W3QS: A Query System for the World-Wide Web

David Konopnicki
konop@cs.Technion.AC.IL
Computer Science Department
Technion, Haifa, 32000, Israel

Oded Shmueli
oshmu@cs.Technion.AC.IL
Computer Science Department
Technion, Haifa, 32000, Israel

Abstract

The World-Wide Web (WWW) is an ever growing, distributed, non-administered, global information resource. It resides on the world-wide computer network and allows access to heterogeneous information: text, image, video, sound and graphic data. Currently, this wealth of information is difficult to mine. One can either manually, slowly and tediously navigate through the WWW or utilize indexes and libraries which are built by automatic search engines (called *knowbots* or *robots*).

We have designed and are now implementing a high level SQL-like language to support effective and flexible query processing, which addresses the structure and content of WWW nodes and their varied sorts of data. Query results are intuitively presented and continuously maintained when desired. The language itself integrates new utilities and existing Unix tools (e.g. grep, awk). The implementation strategy is to employ existing WWW browsers and Unix tools to the extent possible.

1 Introduction

The WWW was started to facilitate the sharing of data of various formats by physicists at CERN. The WWW supports pre-existing services (e.g. ftp) and many data formats (e.g. GIF for images and MPEG for movies). The WWW is organized as a set of HTTP (Hypertext Transmission Protocol) *servers*, where HTTP is a network protocol. A hypertext file format, HTML (Hypertext Markup Language), is used to construct links between documents, supporting a hypertext data organization. Files and services are identified over the network using URLs (Universal Resource Locator).

One of the popular tools for browsing the WWW is *Mosaic* which was developed by NCSA. Mosaic is a WWW *client*. Mosaic presents a point and click graphical user interface. Through Mosaic one may navigate through the WWW and obtain access to various services. Mosaic is basically a WWW *browser* and other browsers exist, e.g. Lynx which supports a vt100 terminal interface, perlWWW written in Perl, and the increasingly popular Netscape[13], a commercial browser which also has a public domain version.

The WWW can be viewed as a gigantic database (mostly read-only). The browsers enable roaming through sites of interest, however they are not *query processors*. In order to get a piece of information one basically needs to know where the data is located. To facilitate the search, there are certain indexes that are maintained over the WWW (no central control). These indexes are constructed by robots (e.g. Lycos[3], WWWW [11], WebCrawler [14]) that occasionally scan the WWW and construct indexes of interesting keywords. These indexes can be useful in locating information by using the browsers or other automatic tools. Some indexes may contain abstracting information in addition to keywords. Certain robots are used to maintain road maps to the WWW and others specialize in certain topics, e.g. mathematics.

Still, there is no high level query language for locating, filtering and presenting WWW-held information. In fact, the situation right now is analogous to that of a huge file system, or a document retrieval system, with many useful indexes but without a convenient facility for querying this information. One is thus forced to retrieve information manually through browsing and indexes, or write special purpose programs to obtain specific pieces of information.

Our main goal is to design and construct a high level querying and display facility for the WWW. This includes:

1. Specifying the syntax and semantics of a high level SQL-like query language.

Proceedings of the 21st VLDB Conference
Zurich, Switzerland, 1995

2. Providing as part of the language, a simple interface to user written programs and Unix utilities.

3. Providing advanced display facