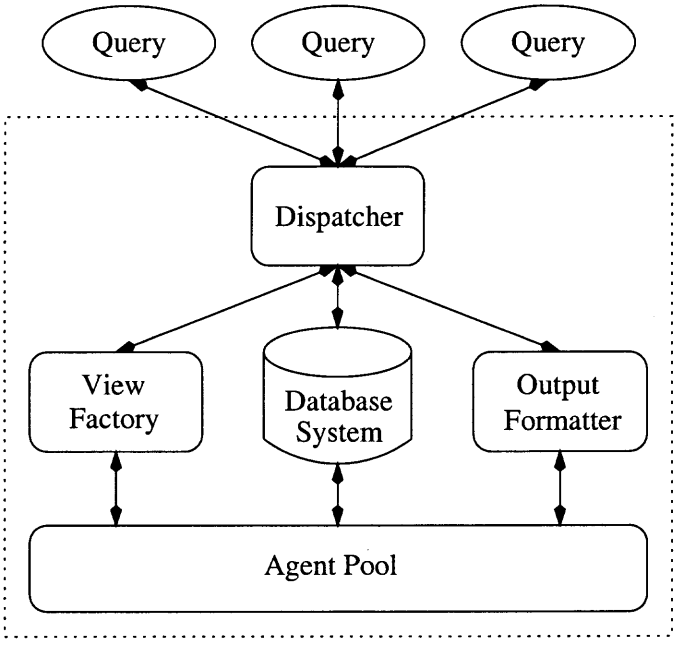


Figure 4.15 AKIRA system architecture.

The Dispatcher has a role similar to the query processor for a DBMS. It is the central component in charge of monitoring processes, taking a query and dispatching it to other components. It also performs query optimizations based on equivalence of algebraic expressions. Its key function in dispatching a query is to match a corresponding schema with elements of the system schema library. The schema is sent to

the Database System and computable queries to the View Factory, which maintains a list of views.

The Database System is used to allow efficient retrieval of objects. AKIRA uses an object-oriented database system that utilizes the PIQL language described in this chapter. Its main goal is to store a Web view using concept classes with a class `Fragment` as the base class.

The Agent Pool is a repository of services available to other components. AKIRA has an extensible agent architecture where new agents can be dynamically plugged into the system. Agents act as intelligent filters that parse the page content and identify fragments relevant to a query. Agents have the knowledge to identify specific types of information in order to apply the fragmentation process. The Output Formatter is the component that is responsible for displaying the result of the query in a user-friendly layout.

CHAPTER 5

MULTIMEDIA SEARCH ENGINES

Multimedia is integral to both human and modern computer communications. As digital sound and imagery proliferate, the need to search for audio and visual information has increased. However, most popular search engines are still textual as described in Chapter 2, even though the diversity of Web content has transformed the Web from a merely textual to a multimedia-based repository. Web information content comes in a variety of audio, video, image, and text formats, a list of the most commonly found media formats and types being given in Table 5.1. The multimedia information is highly distributed, minimally indexed, and lacks appropriate schemas. The critical question in multimedia search is how to design a scalable, visual information retrieval system? Such audio and visual information systems require large resources for transmission, storage and processing, factors which make indexing, retrieving, and managing visual information an immense challenge.

Progress has recently been made in developing and deploying efficient, effective, easy to use, mass-scale multimedia content search engines. Commercial search systems include AltaVista Photo Finder [5], Lycos Pictures and Sounds [94], scour.net [127], Yahoo! Image Surfer [152], Ditto [45], Stream Search [139], Midi Explorer [106], Lycos Fast MP3 Search [93], MP3 [107], and Sound Crawler [136]. Research prototypes for multimedia search engines include AMORE [7, 108, 109], WebSeek [31, 133, 146] and WebSeer [141, 147].

There are three categories of techniques for multimedia searching on the Web: text or keyword-based techniques, semantics or content-based techniques, or techniques based on a combination of both.

Media Format	*File Extension*	*Media Type*
Midi	`midi`	Audio
MP3	`mp3`	Audio
RealAudio	`ra,ram`	Audio
WAV Audio	`wav`	Audio
AVI	`avi`	Video
MPEG Video	`mpeg,mpg,mpe,mpv,mpegv`	Video
QuickTime	`qt,mov,moov`	Video
RealMedia	`ra,ram`	Video
MPEG Audio	`mp2,mpa,abs,mpega`	Video
PNG Image	`png`	Image
Windows Bitmap	`bmp`	Image
X Bitmap	`xbm`	Image
TIFF Image	`tiff,tif`	Image
JPEG Image	`jpeg,jpg,jpe`	Image
GIF Image	`gif`	Image
PDF	`pdf`	Document
TeX DVI Data	`dvi`	Document
Postscript	`ai,eps,ps`	Document

Table 5.1 Media types and file extensions.

- *Text or Keyword-based* – User can specify keywords, and multimedia relevant to the specified keywords can be retrieved. For example, find all images on `cars`.

- *Semantics or Content-based* – User can specify search criteria based on the semantic content of the multimedia object (image, audio, or video). For example, retrieve images visually similar to a given image. This is done using various image processing techniques.

- *Keyword and Content-based* – User can specify both keywords and content-based search criteria, combining the first two techniques.

This chapter surveys some of the most widely used techniques in multimedia searching on the Web. Though text is an inseparable part of a multimedia system, its retrieval and index methods were discussed in Chapter 2.

5.1 Text or Keyword-Based Search

Text or keyword-based multimedia search systems require an inverted file index to describe the multimedia content. The index is needed for fast query response, just as for keyword-based information search discussed in Chapter 2. Thus, building an index is at the heart of keyword-based multimedia searching.

Another important indexing technique used in multimedia search engines is partitioning multimedia content into categories, which the user can browse through for images of interest that match category keywords. Keywords can also be specified for finding similar images.

Identifying the text that describes content is essential to the cataloging and indexing process. The relationship between images and their textual captions in large photographic libraries like newspaper archives has been examined in [122, 137]. The lack of captions for multimedia content on the Web has led to efforts to develop indexes and searchable catalogs based on associated textual content such as the Web address and hyperlink reference text.

The use of the text embedded around multimedia content as a way of identifying its content is an approach born of necessity because automatically categorizing a media item based on its content is not an easy process. For example, in the case of an image, it requires image analysis using pattern recognition, feature extraction, and shape analysis algorithms, which are still wide open areas of research. Therefore, rather than applying complex visual analysis algorithms, models based on the text content and hyperlink structure surrounding the multimedia content have been proposed.

A variety of techniques have been developed for assigning keywords to multimedia content on the Web. For example, the URL and HTML tags associated with the images are considered in [31, 133]; text in the center tag within the same table cell is used in [141]; text after an image URL is used in [68]; and text "near" an image is used in [109, 121]. This section discusses all these techniques.

5.1.1 Keywords Assignment

Multimedia content, especially images, can be incorporated into HTML documents. The ability to link either text or images to another document or section of a document makes HTML very powerful. Very importantly, every image on the Web has a unique Web address (URL) and possibly some associated text that describes the image. The text close to an image therefore may be useful for characterizing and describing the content of the image. Images can thus be indexed or cataloged by using the following kinds of methods based on the surrounding text and hyperlink structure:

- Key Term Extraction.
- Directory Name Extraction.

There is a variety of information available on an HTML page to be used as a basis for assigning keywords to images. Key term extraction and directory name extraction can then be applied to categorize the image using text information in the tags and hypertext. The same techniques can be applied to audio and video files.

In key term extraction, terms are extracted from the hyperlink_text and the `ALT` tag field by chopping the text at non-alphabet characters. For example, consider the expression:

```
"vehicle/cars/nice_car.html" = "vehicle", "cars", and "nice car".
```

A search engine can index the terms extracted for the images using an inverted file as shown in Figure 5.1.

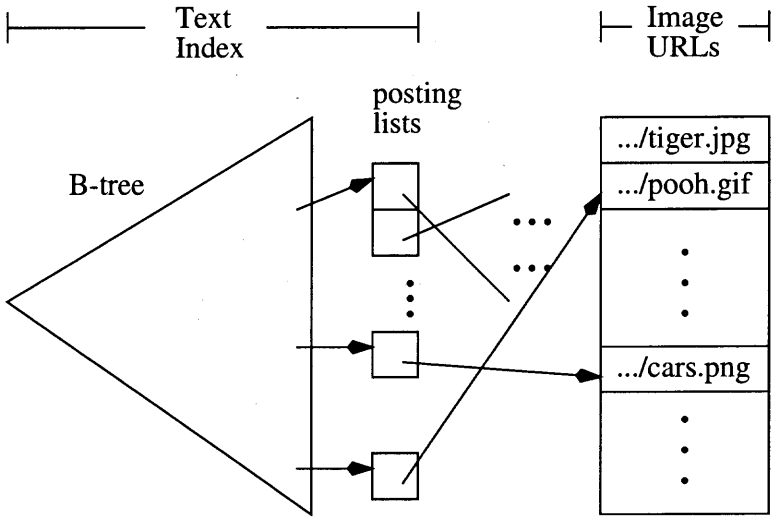


Figure 5.1 Keyword-image inversion index.

In directory name extraction, the name of a directory is extracted from the URL in tag fields such as `HREF` and `SRC`. The directory names are used to map images to subject classes based on a key-term dictionary.

A *key-term dictionary* is a set of key-terms and their corresponding mapping to subject classes. The key-terms are identified either manually or semi-manually by criteria such as frequency count. The mapping from key-terms to subjects also can be automatically or semi-automatically established.

Some common sources for key term and directory name extractions are as follows:

- Anchor tag – used to create links to other documents or images.
- Image tag – used to create inline images in a document.
- Heading tag – used to specify six levels of headers for a document.
- Title tag – used to specify a document title.
- Table tag – used to format a table.
- Document text – used to describe the content of images.

Anchor Tag

Images on the Web are published in two forms: referenced and inlined. A referenced image from a parent document uses the <A> tag, which stands for anchor. An anchor is an area in a document where a mouse click will link to the specified URL. The <A> tag has the following general format:

```
<A  HREF   = "URL"
    NAME   = "text"
    REL    = ["next"|"previous" ... ]
    REV    = [ "next"|"previous" ... ]
    TITLE  = "text" >    hyperlink_text   </A>
```

The HREF is used to specify the URL to be linked to when this hyperlink is clicked. This URL can be any legal URL, including images, sounds, etc. By specifying a NAME field, a specific location in the document will be accessible via another URL. The relationship between the URL specified in HREF and this document can be specified in the REL and REV attributes. REL defines the forward relationship between this document and the HREF URL; and REV defines the reverse relationship between the HREF URL and this document. The TITLE of the anchor can also be specified. The clickable area would be the hyperlink_text which describes the object pointed to by the hyperlink.

Image Tag

The ability to display inlined images is one of the most powerful features of HTML. Image types supported by Web browsers include BMP, GIF, JPEG and PNG. The HTML for including an image has the following general form:

```
<IMG  SRC     = "URL"
      ALT     = "text"
      WIDTH   = "pixels"
      HEIGHT  = "pixels"
      ALIGN   = ["top"|"middle" ... ]
      ISMAP   = "pixels"                >
```

The SRC specifies the URL, which is the location of the image. For text-only browsers such as Lynx, which cannot display graphic images, or for users who might have turned off image loading due to slow Internet connections, HTML provides a mechanism for displaying text in place of the missing images by using the ALT tag. Image size is specified in pixels using WIDTH and HEIGHT tags. Alignment can be specified in an ALIGN tag to align subsequent text. If ISMAP is specified, and an anchor surrounds the image, then the image is treated as a clickable map. That is, the coordinates of the mouse click will be processed by browser script or returned to the parsing script specified by the anchor's URL.

Inlined images can also be used as hyperlinks just like plain text. After an image becomes inlined it is clickable just like regular hypertext. The following HTML code illustrates how to make an image called bird.gif inlined:

```
<A HREF="URL"><IMG SRC="bird.gif" ALT="Bird Image"></A>
```

The URL field in both <A> and <IMG> tags has the following form, where [...] denotes an optional argument:

```
http://host.site.domain[:port]/[dirs/][file[.extension]]
```

Heading Tag

HTML headings in H{1-6} tags may also contain useful text information about images. Depending on the location of the images, some headings will tend to be less relevant to the image content. If an image is in heading Hi, the text for a previous heading Hj, where $j \leq i$, can be ignored [108]. For example, image "image.gif" in the <IMG> tag in the following HTML source is more relevant to heading <H3>Section 1.1.1</H3> than to the other headings.

```
<HTML>
...
<H1>Chapter 1</H1>
   <H2>Section 1.1</H2>
      ...
      <H3>Section 1.1.1</H3>
         ...
         <IMG SRC="image.gif">
      ...
   <H2>Section 1.2</H2>
      ...
</HTML>
```

Title Tag

HTML <TITLE> tags can also be used to help identify the semantics of images on a page [141], since the title tag is used to display a page title in the browser title bar. But, <TITLE> tags may not be useful when the images on a page have diverse content because they might not be directly related to the title.

Table Tag

Images can be formatted using the HTML <TABLE> tag. If the caption of an image is in the same cell as the image, then the caption can be used as an accurate description of the image [141]. But this method is likely to fail when the relevant text for the image is found in another cell. A more sophisticated approach can be applied to determine the corresponding image caption within the table cells. Captions within a table usually follow the same scheme. Caption alignment schemes within a table can then be determined by parsing the entire table and finding a regular pattern within it [108].

Document Text

Document text surrounding an image in a Web page may also be relevant to identifying image content. Determining which part of the text surrounding an image is relevant is a challenging problem because associated text can be aligned in many different ways. Image captions can occur before, after, or both before and after an image. A variety of approaches have been proposed for solving this image caption alignment problem.

For example, the text after an image URL until the end of a paragraph (<P> tag) or the text up to where there is a link to another image (<IMG> tag) can be taken as the caption of the image [68]. Line breaks (
 tag) have also been used [108] to define the visual closeness of a text to an image. Another approach is to use the text near an image, where *nearness* is defined as text lying within a fixed number of lines or words in an HTML document source file [121].

In situations where a text is simultaneously near two images, syntactical comparison can be applied [108]. The syntactical comparison can, for example, find the closeness between an image file name in the URL and the surrounding text. *Closeness*, defined as the syntactic distance d between the image file name f and the text t, is given by:

$$d(f,t) = \frac{c(f,t)}{|f|}$$

where $|f|$ is the number of characters in the string f, and $c(f,t)$ is the number of characters in f that also occur as a string in t in the same order.

5.1.2 Subject Taxonomy

The information extracted using the methods described in Section 5.1.1 in this chapter can be used to classify images into different subject classes. A subject class is an ontological concept that describes the semantic content of an image or video. Using an is-a hierarchy, the subject class can be arranged into a subject taxonomy. When a new and descriptive term is detected, it will be added to the taxonomy if it does not already exist. Table 5.2 shows top level subject classes (categories) available at Yahoo! Image Surfer [152], which is powered by Excalibur's Image Surfer [51]. For example, the Art category contains subcategories like architecture, ceramics, dance, painting, while the Science category contains subcategories like animals, space and astronomy.

• Arts	• Recreation
• Entertainment	• Science
• People	• Vehicles

Table 5.2 Categories of Yahoo! Image Surfer.

A catalog database can be built using subject classes and keywords with the following tables:

```
TYPE (ID, TYPE)
IMAGES (ID, URL, FORMAT, WIDTH, HEIGHT)
SUBJECT (ID, SUBJECT)
CONTENT (ID, TERM)
```

The following SQL query can be used to retrieve images of painting by Van Gogh.

```
select  ID
from    TYPE, SUBJECT, CONTENT
where   TYPE = "image" AND
        SUBJECT = "painting" AND
        TERM = "Van Gogh";
```

5.2 Content-Based Search

The text provided in the directory name and the hyperlink text described in the preceding section can produce an effective image search engine. However, by examining the image content, one can substantially refine the search results as well as provide more search features. Many systems use such techniques to overcome the shortcomings of keyword-based image search tools.

One of the first systems that allowed users to find images similar to a given image was the Query By Image Content (QBIC) [58]. It has attracted considerable attention because it allows finding images based on a given image's color and texture. The notion of image similarity was extended by Markus Stricker at ETH to include fuzzy color matching and fuzzy regions in an image. Image databases discussed in [116, 123] introduced image indexing techniques to find similar images. The Virage [12] system for image retrieval is based on visual features consisting of low level primitives, such as color, shape, texture and possibly domain specific features.

WebSeek [133] performs content-based search for images using color histograms generated from visual scenes. The color histograms describe the distribution of colors

in an image. Image similarity is determined based on the color histogram of each image. The weighted dissimilarity between histograms is calculated using dissimilarity functions. An image type assessment is automatically done using Fisher discriminant analysis to construct a series of uncorrelated linear weightings of the color histograms. This analysis provides maximum separation between training classes. Feedback from the user can also be used to reformulate a query to obtain better results, based on the principle that users are after all the best judges for determining the relevance of retrieved images.

A system called AMORE [7] uses a Content-Oriented Image Retrieval (COIR) library and allows a content-based query to specify images containing certain object shapes. AMORE has two subsystems: an indexing engine and a matching engine. The indexing engine is used to index identified shapes in images and the matching engine is used to search for images that match the shapes given in a query, as discussed in Section 5.2.1 in this chapter.

The indexing and retrieval complexity associated with audio and video content search has thus far impeded the large scale use of multimedia search engines on the Web. Therefore, most audio and video search engines are still text-based. In this section, three popular techniques, shape analysis, color analysis, and texture analysis, used in image similarity based search are discussed.

5.2.1 Shape Analysis

Shape analysis is the process of extracting objects from images. The process subdivides the original image into regions based on color, edge, position, and texture using image processing techniques like edge detection, color analysis and region division. Each region has a set of attributes such as color, shape, texture, size, object location and object composition. The extracted attribute values are stored

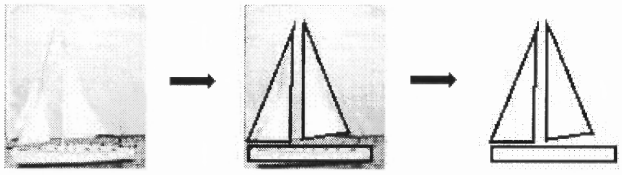


Figure 5.2 Object extraction from an image.

as metadata to be used during the matching process. Figure 5.2 shows the objects extracted from an image.

Objects extracted from images come in a wide variety of shapes and sizes, so it is a nontrivial issue as to how to cluster extracted objects. Predefined shapes such as circles, ellipses, triangles, rectangles, and squares of different sizes and aspect ratios may be used as templates for clustering seeds, as illustrated in Figure 5.3. An object only needs to be similar to a seed shape in order to be added to the corresponding cluster.

When a query image is specified for which one seeks to find similar images, the attribute values associated with the query image are extracted and stored as metadata. This metadata is then compared with image metadata in the database of indexed images and results are returned based on the matching scores of the comparison. Queries can also be formulated to find images containing certain shapes.

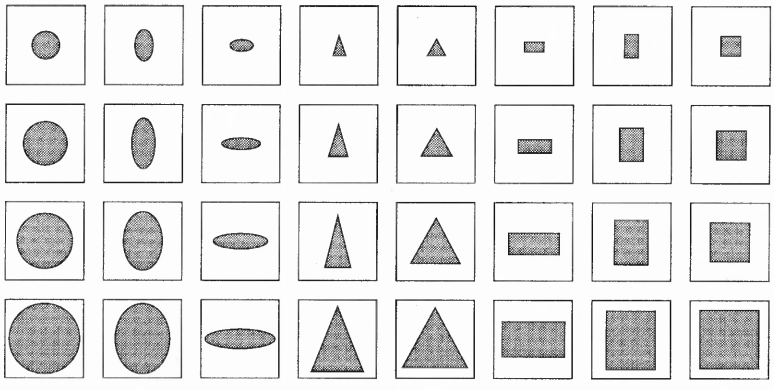


Figure 5.3 Shape templates for clustering seeds.

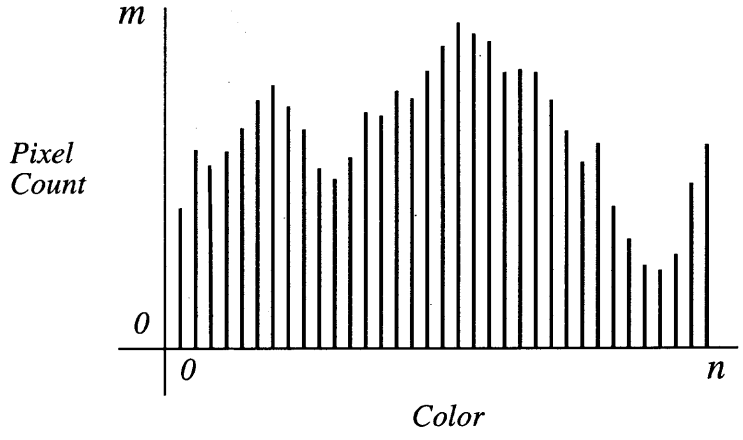


Figure 5.4 Color histogram.

5.2.2 Color Analysis

Another technique for representing the image content or characterizing an image is the *color histogram* [140]. A color histogram describes the color distribution in an image or video scene. A quantized HSV color space is typically used to represent the color. A color histogram $H(M)$ is denoted as a vector $(h_1, h_2, \ldots, h_n)$ in an n-dimensional vector space, where each h_c represents the number of pixels of color c in the image M. Figure 5.4 illustrates a color histogram. Given two color histograms, α and β, a function that measures the weighted dissimilarity between them can be defined as:

$$sim(\alpha, \beta) = 1 - \sum_{i=1}^{n} min(\alpha[i], \beta[i]),$$

where n is the number of bins used in the histogram, and $\alpha[i]$ and $\beta[i]$ are h_i in α and β vectors respectively. This method can be used to accurately and efficiently measure the (dis)similarity between two images based on color [134]. However, this method is not able to capture semantic information because two images can have similar color histograms but with different contents.

5.2.3 Texture Analysis

Texture analysis is another method of determining the similarity between images. The most commonly used features to represent texture are *coarseness, contrast, and directionality* (CCD) as developed in [55, 142].

- Coarseness – a measure of granularity of the texture.
- Contrast – a measure of the distribution of luminance.
- Directionality – a measure of the direction of an image.

An enhanced version of CCD using histogram-based features to reduce noise is described in [103]; and various content-based query models over image databases are discussed in [114].

CHAPTER 6

WEB CRAWLING AGENTS

An essential component of information mining and pattern discovery on the Web is the Web Crawling Agent (WCA). General-purpose Web Crawling Agents, which were briefly described in Chapter 2, are intended to be used for building generic portals. The diverse and voluminous nature of Web documents presents formidable challenges to the design of high performance WCAs. They require both powerful processors and a tremendous amount of storage, and yet even then can only cover restricted portions of the Web. Nonetheless, despite their fundamental importance in providing Web services, the design of WCAs is not well-documented in the literature. This chapter describes the conceptual design and implementation of Web crawling agents.

6.1 What is a Web Crawling Agent?

The term "Web crawling agent" combines the concepts of *agent* and *Web crawler.* Web crawling agents are responsible for intelligently gathering important data from the Web, data which can then be used in a pattern discovery process.

6.1.1 What is an agent?

The term "agent" is used in different ways in contemporary computing, the following three being the most common:

- Autonomous agents – refer to self-maintained programs that can move between hosts according to conditions in the environment.

- Intelligent agents – refer to programs that assist users in finding requested data items, filling forms, and using software.

- User-agents – refer to programs that execute requests for services through a network on behalf of a user. For example, Netscape Navigator is a Web user-agent and Eudora is an email user-agent.

Autonomous agents can travel only between special hosts for security reasons and are not widely used on the Internet. Intelligent agents are host-based software that have little to do with networking. User-agents need constant interaction and have limited intelligence. Web crawling agents exhibit the characteristics of both intelligent agents and user-agents, since they act intelligently and sometimes require user interaction. However, they are not autonomous agents since they do not travel between hosts.

6.1.2 What is a Web Crawler?

Web crawlers, which were briefly discussed in Chapter 2, are also known as crawlers, robots, spiders, walkers, wanderers, and worms. The Web crawlers are responsible for gathering resources from the Web, such as HTML documents, images, postscript files, text files and news postings. Due to the large volume of data on the Web, there is a need to automate the resource gathering process, which motivates the use of Web crawlers.

In general, crawlers are programs that automatically traverse the Web via hyperlinks embedded in hypertext, news group listings, directory structures or database schemas. In contrast, browsers are utilized by a user to interactively traverse portions of the Web by following hyperlinks explicitly selected by the user or by interfacing with search engines sites. Crawlers are commonly used by search engines to generate their indexes [117]. The traversal methods used by crawlers

include depth-first and breadth-first search combined with heuristics that determine the order in which to visit documents. An empirical study conducted in [86] has shown that roughly 190 million Web pages have been indexed by six major search engines. More recently, Google announced that its index encompasses over one billion (10^9) pages. All of these indexed pages were visited by crawlers deployed by the search engines.

Early papers on Web crawling include [48, 98, 117]; more recent studies are [29, 34, 71, 143]. According to the literature, crawlers are used to perform the following tasks:

- Indexing – build and maintain indexes for search engines.
- HTML validation – check whether a page conforms to HTML DTD.
- Link validation – check whether a link is still valid.
- Information monitoring – monitor changes to HTML pages.
- Information search – search for wanted documents.
- Mirroring – build mirror (duplicate) sites.

6.2 Web Crawling Architecture

Designing a scalable and extensible Web crawling agent presents formidable challenges due to the large volume of data on the Web. *scalable* means the crawling agent must be able to scale up to the entire Web. The design of scalable crawling agents presents both software and hardware challenges, because system hardware must be able to sustain constant heavy loads from the streams of data retrieved by the crawling agents. On the *extensible* side, the system must be designed in a

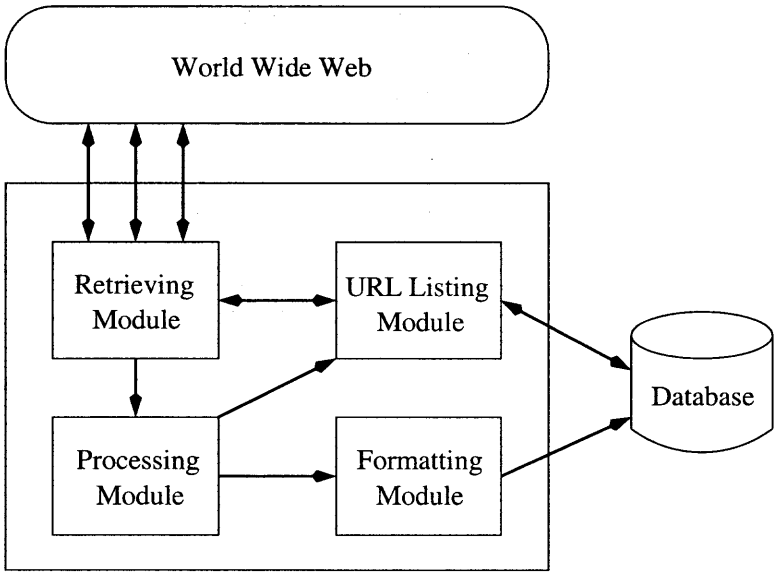


Figure 6.1 A Web crawling architecture.

modular way. The support for extensibility and customizability must be easily achieved by plugging in third party software or hardware modules.

The architecture of a Web crawling agent consists of the following subsystems: Retrieving Module Processing Module Formatting Module URL Listing Module.

Web crawling systems use multiple processors to share the load of information gathering and processing. A scalable distributed Web crawling system can effectively utilize these computing resources by using collaborative Web crawling [143]. Figure 6.1 illustrates the architecture of a distributed Web crawling system. The components used in this system are described below.

6.2.1 Retrieving Module

The Retrieving Module is responsible for retrieving all types of information in any format (text, image, graphics, audio, video) from the Web. The Retrieving Module fetches URLs from a pool of candidate URLs stored in the URL Listing Module. The Retrieving Module can be implemented using prebuilt components in various languages. Figure 6.2 illustrates a simple Retrieving Module written in Perl which

```
#!/usr/bin/perl
use URI::URL;
use HTTP::Request;
use HTTP::Response;
use LWP::UserAgent;

my $request_url = "http://www.kluwer.nl";
my $html_page = new URI::URL($request_url);
my $agent = new LWP::UserAgent;
my $request = new HTTP::Request('GET', $html_page);
my $response = $agent->request($request);

if ($response->is_success &&
   $response->header('Content-type') eq 'text/html') {
   print $response->content;
} else {
   print "ERROR: $request_url failed!\n";
}
```

Figure 6.2 A simple Retrieving Module.

retrieves an HTML page. It uses the Libwww-Perl library, a collection of Perl modules that provides a simple, consistent programming interface to the World Wide Web. For additional information on Perl, refer to Comprehensive Perl Archive Network (CPAN) at `http://www.perl.com/CPAN/`.

The program in Figure 6.2 uses three Perl modules: `URI`, `HTTP`, and `LWP`. First, a URL in the `URI` module is initialized with `http://www.kluwer.nl`, followed by instantiation of a `UserAgent` in the `LWP` module. The request is then prepared using the HTTP `GET` method and requested using the `UserAgent`'s `request` method. If the response is successful and the returned content type is HTML, then the page content is displayed; otherwise, an error message is displayed.

Due to the latency effects caused by network traffic and server load, the Retrieving Module should be designed to allow simultaneous requests to multiple

Web servers. Sites like Google and Internet Archive use multiple machines for crawling [22, 26]. Retrieved resources are then passed to the Processing Module.

6.2.2 Processing Module

The Processing Module is responsible for processing retrieved resources. Its functions include (i) determining the retrieved data type, (ii) summary extraction, (iii) hyperlink extraction, and (iv) interaction with knowledge bases or databases. The result is then passed to the Formatting Module. The Processing Module must also be capable of making decisions as to which URLs should be added to the pool of candidate URLs stored in the URL Listing Module.

6.2.3 Formatting Module

The Formatting Module is responsible for converting the diverse types of data sources retrieved and for transforming or summarizing them into a uniform metadata format. The metadata format generally reflects the format that will be used in the pattern discovery phase, and includes tuples that can be inserted into database tables and XML documents.

6.2.4 URL Listing Module

The URL Listing Module is responsible for feeding the Retrieving Module with candidate URLs to be retrieved. Usually a pool of candidates is stored in a queue with first-come-first-serve priority scheduling, unless specified otherwise. This module can also insert new candidate URLs into its list from the Processing Module.

6.3 Crawling Algorithms

A crawling agent uses an algorithm, often called a crawling algorithm, to retrieve Web pages and files. For example, in Harvest [19], the crawling algorithm starts

by retrieving an initial page, P_0, or a set of initial pages, $\{P_0 \ldots P_n\}$. The initial pages(s) are pre–selected by the user. URLs are extracted from the retrieved pages and added to a queue of URLs to be processed. The crawler then gets URLs from the queue and repeats the process.

One of the main objectives of a search engine is to capture the most recent view of the Web that can be achieved. However, most crawlers will not be able to visit every possible page for the following reasons:

- In addition to scanning the Web for new pages, a crawler must also periodically revisit pages already seen to update changes made since the last visit. These changes affect the index and crawling paths.

- Server resources are limited by their storage capacity. The Web has been estimated to contain 320 million indexable pages [86] and has already grown provably beyond one billion (10^9) pages [75].

- Network bandwidth is limited to the type and number of connections to the Internet. Only a limited number of simultaneous crawlers can be launched by a given Web crawling system.

Due to these limitations, smart crawling techniques are needed to capture the most current view of the Web in the most efficient manner. Two recent studies on smart crawling techniques propose crawling using techniques called URL ordering [34] and collaborative Web crawling [143]. These techniques significantly improve the efficiency and accuracy of crawlers.

6.3.1 URL ordering

Let $V = \{p_x, \ldots, p_y\}$ be a set of visited pages and $U = \{p_m, \ldots, p_n\}$ be a set of pages, called *front pages*, defined as follows:

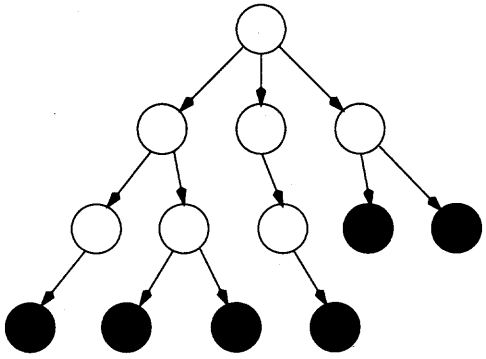


Figure 6.3 Illustration of front pages.

1. U consists of unvisited pages only, so U and V are disjoint.

2. For all $p_i \in U$, there exists a $p_j \in V$, such that p_i is 1-distance away from p_j (i.e., every page in U is at most one link away from a page in V). Since hyperlinks are directed paths, the converse may not hold.

Figure 6.3 illustrates the concept of front pages. The visited pages are the white nodes and front pages are the black nodes.

Front pages are stored in a queue in order of visitation, and await processing by the crawler. At any given instant in the crawling process, the decision according to which a crawler selects a page p_i from this pool of front pages is important. Selecting the most important pages to visit is significant because the crawler's ability to find the most important pages on the Web must be maximized. Several useful definitions of this importance measure can be used to prioritize the crawling order of front pages.

Given a Web page p_i, an importance measure of this page can be defined using one or a combination of the following methods:

- Document similarity measure
- hyperlink measure
- URL measure

Different implementation measures yield different URL orderings for crawling. The more important a page is, the earlier it is visited by the crawler. Each importance measure is described in turn below.

6.3.1.1 Document Similarity Measure The similarity between a pair of pages is one indication of their mutual relevance. For example, pages containing many of the same keywords would presumably be relevant to each other. Relevance is in turn an indicator of importance.

In a document similarity measure method, $sim(P,Q)$ is defined as the textual similarity between P and Q, where P is a Web page and Q is the query used as the search criterion that drives the crawler. For example, the criterion can be "Find all documents related to data mining". To compute similarities, a vector model [124] developed for information retrieval can be used. Basically, P and Q are represented as t-dimensional vectors, where t is the total number of index terms in the system. The degree of similarity of P and Q is the correlation between the vectors $\vec{P}$ and $\vec{Q}$. This correlation can be quantified by using the cosine of the angle between these two vectors as follows:

$$sim(P,Q) = \frac{\vec{P} \bullet \vec{Q}}{|\vec{P}||\vec{Q}|}$$

Thus, if the vectors are identical, the similarity is one. On the other hand, if the vectors have no terms in common (i.e. they are orthogonal), the similarity is zero.

Dissimilarity can be measured by the inverse document frequency (IDF) of a term w_i in the page collection. IDF is used to distinguish a relevant document from a non–relevant one by the number of times the term w_i occurs in a page. Observe that frequently occurring terms are not helpful in this decision process. Furthermore, because the crawler has not visited all pages, the IDF can only be calculated from the visited pages. Other document similarity models such as the Boolean, probabilistic, and fuzzy set models can also be used.

6.3.1.2 Hyperlink Measure Given a Web page, P, there are two types of links with respect to P:

- Back-links or inlinks – which are hyperlinks that are linked to P.
- Forward-links or outlinks – which are hyperlinks that are linked from P.

The *back-link count* is defined as the number of back-links to P; the *forward-link count* is defined as the number of forward-links from P.

The assumption underlying the use of this measure is that a page is important if there are many back-links to it. The corresponding importance measure $\zeta(P)$ is defined as the number of links to P over the entire Web. A crawler may estimate $\zeta(P)$ based on the number of back-links to P from pages it has visited or using search engines to get pre-calculated back-link counts.

The forward-link count for a page P is not an indicator of whether P is an important page. Furthermore, links to a page should be weighted differently if they are from an important site such as the homepage of a major search engine. This idea leads to the development of a page rank metric [115] which calculates $\zeta(P)$ using the weighted value for each page: a page with many back-links from important pages should be considered important.

6.3.1.3 URL Measure In this method, $\zeta(P)$ is a function of its hyperlink location (URL), rather than its text contents. For example, URLs ending with ".com" would have different weights than those with other endings. A URL starting with "www" and "home" may be more interesting because it represents the homepage of a Website.

6.3.2 Web–Graph Partitioning

The Web is often viewed as a directed graph $G = (V, E)$, where each $v \in V$ corresponds to a Web page or URL and the hyperlinks between pages define the set E of directed edges. From a graph point of view, building a scalable, distributed collaborative Web crawling system (CWCS) [143] corresponds to developing a Web-graph partitioning algorithm for automatic load balancing among crawling processors.

How are the processors affected by graph partitioning? Ideally one wants each processor to process a disjoint set of URLs from the other processors. However, since pages are interlinked in an unpredictable way, it is not obvious how this can be done. Additionally, the processors must communicate with one another to ensure they are not redundantly processing URLs. An effective Web graph partitioning algorithm will attempt to partition the Web-graph to minimize redundancy.

One of the main difficulties in the crawling process derives from the fact that G is unknown prior to the crawling process occurring. Furthermore, even after the crawling process is performed, only the subsets of G are known. This is because the Web is undergoing dynamic changes—information is being added and deleted, even as crawling is taking place. Moreover, not all reachable sites are visited. This complicates the application of graph partitioning methods such as those designed for static graphs in VLSI circuit design. Dynamic re-partitioning and load re-balancing are required for the Web-graph.

A *k-way* partition of the Web-graph $G = (V, E)$ is a division of G into k subgraphs $U = (U_1, U_2, \ldots, U_k)$, where k is the number of processors in the system. Each U_i is mapped to processor i and $L_{i,j}$ denotes the set of cross partition links from U_i to U_j. The communication between processors i and j, reflected in the links in $L_{i,j}$, represents overhead in a collaborative Web crawling system.

The partitioning scheme has three basic steps:

1. *Coarsening.* Apply node and edge contraction to the Web-graph in order to obtain a manageable structure to which to apply the partitioning algorithm.

2. *Partitioning.* Apply a static graph partitioning algorithm to divide the coarsened Web-graph, and then project this partition back to the original Web-graph.

3. *Dynamic re-coarsening and repartitioning.* When the load becomes unbalanced, re-coarsen the Web-graph and repartition the coarsened Web-graph based on the current Web-graph information generated by the crawler after the last partitioning.

6.4 Topic-Oriented Web Crawling

The first generation of Web search engines assumed queries could be posed about any topic. This single database model of information retrieval was successful because the Web was limited in size and in the complexity of its contents [27]. Indeed, while it is true that when a query is made to a search engine, the query is most likely related to a limited set of topics [36]; nonetheless, such a "one size fits all" model of search tends to limit diversity, competition, and functionality [84]. Consequently, Web search has evolved beyond the single database model to the multi-database, topic-oriented model, in an effort to reflect the diversity of topics on the Web, as well as to achieve higher precision search results.

Topic-oriented Web crawling (TOWC) [29, 43] was designed to facilitate building topic-oriented search engines. TOWC is intended to search the portion of the Web most closely related to a specific topic. In the TOWC approach, the links in the crawling queue are reordered, as described in Section 3.1 of this chapter, to increase the likelihood that relevant pages are discovered quickly.

To achieve this, TOWC algorithms start with a seed-set (or root-set) of relevant pages, and then attempt to efficiently seek out relevant documents based on a combination of link structure and page content analysis, using criteria such as: in-degree link count, out-degree link count, authority score, hub score, and topic-relevant score.

Topic-relevant scoring functions use techniques such as: *text classification* to classify and rank the relevance of a Web page to a given topic and build a topic hierarchy [47, 99]; *machine learning* to classify vectors in multi-dimensional space [120]; and *PageRank measures* to compute a weighted hyperlink measure, which is intended to be proportional to the quality of the page containing the hyperlink [22]. TOWC has been successfully used to efficiently generate indexes for search portals and user groups [100].

CHAPTER 7

ENVIRODAEMON

7.1 Background

Engineers and scientists have longed for instantaneous, distributed network access to the entire science and technology literature. These longings are well on their way to being realized as a result of the improvement and convergence of the computing and communications infrastructure and the indispensability of the Internet for scientific research. The size of the organization able to perform a search has decreased as groups of lay people and scientists can now search "digital libraries" without the aid of trained reference librarians.

Similarly, as discussed in the previous chapters, the document being sought has changed: from a citation with descriptive headers, to an abstract, to complete multimedia contents including text, audio, video, and animation. There are many reasons for the rise in digital information. The preservation of the contents of physical, paper-based texts, the convenience associated with maintaining, searching and retrieving electronic text, and the lowered cost of acquiring and maintaining electronic information as opposed to books and journals are often cited reasons [89].

This chapter reports a system for Internet search in a category-specific area: the environment. In the environmental domain, sustainable development is influenced by environmental decisions throughout the entire life-cycle of products (in a wide sense, including non-tangible products and services). In the early stages of the life-cycle, for example, in requirements gathering and design, decisions may be made with far-reaching consequences for the rest of the products' life-cycle. At this point in time, it is clearly impractical to teach all the people involved in such decisions all relevant environmental considerations. A more practical approach is to make tools available that permit those people most directly involved in the decision making to evaluate

various alternatives of the products, and to assess the environmental impact of their decisions.

Such tools will rely heavily on locating relevant information, and making it available in a suitable form when it is needed. Current tools for locating information, such as the Web search engines discussed in Chapter 2, are mostly keyword based, and are not adequate for identifying relevant items and filtering out irrelevant ones. As a consequence, needed information is not easily available, and the quality of the work performed may suffer.

Searching involves queries that can involve structure (i.e., words or concepts). There are two basic types of hypertext queries: content queries, which are based on the content of a single node of the hypertext; and structure-specifying queries, which take advantage of the information, conveyed in the hypertext organization itself. Retrieval involves end user displays such as graphical user interfaces commonly, and can involve categorization, filtering and routing.

At present, access to the Web is based on navigationally oriented browsers. The end result is often a "lost-in-cyberspace" phenomenon because a) there is no reliable road map for the Web; b) obtained information is heterogeneous and difficult to analyze; and c) organization of documents conveys information which is often not exploited.

Most of the currently available Web search tools as described in previous chapters suffer from some of the following drawbacks:

- User partial knowledge is not fully exploited. For example, the user may know that a particular keyword is in the header or in the body of a document which would aid in its location and retrieval.

- The restructuring ability of current tools is limited or nonexistent. The querying tool should permit ad hoc specification of the format in which the answer should be presented. One should be able to search for two chapters that have references to the same article or to view two references side by side when output is given.

- The dynamic nature of Web documents are unaccounted for, which results in poor query result quality.

The primary need of end users is to quickly and easily help them find the exact piece of information they need (even though they might not be able to exactly describe their needs) without letting them drown in an information sea. When faced with the task of searching for something, one can ask for recommendations for Websites from others or use Web indexes which are manually constructed and organized by category (e.g., the search engine Yahoo! is a good example). Using this latter scheme, sites appear more quickly than can be indexed by hand and a search engine can rapidly scan an index of Web pages for certain key words. A better solution involves the use of visualization methods.

7.2 EnviroDaemon (ED)

As mentioned before, information retrieval has evolved from searches of references, to abstracts, to documents. Search on the Web involves search engines that promise to parse full-text and other files: audio, video, and multimedia. With the indexable Web approaching one billion pages and growing, difficulties with locating relevant information have become apparent. The most prevalent means for information retrieval relies on syntax-based methods: keywords or strings of characters are presented to a search engine, and it returns all the matches in the available documents. This method is satisfactory and easy to implement, but it has some inherent limitations

that make it unsuitable for many tasks. Instead of looking for syntactical patterns, the user is often interested in keyword meaning or the location of a particular word in a title or header.

In this section, a search engine developed in NJIT DB Lab, called Enviro-Daemon, is discussed. This system was built to ease the task of finding information about pollution prevention (P2), an emerging environmental field in which a premium is placed on preventing pollution in the first place instead of accepting waste as a given and treatment and disposal as necessary evils.

EnviroDaemon [69] automatically builds and updates a catalog of objects at pre-selected Internet sites that are related to P2. Users search for information by submitting keywords. Searches can be restricted to small subsets of the indexed sites by choosing one of five search criteria (Pollution Prevention and Pollution Control, Regulatory, International, ISO 14000, and Recycling and Materials Exchange), or the entire catalog can be searched. The results are returned rapidly and are embedded in several lines of text to provide context. If the text looks promising, the user can click on a hyperlink to access the full article. Figure 7.1 shows the front-end interface for EnviroDaemon. In contrast to generic search engines such as Yahoo! and Lycos, EnviroDaemon is highly specific, focusing solely on P2 information.

EnviroDaemon employs software constructs termed Gatherer and Broker of the Harvest application system to gather, extract, organize, retrieve and filter information. Atop Harvest is the Glimpse tool, which functions as an indexer of the information and provides a query system for searching through all of the files. Enviro-Daemon updates its catalog of more than 35,000 pollution prevention-related objects from about 160 Internet servers regularly.

The Internet holds vast stores of information pertaining to pollution prevention and environmental issues, and with each day more is added. Yet, as this storehouse grows, the difficulty of finding specific information increases. This section details how

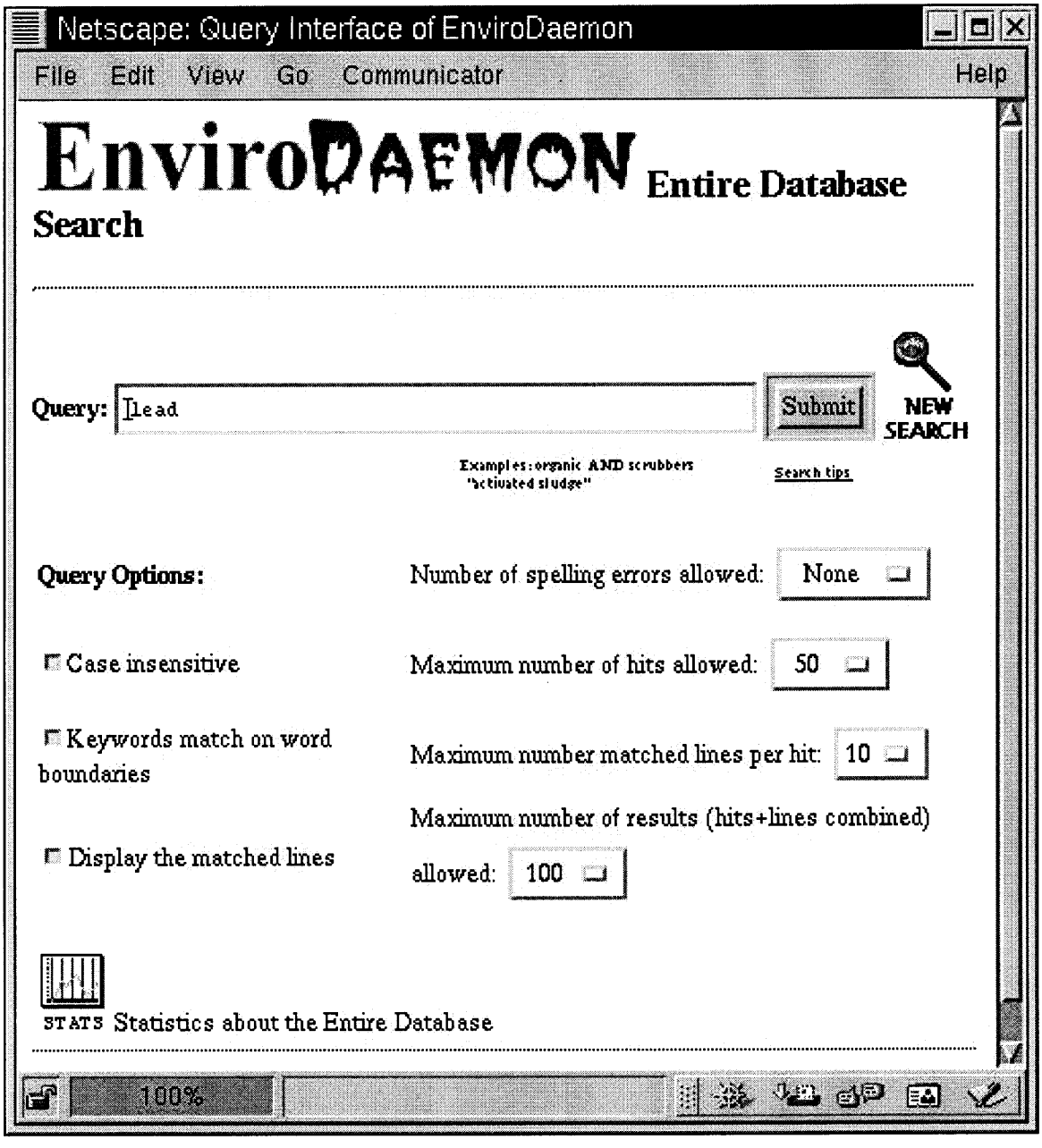


Figure 7.1 Front-end interface for EnviroDaemon.

EnviroDaemon was built to ease the task of finding this information and building a knowledge base.

More and more of the information that environmental managers need is becoming available on the Internet. Each day, an increasing amount of valuable, substantive information is made available via sources like Websites, email list servers, and bulletin board systems. While many of these computerized information sources are potentially helpful, no organized structure exists to catalog all the data to enable someone to find specific information quickly. This is true of pollution prevention (P2), a rapidly developing field of environmental science and engineering that focuses on elimination of sources of hazardous substances, rather than on treatment or disposal.

To make computerized pollution prevention information more easily accessible, we developed EnviroDaemon, a P2-specific search engine. EnviroDaemon is designed to be useful for anyone who is searching the Internet for environmental information, whether that person is a government policymaker, consultant, researcher, student, or environmental advocate.

One can query certain specialized environmental databases (for instance, the Solvent Alternatives Guide-SAGE [135]), but no individual Website is so comprehensive that the P2 information-seeker only needs to look there. To do a thorough job, the P2 information-seeker needs to search many P2 sites, or risks missing valuable information. Although some of these sites are linked together, most are not. They need to be searched individually. Thoroughly searching ten sites is at least an afternoon's work. Hence, searching the "information superhighways" for particular information can be quite time-consuming. Even expert users who are familiar with environmental sites need to spend considerable time in searching, because new sources of data are appearing continually.

Generic search engines were designed to mine the Web. Given a search term, these engines return lists of Websites and specific references, often copiously. One could use one of the generic search engines to look up terms such as "pollution prevention" or a specific solvent and then wade through the resulting Websites, hoping to chance upon helpful references. However, these search engines return so many "hits," most of which are unrelated to P2, that going through them all becomes burdensome.

7.2.1 An Intelligent Librarian

EnviroDaemon operates like the generic search engines, but its scope is focused on pollution prevention information. EnviroDaemon has expedited the gathering of P2 information by performing highly efficient, specific searches.

EnviroDaemon acts like an intelligent librarian, who in the vast library of the Internet, searches only specific collections devoted to pollution prevention. It keeps a catalog of items on more than 160 preselected P2 sites on the Internet. In response to a query, EnviroDaemon matches the search terms against its catalog. The user may further narrow the search by choosing one of six specialized search criteria:

- Pollution prevention and pollution control
- Regulatory
- International
- ISO 14000
- Recycling and materials exchange
- The entire P2 database

This service saves the information-seeker the time and trouble of searching dozens of Websites and, because the "hits" come only from P2 sites, it returns only

the information that is most likely to be germane. To allow the user to select the most pertinent citations, EnviroDaemon returns the search item within the context of its surrounding text. Up to ten lines, as determined by the user, may be displayed.

The EnviroDaemon project followed the following steps, the first of which was to identify extant search engines, and to decide whether to tailor one or to construct EnviroDaemon *de novo*. The decision was made to build a search engine de novo. The third step was to determine the appropriate Websites to mine for P2-related information. Next, EnviroDaemon was constructed and tested on the Web.

A look at the process by which EnviroDaemon finds, catalogs, and reports P2 information will give the reader an idea of its power and utility. It may also show the way to others who might wish to create similar custom search engines.

7.2.2 Selecting Tools

Once the decision was made to build a search engine, a careful, detailed analysis was made to select the appropriate tools. Harvest software tools [19], the Glimpse index/search mechanism [97] and the associated software that is required by Harvest and Glimpse (Glimpse stands for Global IMPlicit Search) was chosen. Harvest employs a system of Gatherers and Brokers to retrieve and filter information. Glimpse indexes the information and provides a query system for rapidly searching through all of the gathered files.

Harvest software provides an integrated set of tools to gather, extract, organize, search, and replicate relevant information across the Internet. It possesses a flexible data-gathering architecture, which can be configured in various ways to create many types of indexes, making efficient use of Internet servers, network links, and index space on disk. Harvest permits users to build indexes using manually constructed templates (for maximum control over index content), automatically constructed templates (for easy coverage of large collections), or a combination of the two

methods. Users also may extract structured (attribute-value pair) information from many different information formats and build indexes that permit these attributes to be referenced during queries (e.g., searching for all documents with a certain regular expression in the title field).

Measurements indicate that Harvest can reduce server load by a factor of over 6,000, network traffic by a factor of 60, and index space requirements by a factor of over 40 when building indexes compared with other Internet search engines, such as Archie, Wide Area Information Services (WAIS), and the World Wide Web Worm.

7.2.3 Building EnviroDaemon

EnviroDaemon was originally designed and run on a Sun Sparc 5 computer with Solaris 2.3, with a 4GB SCSI drive. EnviroDaemon does not require a fast processor, but does require more memory. The critical factor affecting RAM usage is how much data EnviroDaemon tries to index. The more data, the more disk input-output is performed at query time, and the more RAM it takes to provide a reasonable disk buffer pool.

The amount of disk space required was decided after evaluating the size of the data to be indexed. Approximately 10 percents as much disk space as the total size of the data to be indexed is needed to hold the databases of Gatherers and Brokers. The actual space needs depend on the type of data indexed.

For example, PostScript achieves a much higher indexing space reduction than HTML because so much of the PostScript data (such as page positioning information) is discarded when building the index. An additional 50 MB of free disk space is required to run the Harvest Object Cache.

7.2.4 Harvest Subsystem Overview

Harvest consists of a number of different subsystems. The Gatherer subsystem collects indexing information (such as keywords, author names, and titles) from the resources

available at provider sites (such as FTP and Web servers). The Broker subsystem retrieves indexing information from one or more Gatherers, suppresses duplicate information, incrementally indexes the collected information, and provides a Web query interface to it. The Replicator subsystem efficiently replicates Brokers around the Internet. Information can be retrieved through the Cache subsystem. The Harvest Server Registry (HSR) is a distinguished Broker that holds information about each Harvest Gatherer, Broker, Cache, and Replicator in the Internet.

7.2.5 Installing the Harvest Software

Harvest Gatherers and Brokers can be configured in various ways. Six different Brokers and Gatherers were running on the server. Each of the six search criteria has one dedicated Broker and Gatherer running. Since the Gatherers and Brokers are running locally, the search process is very efficient and quite fast.

In addition to the above platform requirements, the following software packages were installed:

- All Harvest servers require Perl v4.0 or higher (v5.0 is preferred).
- The Harvest Broker and Gatherer require GNU gzip v1.2.4 or higher.
- The Harvest Broker requires a Web server.
- Compiling Harvest requires GNU gcc v2.5.8 or higher.
- Compiling the Harvest Broker requires Flex v2.4.7 and Bison v1.22.

7.2.6 The Gatherer

The Gatherer retrieves information resources using a variety of standard access methods (e.g., FTP, Gopher, HTTP, NNTP, and local files) and then summarizes

those resources in various type-specific ways to generate structured indexing information. For example, a Gatherer can retrieve a technical report from an FTP archive, and then extract the author, title, and abstract from the paper to summarize the report. Brokers can then retrieve the indexing information from the Gatherer.

The Gatherer consists of a number of separate components. The Gatherer program reads a Gatherer configuration file and controls the overall process of enumerating and summarizing data objects. The structured indexing information that the Gatherer collects is represented as a list of attribute-value pairs using the Summary Object Interchange Format (SOIF). The gathered daemon serves the Gatherer database to Brokers. It remains in the background after a gathering session is complete. A stand-alone gathering program is a client for the gathered server. It can be used from the command line for testing and is used by a Broker. The Gatherer uses a local disk cache to store objects that it has retrieved.

Even though the gathered daemon remains in the background, the Gatherer does not automatically update or refresh its summary objects. Each object in the Gatherer has a time-to-live value: Objects remain in the database until they expire. The Gatherer needs a list of the Uniform Resource Locators (URLs) from which it will gather indexing information from the list of 160 pollution prevention servers. This list is specified in the Gatherer configuration file.

To prevent unwanted objects from being retrieved across the network, the Gatherer employs a series of filters. This dramatically reduces gathering time and network traffic. Since these filters are invoked at different times, they have different effects. The URL-filter, for example, allows only those Internet sites that have been pre-selected. After the Gatherer retrieves a document, it passes the document through a subsystem called Essence, which screens out certain documents from being indexed. Essence provides a powerful means of rejecting indexing that is based not only on file-naming conventions, but also on file contents (e.g., looking at strings at the

beginning of a file or at UNIX "magic" numbers) and also on more sophisticated file-grouping schemes. Independent of these customizations, the Gatherer attempts to avoid retrieving objects when possible, by using a local disk cache of objects, and by using the HTTP "If-Modified-Since" request header.

Essence extracts indexing information from those documents that are not filtered out. Essence allows the Gatherer to collect indexing information from a wide variety of sources, using different techniques depending on the type of data. The Essence subsystem can determine the type of data (e.g., PostScript vs. HTML), "unravel" presentation nesting formats (such as compressed "tar" files), select which types of data to index (e.g., don't index Audio files), and then apply a type-specific extraction algorithm (called a summarizer) to the data to generate a content summary. Harvest is distributed with a stock set of type recognizers, presentation unnesters, candidate selectors, and summarizers that work well for many applications. Summarizers were customized to change how they operate, and added summarizers for new types of data.

To reduce network traffic when restarting aborted gathering attempts, the Gatherer maintains a local disk cache of files that were retrieved successfully. By default, the Gatherer's local disk cache is deleted after each successful completion. Since the remote server must be contacted whenever the Gatherer runs, we did not set up the Gatherer to run frequently. A typical value might be weekly or monthly, depending on how congested the network is and how important it is to have the most current data. By default, objects in the local disk cache expire after seven days.

7.2.7 The Broker

The Broker supports many different index/search engines. Harvest Broker and Glimpse were chosen because they have been used in many different search engines. Particularly for large Brokers, it is often helpful to use more powerful queries because

a simple search of a common term may take a long time. EnviroDaemon's query page contains several checkboxes that allow some control over query specification. Glimpse supports:

- Case-insensitive (lower and upper case letters are treated the same) and case sensitive queries;
- Matching parts of words, whole words, or phrases (like "resource recovery");
- Boolean (and/or) combinations of keywords;
- Approximate matches (e.g., allowing spelling errors); structured queries (allowing you to constrain matches to certain attributes);
- Displaying matched lines or entire matching records (e.g., for citations); specifying limits on the number of matches returned; and
- A limited form of regular expressions (e.g., allowing "wild card" expressions that match all words ending in a particular suffix).

EnviroDaemon allows the search to contain a number of errors. An error is either a deletion, insertion, or substitution of a single character. The Best Match option will find the match(es) with the least number of errors.

To allow popular Web browsers to mesh easily with EnviroDaemon, A Web interface to the Broker's query manager and administrative interfaces were implemented. This interface consists of the following: HTML files that use Forms support to present a graphical user interface (GUI), Common Gateway Interface (CGI) programs that act as a gateway between the user and the Broker, and "help" files for the user. The Broker also needs to run in conjunction with a Web server.

Users go through the following steps when using the Broker to locate information:

1. The user issues a query to the Broker.
2. The Broker processes the query, and returns the query results to the user.
3. The user can then view content summaries from the result set, or access the URLs from the result set directly.

The Broker retrieves indexing information from Gatherers or other Brokers through its Collector interface. A list of collection points is specified along with the host of the remote Gatherer or Broker, and the query filter if there is one. The Broker supports various types of collections. EnviroDaemon Brokers use indexing information retrieved from Gatherers through incremental collections.

7.2.8 Glimpse

Glimpse is an indexing and query system that searches through files very quickly. Glimpse supports most of `agrep`'s options which is a more powerful version of `grep` and includes approximate matching (e.g., finding misspelled words), Boolean queries, and even some limited forms of regular expressions. For example, if one is looking for a word such as "needle" anywhere in the file system, all that one needs to specify is, "glimpse needle," and all the lines containing the word "needle" will appear preceded by the file name. To use Glimpse, the files are first indexed with Glimpseindex, which is typically run every night.

The speed of Glimpse depends mainly on the number and sizes of the files that contain a match, and only secondarily on the total size of all indexed files. If the pattern is reasonably uncommon, then all matches will be reported in a few seconds, even if the indexed files total 500 MB or more.

Glimpse includes an optional new compression program, called Case, which permits Glimpse (and agrep) to search compressed files without having to decompress them. The search is significantly faster when the files are compressed.

Glimpse can search for Boolean combinations of "attribute-value" terms by using the EnviroDaemon's SOIF parser library. To search this way, the index is made by using the -s option of Glimpseindex (this is used in conjunction with other Glimpseindex options). For Glimpse and Glimpseindex to recognize "structured" files, they must be in SOIF format. Any string can serve as an attribute name. The scope of Boolean operations changes from records (lines) to whole files when structured queries are used in Glimpse.

Glimpse's index is word based. A pattern that contains more than one word cannot be found in the index. The way Glimpse overcomes this weakness is by splitting any multi-word pattern into its set of words, and looking for files in the index that contain the whole set.

7.3 ED with Hierarchical Search

This section describes some precise search approaches in the environmental domain that locate information according to syntactic criteria, augmented by the utilization of information in a certain context. The main emphasis lies in the treatment of structured knowledge, where essential aspects about the topic of interest are encoded not only by the individual items, but also by their relationships among each other. Examples for such structured knowledge are hypertext documents, diagrams, logical and chemical formulae. Benefits of this approach are enhanced precision and approximate search in EnviroDaemon.

EnviroDaemon has also been integrated with a Hierarchical Information Search Tool (HIST), which permits queries based on hierarchical structure. The rationale for this extension is that search conditions incorporating both keyword and structure-

based queries will help the user more precisely locate information. Figure 7.2 shows the front-end interface for EnviroDaemon with HIST. The query in the figure is to search for documents with the word "energy" in an `h1` tag.

Many search engines have their distinctive features: single domain, multi-domain, meta-search engines which are front-ends of other search engines. However, few of them make attempts to exploit the explicit underlying hypertext tags in a single domain. HIST, a search filtering tool, however, is able to exploit the full syntactic detail of any hypertext markup language and provide hierarchical query capability.

The essence of HIST is to permit end users to specify in which parts of a document a keyword should appear: in a document title, in a section header, somewhere in a paragraph, or in a table. This allows more precise control over the search process, and hence, results in much better selection of relevant material. Another salient feature of HIST is the idea of "fuzzy search." The documents returned by HIST do not have to be exact matches. Users have control over how similar the target document should be in the hierarchical query. Furthermore, at the speed that the Web technology standard is proceeding, an information retrieval tool must be able to meet the challenges of the next generation of the document standard on the Web. Since HIST is based on the Document Type Definition (DTD) Model, it is suitable for the new emerging document exchange standard for the Web, namely Extensible Markup Language (XML).

HIST does not replace the existing EnviroDaemon search engine. It enhances the EnviroDaemon for information searching on associated environment-related topics. For a general search on the Web, where the user has little knowledge about the nature of the document (in terms of document structure), we still encourage the use of EnviroDaemon. On the other hand, when the situation permits, the HIST

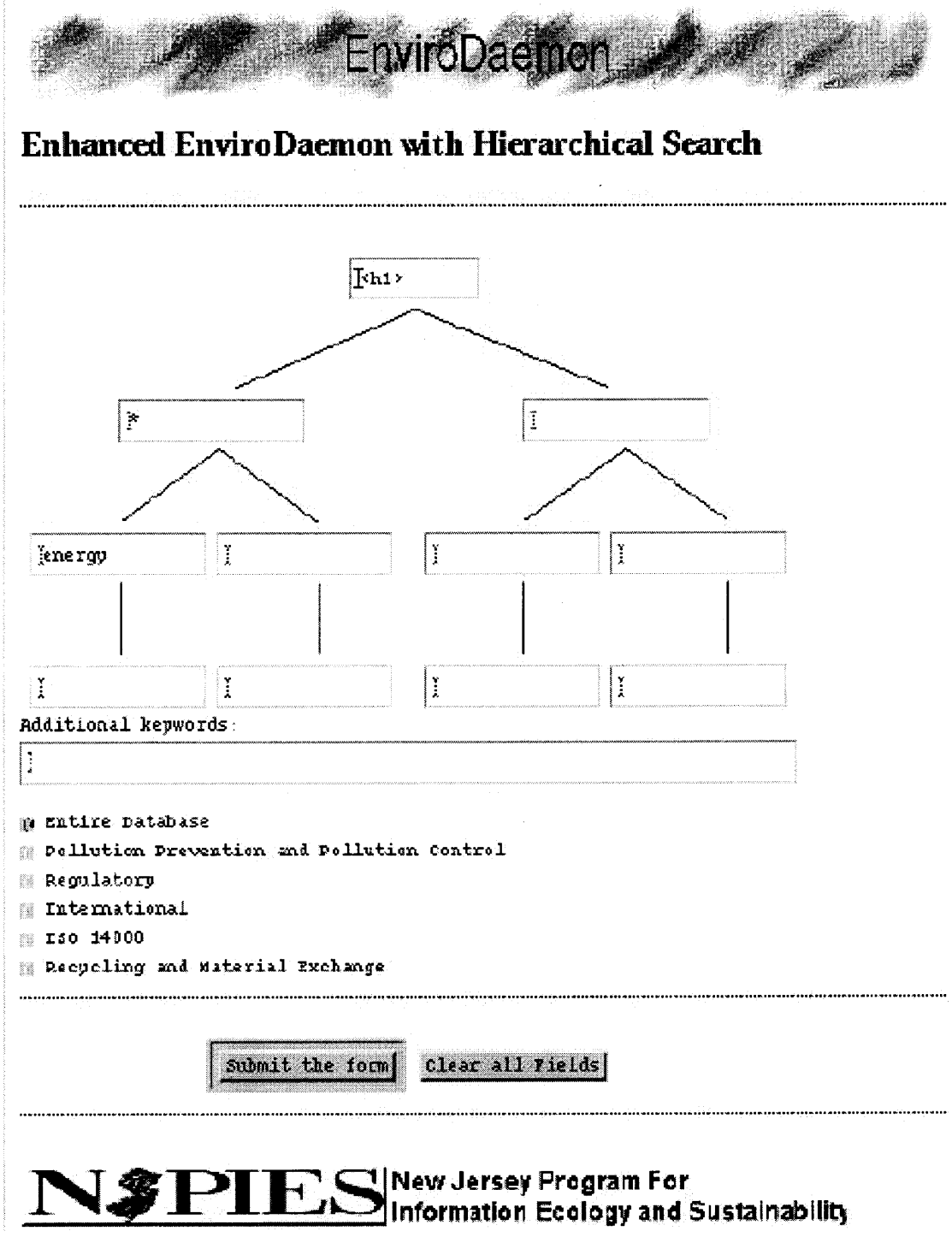


Figure 7.2 Front-end interface for EnviroDaemon with HIST.

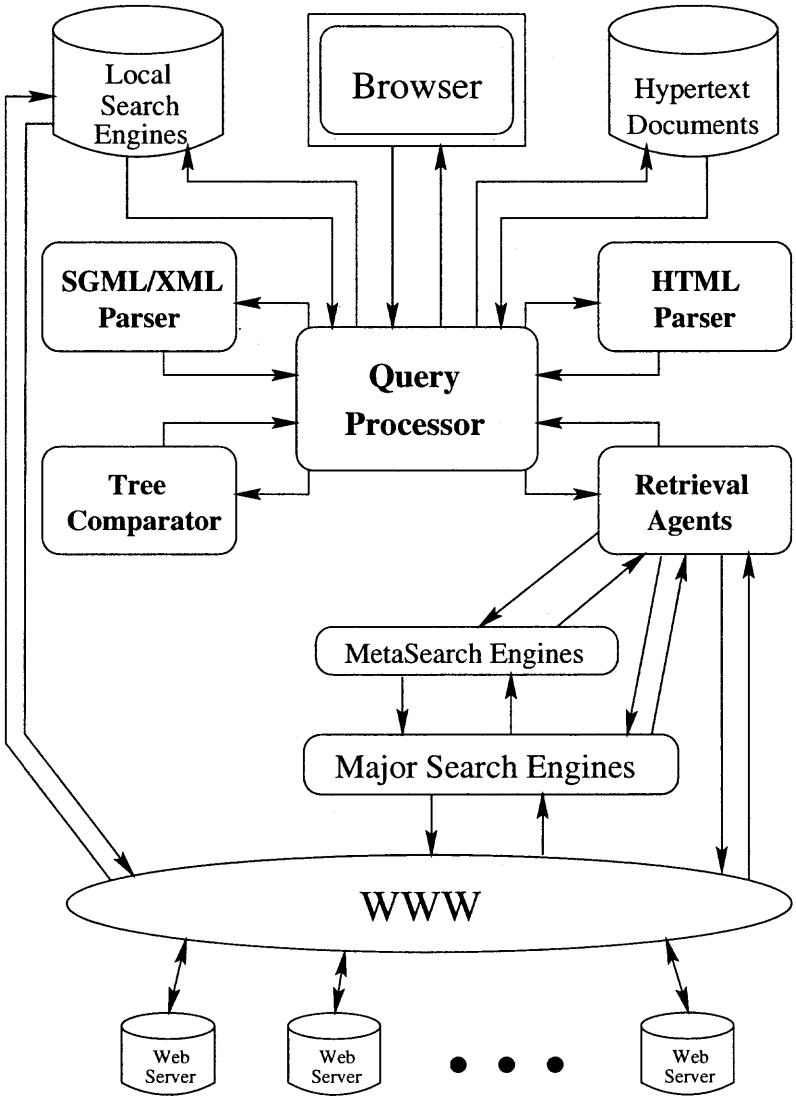


Figure 7.3 HIST system architecture.

tool will be extremely useful when the target document structure is partially or completely known.

HIST is composed of the following modules: Query Processor, SGML/XML Parser, HTML Parser, Tree Comparator, Retrieval Agents, and Front-end Interface. Figure 7.3 illustrates the HIST architecture.

The Query Processor handles queries from the Front-end Interface and invokes the Tree Comparator when necessary. The Retrieval Agents retrieve the actual HTML pages that satisfy keywords specified in the Front-end Interface from EnviroDaemon.

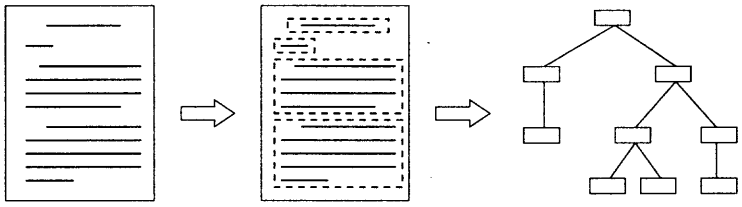


Figure 7.4 Converting a hypertext document to a labeled tree using DTD.

The HTML Parser translates the HTML pages into hierarchical tree structures based on the HTML DTD. The SGML/XML Parser translates the HTML pages into hierarchical tree structures based on a document's DTD. The Tree Comparator contains various approximate tree and string matching programs.

As many previous approaches have suggested, to analyze documents from the Web, the first step is to derive a scheme that describes the HTML page. While a specially designed schema provides a structural way of analyzing a hypertext document, much of the semantics associated with the document is lost after the transformation process. The approach differs from others in that it does not design a special schema, but, instead work with the DTD associated with hypertext documents.

With DTD in hand, each hypertext document can be parsed using its own DTD, capitalizing on the fact that the DTD provides not only the grammatical information, but also semantic information. For HTML documents, it can be parsed using HTML DTD 4.0. For the article type of SGML/XML documents, we can parse it with article DTD. The parsed tree structure of a typical document is illustrated in Figure 7.4. In this figure, the logical structure of the document is identified by the parser, which is denoted by dashed rectangles.

7.4 A Hierarchical Query Language

EnviroDaemon with HIST uses a query language called WAQL which is described in Chapter 3. Consider the hierarchical query in Figure 7.5(a). This query is to find the HTML pages containing the word "database" in an H1 header followed by a paragraph consisting of "object" followed by "relational." The * notation in the internal node of the query is a "variable length don't care" (VLDC) symbol, which represents an unspecified portion of a document as described below. The query may be issued when an individual intends to locate some HTML pages available on the Internet while conducting database-related research. Here the user places an emphasis on "database" and is interested in only those HTML pages having the word in an H1 header, rather than in any other place of a document.

To process such a hierarchical query using EnviroDaemon with HIST, it takes the conjunction of the keywords appearing in the query and invoke EnviroDaemon search engine to find the HTML pages containing these keywords. The EnviroDaemon search engine returns a collection of candidate uniform resource locators (URLs), ranked based on their relevance to the keywords. Duplicate URLs are deleted and a document corresponding to each matching URL is then retrieved using a libwww-Perl4 module, with time-outs set to 20 seconds to account for a busy network or failed connections. Each retrieved HTML document is then transformed into a hierarchical tree structure based on the DTD described earlier. The transformed tree structures then become candidate HTML trees to be compared with hierarchical query trees. Figure 7.5(b) shows an example tree for an HTML page. The tree is rooted, labeled and ordered (i.e., each node has a label and the order of siblings is important). An internal node represents an HTML tag and a leaf contains the associated text.

The system compares the query tree with each candidate HTML tree using a previously developed approximate tree matching algorithm [153] in conjunction with

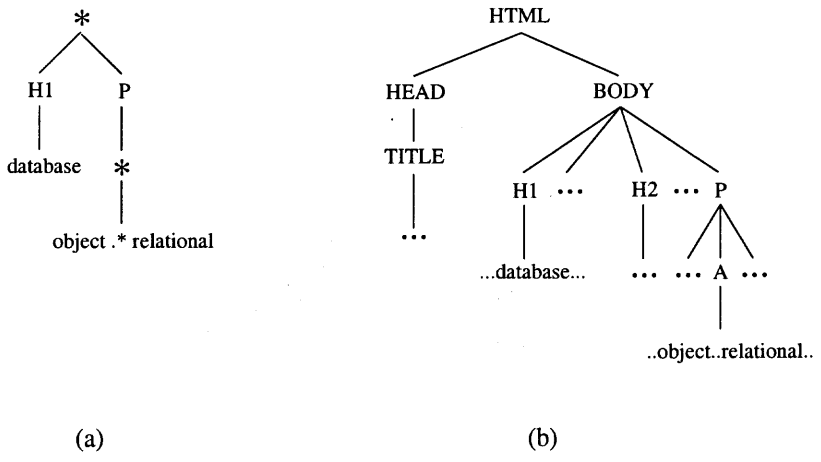


Figure 7.5 (a) An hierarchical search pattern; (b) an HTML tree.

regular expression matching on leaves when required. The URLs of qualified pages are then returned. In comparing the query tree with an HTML tree, a VLDC can be matched, at no cost, with a path or portion of a path in the HTML tree. The tree matching algorithm calculates the minimum edit distance between the query tree and the HTML tree after implicitly computing an optimal substitution for the VLDCs in the query tree, allowing zero or more cuttings at nodes from the HTML tree [153]. Cutting at a node n means removing the subtree rooted at n.

Given two trees T_1 and T_2, the algorithm runs in time $O(|T_1| \times |T_2| \times \min\{depth(T_1), leaves(T_1)\} \times \min\{depth(T_2), leaves(T_2)\}$. Thus, for example, in matching the query tree in Figure 7.5(a) and the HTML tree in Figure 7.5(b), the $*$ at the root in Figure 7.5(a) would be matched with (or instantiated into) the nodes HTML and BODY in Figure 7.5(b), and the $*$ underneath P in Figure 7.5(a) would be matched with the node A (i.e., the Anchor tag) in Figure 7.5(b). The nodes H1, "database" and "object.*relational" in Figure 7.5(a) would be matched with their corresponding nodes in Figure 7.5(b). All the other nodes in Figure 7.5(b) are cut.

7.5 Summary

EnviroDaemon has proven to be useful by rapidly returning results that are focused on pollution prevention (P2) and other environmental topics such as ISO 14000. By limiting its scope to P2-related Websites, EnviroDaemon can afford to be more thorough, searching whole documents rather than just titles and keywords. The resulting "hits" are embedded in up to ten lines of surrounding text; knowing the context of how the term was employed allows the user to thin out inappropriate sites without taking the time to actually visit them. EnviroDaemon obviates the need to look through one Website after another, or to know beforehand which sites are most promising.

In this chapter, EnviroDaemon with a Hierarchical Information Search Tool (HIST) on the Web is described. The system makes several contributions. It is context-specific and one of two environmentally directed search engines on the Web (the other being the proprietary EnviroSources [50]). It permits more detailed search than existing search engines. Additionally, it allows several kinds of approximate search at different levels:

- approximate search on the structural level, based on edit distance between structures;
- approximate search on the string level;
- variable length don't cares on the structural level; and
- any combination of the above.

It provides a query language that allows users to flexibly combine a variety of constructs and has proven to be useful for the Web environment where the languages employed are HTML. However, the flexibility of its design makes its integration with SGML and XML effortless. EnviroDaemon with HIST promises to save users from

drowning in an information sea and is generic enough to be useful for the Web and intranets. Future research that can be extended from this prototype is discussed in the next Chapter.

CHAPTER 8

SUMMARY AND FUTURE WORK

8.1 Summary of the Dissertation

In this dissertation proposal, a Web-Based Approximate Query System has been presented. The system makes several contributions.

1. It allows a more detailed search than with existing search engines.
2. It allows several kinds of approximate search at different levels:
   (a) approximate search on the structural level, based on "distance" between structures;
   (b) approximate search on the string level;
   (c) variable length don't care on the structural level; and
   (d) any combination of the above.
3. It provides an SQL-like query language that allows users to flexibly combine a variety of constructs in a Web query.
4. It is an useful tool for HTML, SGML, and XML documents.
5. It has been integrated with a domain specific search engine – EnviroDaemon.

In order to provide users with better text and data searching capabilities under XML, software developer must provide new tools that can handle XML-tagged text and data. In this sense, a whole new generation of applications will emerge. Tools from intelligent Web-agents to database creations without writing a database-specific code will soon be possible. New method and tool introduced here provide the first step towards analyzing this data-centric document standard.

8.2 Future Work

The future research will mainly focus on the further improvement of WAQS in the following directions:

- Weighted Querying Processing.

  This will allows user to add a weight to each node in the search pattern. The weight indicates the importance of the node in respect to other nodes in the query. Hence, this gives the user more control over the searching process.

- Improve the Visual Querying Environment.

  A new Java-based visual querying environment is currently being developed. This new interface will give the user a more intuit way of interacting with WAQS.

REFERENCES

1. S. Abiteboul and V. Vianu, "Queries and computation on the Web," in *Proceedings of International Conference on Database Theory*, Delphi, Greece, pp. 262–275, January 1997.

2. S. Abiteboul, P. Buneman, and D. Suciu, *Data on the Web: From Relations to Semistructured Data and XML*, Morgan Kaufmann, San Francisco, California, 2000.

3. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener, "The Lore query language for semistructured data," *International Journal on Digital Libraries*, vol. 1, no. 1, pp. 68–88, 1997.

4. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, New York, New York, 1994.

5. AltaVista, "http://www.altavista.com," May 2001.

6. A. Amir, M. Lewenstein, and N. Lewenstein, "Pattern matching in hypertext," in *Proceedings of the Workshop on Algorithms and Data Structures*, Halifax, Nova Scotia, Canada, pp. 160–173, August 1997.

7. AMORE, "http://www.ccrl.com/amore," May 2001.

8. M. Aragújo, G. Navarro, and N. Ziviani, "Large text searching allowing errors," in *Proceedings of the South American Workshop on String Processing*, Valparaíso, Chile, pp. 2–20, 1997.

9. AskJeeves, "http://www.askjeeves.com," May 2001.

10. P. Atzeni and G. Mecca, "Cut and paste," in *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Tucson, Arizona, May 1997.

11. P. Atzeni, G. Mecca, and P. Merialdo, "To weave the Web," in *Proceedings of the 23rd International Conference on Very Large Data Bases*, Athens, Greece, August 1997.

12. J. R. Bach, C. Fuller, A. Gupta, A. Hampapur, B. Horwitz, R. Jain, and C. Shu, "Virage image search engine: An open framework for image management," in *Proceedings of the SPIE Conference on Storage and Retrieval for Image and Video Databases*, San Jose, California, pp. 76–87, February 1996.

13. R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*, ACM Press, Addison–Wesley, New York, New York, 1999.

14. R. Bayer and E. M. McCreight, "Organization and maintenance of large ordered indexes," *Acta Informatica*, vol. 1, no. 3, pp. 173–189, 1972.

15. R. Bayer and K. Unterauer, "Prefix B-trees," *ACM Transactions on Database Systems*, vol. 2, no. 1, pp. 11–26, 1977.

16. N. J. Belkin and W. B. Croft, "Information filtering and information retrieval: Two sides of the same coin?," *Communications of the ACM*, vol. 35, no. 12, pp. 29–38, 1992.

17. T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret, "The World Wide Web," *Communications of the ACM*, vol. 37, no. 8, pp. 76–82, 1994.

18. BioCrawler, "http://www.biocrawler.com," May 2001.

19. C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz, "The Harvest information discovery and access system," in *Proceedings of the 2nd International World Wide Web Conference*, Chicago, Illinois, October 1994.

20. R. S. Boyer and J. S. Moore, "A fast string searching algorithm," *Communications of the ACM*, vol. 20, no. 10, pp. 762–772, 1977.

21. D. Briandais, "File searching using variable length keys," in *Proceedings of the AFIPS Western JCC*, San Francisco, California, pp. 295–298, 1959.

22. S. Brin and L. Page, "The anatomy of a large-scale hypertextual Web search engine," in *Proceedings of the 7th International World Wide Web Conference*, Brisbane, Australia, pp. 107–117, April 1998.

23. P. Buneman, "Semistructured data," in *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Tucson, Arizona, pp. 117–121, May 1997.

24. P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu, "A query language and optimization techniques for unstructured data," in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, pp. 505–516, June 1996.

25. P. Buneman, S. B. Davidson, and D. Suciu, "Programming constructs for unstructured data," in *Proceedings of the 5th International Workshop on Database Programming Languages*, Umbria, Italy, p. 12, September 1995.

26. M. Burner, "Crawling towards eternity: Building an archive of the World Wide Web," *Web Techniques Magazine*, vol. 2, no. 5, 1997.

27. J. Callan, "Searching for needles in a world of haystacks," *IEEE Data Engineering Bulletin*, vol. 23, no. 3, pp. 33–37, 2000.

28. R. G. Cattell and D. K. Barry, eds., *The Object Database Standard: ODMG 2.0*, Morgan Kaufmann, San Francisco, California, 1997.

29. S. Chakrabarti, M. van den Berg, and B. Dom, "Focused crawling: A new approach to topic-specific Web resource discovery," in *Proceedings of the 8th International World Wide Web Conference*, Toronto, Canada, pp. 1623–1640, May 1999.

30. R. Chandrasekar and B. Srinivas, "Gleaning information from the Web: Using syntax to filter out irrelevant information," in *Proceedings of the AAAI Spring Symposium on Natural Language Processing form the WWW*, Stanford, California, March 1997.

31. S. Chang, J. Smith, M. Beigi, and A. Benitez, "Visual information retrieval from large distributed online repositories," *Communications of the ACM*, vol. 40, no. 12, pp. 63–71, 1997.

32. 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 the IPSJ Conference*, Tokyo, Japan, pp. 7–18, October 1994.

33. J. Chen, D. J. DeWitt, F. Tian, and Y. Wang, "NiagaraCQ: A scalable continuous query system for Internet databases," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, Dallas, Texas, pp. 379–390, May 2000.

34. J. Cho, H. Garcia-Molina, and L. Page, "Efficient crawling through URL ordering," in *Proceedings of the 7th International World Wide Web Conference*, Brisbane, Australia, pp. 161–172, April 1998.

35. V. Christophides, S. Cluet, and G. Moerkotte, "Evaluating queries with generalized path expressions," in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, pp. 413–422, June 1996.

36. W. Cohen, A. McCallum, and D. Quass, "Learning to understand the Web," *IEEE Data Engineering Bulletin*, vol. 23, no. 3, pp. 17–24, 2000.

37. R. Cooley, B. Mobasher, and J. Srivastava, "Web mining: Information and pattern discovery on the World Wide Web," in *Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence*, pp. 558–567, 1997.

38. Cora, "http://cora.whizbang.com," May 2001.

39. M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. M. Mitchell, K. Nigam, and S. Slattery, "Learning to extract symbolic knowledge from the World Wide Web," in *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, 1998.

40. W. B. Croft, "What do people want from information retrieval?," *D-Lib Magazine*, vol. November, 1995.

41. Deja News, "http://www.deja.com," December 2000.

42. A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu, "XML-QL: A query language for XML," tech. rep., World Wide Web Consortium, 1998. `http://www.w3.org/TR/NOTE-xml-ql/`.

43. M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori, "Focused crawling using context graphs," in *Proceedings of the 26th International Conference on Very Large Data Bases*, Cairo, Egypt, September 2000.

44. Direct Hit, "http://www.directhit.com," May 2001.

45. Ditto, "http://www.ditto.com," May 2001.

46. D. Dreilinger and A. E. Howe, "Experiences with selecting search engines using metasearch," *ACM Transactions on Information Systems*, vol. 15, no. 3, pp. 195–222, July 1997.

47. S. T. Dumais, J. Platt, D. Hecherman, and M. Sahami, "Inductive learning algorithms and representations for text categorization," in *Proceedings of ACM CIKM International Conference on Information and Knowledge Management*, Bethesda, Maryland, 1998.

48. D. Eichmann, "The RBSE spider - Balancing effective search against Web load," in *Proceedings of the 1st International World Wide Web Conference*, pp. 113–120, 1994.

49. ElectricMonk, "http://www.electricmonk.com," May 2001.

50. EnviroSources, "http://www.envirosources.com," May 2001.

51. Excalibur, "http://www.excalib.com," November 2000.

52. Excite, "http://www.excite.com," May 2001.

53. C. Faloutsos and S. Christodoulakis, "Signature files: An access method for documents and its analytical performance evaluation," *ACM Transactions on Office Information Systems*, vol. 2, no. 4, pp. 267–288, 1984.

54. C. Faloutsos and S. Christodoulakis, "Description and performance analysis of signature file methods," *ACM Transactions on Information Systems*, vol. 5, no. 3, pp. 237–257, 1987.

55. C. Faloutsos, M. Flocker, W. Niblack, D. Petkovic, W. Equitz, and R. Barver, "Efficient and effective querying by image content," tech. rep., IBM, 1993.

56. FAST Search, "http://www.alltheweb.com," May 2001.

57. M. Fernandez and J. Robie, "XML Query Data Model," tech. rep., World Wide Web Consortium, 2000. `http://www.w3.org/TR/query-datamodel/`.

58. M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yonker, "Query by image and video content: The QBIC system," *IEEE Computer*, vol. 28, no. 9, pp. 23–32, 1995.

59. J. Fürnkranz, T. M. Mitchell, and E. Riloff, "A case study in using linguistic phrases for text categorization on the WWW," in *Working Notes of the 1998 AAAI/ICML Workshop on Learning for Text Categorization*, 1998.

60. H. Garcia-Molina, J. Widom, and J. D. Ullman, *Database System Implementation*, Prentice Hall, Upper Saddle River, New Jersey, 2000.

61. A. Glossbrenner and E. Glossbrenner, *Search Engines for the World Wide Web*, Peachpit Press, Berkeley, California, 1998.

62. E. J. Glover, S. Lawrence, W. P. Birmingham, and C. L. Giles, "Architecture of a metasearch engine that supports user information needs," in *Proceedings of the ACM CIKM International Conference on Information and Knowledge Management*, Kansas City, Missouri, 1999.

63. Go (Infoseek), "http://www.go.com," May 2001.

64. C. Goldfarb, *The XML Handbook*, Prentice Hall, Upper Saddle River, New Jersey, 1999.

65. R. Goldman and J. Widom, "WSQ/DSQ: A practical approach for combined querying of databases and the Web," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, Dallas, Texas, pp. 285–296, May 2000.

66. G. H. Gonnet, "Examples of PAT applied to the Oxford English Dictionary," tech. rep., University of Waterloo, 1987.

67. Google, "http://www.google.com," May 2001.

68. V. Harmandas, M. Sanderson, and M. Dunlop, "Image retrieval by hypertext links," in *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Philadelphia, Pennsylvania, pp. 296–303, July 1997.

69. M. J. Healey, J. T. Lewis, and G. Samtani, "Using EnviroDaemon to search the Internet for environmental information and building custom search engines," *Environmental Quality Management*, vol. 8, no. 1, pp. 103–109, 1998.

70. M. J. Healey, J. T. L. Wang, G. Chang, A. Revankar, and G. Samtani, "Precise environmental searches: EnviroDaemon with hierarchical information search," *Environmental Quality Management*, vol. 9, no. 1, pp. 51–62, 1999.

71. A. Heydon and M. Najork, "Mercator: A scalable, extensible Web crawler," *World Wide Web*, vol. 2, no. 4, pp. 219–229, 1999.

72. J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Language and Computation*, Addison-Wesley, New York, New York, 1979.

73. HotBot, "http://www.hotbot.com," May 1999.

74. Inktomi, "http://www.inktomi.com," May 2001.

75. Inktomi/NEC press release, "Web surpasses one billion documents," January 2000. http://www.inktomi.com.

76. R. M. Karp and M. O. Rabin, "Efficient randomized pattern–matching algorithms," tech. rep., Harvard University, 1984.

77. D. E. Knuth, J. H. Morris, Jr., and V. R. Pratt, "Fast pattern matching in strings," *SIAM Journal on Computing*, vol. 6, no. 1, pp. 323–350, 1977.

78. D. Konopnicki and O. Shmueli, "W3QS: A query system for the World Wide Web," in *Proceedings of the 21st International Conference on Very Large Data Bases*, Zurich, Switzerland, pp. 54–65, September 1995.

79. H. F. Korth and A. Silberschatz, *Database System Concepts*, McGraw Hill, New York, New York, 1991.

80. Z. Lacroix, A. Sahuguet, and R. Chandrasekar, "Information extraction and database techniques: A user-oriented approach to querying the Web," in *Advanced Information Systems Engineering, 10th International Conference CAiSE'98*, Pisa, Italy, pp. 289–304, June 1998.

81. Z. Lacroix, A. Sahuguet, R. Chandrasekar, and B. Srinivas, "A novel approach to query the Web," in *Proceedings of the ER97 Workshop on Conceptual Modeling for Multimedia Information Seeking*, Los Angeles, California, November 1997.

82. L. V. Lakshmanan, F. Sadri, and I. N. Subramanian, "A declarative language for querying and restructuring the Web," in *Proceedings of the 6th International Workshop on Research Issues in Data Engineering: Interoperability of Nontraditional Database Systems*, 1996.

83. L. S. Larkey and W. B. Croft, "Combining classifiers in text categorization," in *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Zurich, Switzerland, pp. 289–297, August 1996.

84. S. Lawrence, "Context in Web search," *IEEE Data Engineering Bulletin*, vol. 23, no. 3, pp. 25–32, 2000.

85. S. Lawrence and C. L. Giles, "Context and page analysis for improved Web search," *IEEE Internet Computing*, vol. 2, no. 4, pp. 38–46, July/August 1998.

86. S. Lawrence and C. L. Giles, "Searching the World Wide Web," *Science*, vol. 280, no. 5360, pp. 98–100, April 1998.

87. S. Lawrence and C. L. Giles, "Accessibility of information on the Web," *Nature*, vol. 400, no. 6740, pp. 107–109, 1999.

88. A. Layman, E. Jung, E. Maler, H. S. Thompson, J. Paoli, J. Tigue, N. H. Mikula,
and S. D. Rose, "XML-Data," tech. rep., World Wide Web Consortium, 1998. `http://www.w3.org/TR/1998/NOTE-XML-data-0105/`.

89. M. Lesk, "Going digital," *Scientific American*, pp. 58–60, March 1997.

90. A. Y. Levy, A. Rajaraman, and J. J. Ordille, "Querying heterogeneous information sources using source descriptions," in *Proceedings of the 22nd International Conference on Very Large Data Bases*, Bombay, India, pp. 54–65, September 1996.

91. M. Ley, "Database systems and logic programming bibliography server. http://www.informatik.uni-trier.de/~ley/db/index.html," May 2001.

92. Lycos, "http://www.lycos.com," May 2001.

93. Lycos Fast MP3 Search, "http://mp3.lycos.com," May 2001.

94. Lycos Pictures and Sounds, "http://multimedia.lycos.com," May 2001.

95. U. Manber and G. Myers, "Suffix arrays: A new method for on–line string searches," in *Proceedings of the ACM–SIAM Symposium on Discrete Algorithms*, San Francisco, California, pp. 319–327, 1990.

96. U. Manber, M. Smith, and B. Gopal, "WebGlimpse: Combining browsing and searching," in *Proceedings of USENIX Technical Conference*, Anaheim, California, pp. 195–206, January 1997.

97. U. Manber and S. Wu, "GLIMPSE: A tool to search through entire file systems," in *Proceedings of the USENIX Technical Conference*, San Francisco, California, pp. 23–32, January 1994.

98. O. A. McBryan, "GENVL and WWWW: Tools for taming the Web," in *Proceedings of the 1st International World Wide Web Conference*, pp. 79–90, 1994.

99. A. McCallum and K. Nigam, "A comparison of event models for naive Bayes text classification," in *Proceedings of the AAAI-98 Workshop on Learning for Text Categorization*, 1998.

100. A. McCallum, K. Nigam, J. Rennie, and K. Seymore, "A machine learning approach to building domain-specific search engines," in *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, pp. 662–667, August 1999.

101. E. M. McCreight, "A space-economical suffix tree construction algorithm," *Communications of the ACM*, vol. 23, no. 2, pp. 262–272, 1976.

102. J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom, "Lore: A database management system for semistructured data," *SIGMOD Record*, vol. 26, no. 3, pp. 54–66, September 1997.

103. S. Mehrotra, K. Chakrabarti, M. Ortega, Y. Rui, and T. S. Huang, "Towards extending information retrieval techniques for multimedia retrieval," in *Proceedings of the 3rd International Workshop on Multimedia Information Systems*, Como, Italy, pp. 39–45, September 1997.

104. A. O. Mendelzon, G. A. Mihaila, and T. Milo, "Querying the World Wide Web," *International Journal on Digital Libraries*, vol. 1, no. 1, pp. 54–67, April 1997.

105. MetaCrawler, "http://www.metacrawler.com," May 2001.

106. Midi Explorer, "http://www.musicrobot.com," May 2001.

107. MP3, "http://www.mp3.com," May 2001.

108. S. Mukherjea and J. Cho, "Automatically determining semantics for World Wide Web multimedia information retrieval," *Journal of Visual Languages and Computing*, vol. 10, pp. 586–606, 1999.

109. S. Mukherjea, K. Hirata, and Y. Hara, "Towards a multimedia World Wide Web information retrieval engine," in *Proceedings of the 6th International World Wide Web Conference*, Santa Clara, California, April 1997.

110. G. Navarro, "Improved approximate pattern matching on hypertext," in *Proceedings of LATIN '98: Theoretical Informatics, 3rd Latin American Symposium*, Campinas, Brazil, pp. 352–357, 1998.

111. Netscape Directory, "http://directory.netscape.com," May 2001.

112. K. Nigam, A. McCallum, S. Thrun, and T. M. Mitchell, "Learning to classify text from labeled and unlabeled documents," in *Proceedings of the 15th National Conference on Artificial Intelligence*, Madison, Wisconsin, 1998.

113. Northern Light, "http://www.northernlight.com," May 2001.

114. M. Ortega, Y. Rui, K. Chakrabarti, S. Mehrotra, and T. S. Huang, "Supporting similarity queries in MARS," in *Proceedings of the 5th ACM International Conference on Multimedia*, Seattle, Washington, pp. 403–413, November 1997.

115. L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the Web," in *Proceedings of the 7th International World Wide Web Conference*, Brisbane, Australia, pp. 161–172, April 1998.

116. R. W. Picard and M. Stonebraker, "Vision texture for annotation," *ACM Journal of Multimedia Systems*, vol. 3, no. 1, pp. 3–14, 1995.

117. B. Pinkerton, "Finding what people want: Experiences with the WebCrawler," in *Proceedings of the 2nd International World Wide Web Conference*, Chicago, Illinois, pp. 7–18, October 1994.

118. R. Ramakrishnan and J. Gehrke, *Database Management Systems*, McGraw-Hill, New York, New York, 1999.

119. E. J. Ray, D. S. Ray, and R. Seltzer, *The AltaVista Search Revolution*, Osborne/McGraw-Hill, Berkeley, California, 1998.

120. J. Rennie and A. McCallum, "Using reinforcement learning to spider the Web efficiently," in *Proceedings of the 16th International Conference on Machine Learning*, Bled, Slovenia, June 1999.

121. N. C. Rowe and B. Frew, "Automatic caption localization for photographs on World Wide Web pages," *Information Processing and Management*, vol. 34, no. 1, pp. 95–107, 1998.

122. N. C. Rowe, "Using local optimality criteria for efficient information retrieval with redundant information filters," *ACM Transactions on Information Systems*, vol. 14, no. 2, pp. 138–174, April 1996.

123. Y. Rubner and C. Tomasi, "Coalescing texture description," in *Proceedings of the ARPA Image Understanding Workshop*, pp. 927–936, 1996.

124. G. Salton and M. E. Lesk, *Introduction to Modern Information Retrieval*, McGraw–Hill, New York, New York, 1983.

125. SavvySearch, "http://www.savvysearch.com," May 1999.

126. R. E. Schapire, Y. Singer, and A. Singhal, "Boosting and Rocchio applied to text filtering," in *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Melbourne, Australia, pp. 215–223, August 1998.

127. Scour, "http://www.scour.net," May 2001.

128. Search Engine Watch, "http://www.searchenginewatch.com," May 2001.

129. E. Selberg and O. Etzioni, "Multi-service search and comparison using the MetaCrawler," in *Proceedings of the 4th International World Wide Web Conference*, Boston, Massachusetts, December 1995.

130. E. Selberg and O. Etzioni, "The MetaCrawler architecture for resource aggregation on the Web," *IEEE Expert*, vol. 12, no. 1, pp. 11–14, January/February 1997.

131. P. Sellers, "The theory and computation of evolutionary distances: Pattern recognition," *Journal of Algorithms*, vol. 6, pp. 132–137, 1980.

132. SherlockHound, "http://www.sherlockhound.com," May 2001.

133. J. R. Smith and S.-F. Chang, "Searching for images and videos on the World-Wide-Web," tech. rep., Columbia University, 1996.

134. J. R. Smith and S.-F. Chang, "Tools and techniques for color image retrieval," in *Proceedings of the IS & T/SPIE*, San Jose, California, pp. 426–437, March 1996.

135. Solvent Alternatives Guide-SAGE, "http://clean.rti.org," May 2001.

136. Sound Crawler, "http://www.soundcrawler.com," May 2001.

137. R. Srihari, "Automatic indexing and content-based retrieval of captioned images," *IEEE Computer*, vol. 28, no. 9, pp. 49–56, 1995.

138. J. Srivastava, R. Cooley, M. Deshpande, and P.-N. Tan, "Web usage mining: Discovery and applications of usage pattern from Web data," *SIGKDD Explorations*, vol. 1, no. 2, pp. 1–12, 2000.

139. Stream Search, "http://www.streamsearch.com," May 2001.

140. M. J. Swain and D. H. Ballard, "Color indexing," *International Journal of Computer Vision*, vol. 7, no. 1, pp. 11–32, 1991.

141. M. J. Swain, C. Frankel, and V. Athitsos, "WebSeer: An image search engine for the World Wide Web," tech. rep., Department of Computer Science, University of Chicago, 1996.

142. H. Tamura and T. Yamawaki, "Texture features corresponding to visual perception," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 6, no. 4, pp. 460–473, 1979.

143. S.-H. Teng, Q. Lu, M. Eichstaedt, D. Ford, and T. Lehman, "Collaborative Web crawling: Information gathering/processing over Internet," in *Proceedings of the 32nd Hawaii International Conference on System Sciences*, Maui, Hawaii, pp. 1–12, 1999.

144. E. Ukkonen, "Constructing suffix trees on-line in linear time," *Algorithmica*, vol. 14, no. 3, pp. 249–260, 1995.

145. WebCrawler, "http://www.webcrawler.com," May 2001.

146. WebSeek, "http://disney.ctr.columbia.edu/webseek/," May 1999.

147. WebSeer, "http://infolab.cs.uchicago.edu/webseer/," May 2001.

148. I. H. Witten, A. Moffat, and T. C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*, Van Nostrand Reinhold, San Francisco, California, 2nd ed., 1999.

149. World Wide Web Consortium (W3C), "http://www.w3c.org," May 2001.

150. S. Wu and U. Manber, "Fast text searching allowing errors," *Communications of the ACM*, vol. 35, no. 10, pp. 83–91, 1992.

151. Yahoo!, "http://www.yahoo.com," May 2001.

152. Yahoo! Image Surf, "http://gallery.yahoo.com," May 2001.

153. K. Zhang, D. Shasha, and J. T. L. Wang, "Approximate tree matching in the presence of variable length don't cares," *Journal of Algorithms*, vol. 16, no. 1, pp. 33–66, 1994.

Index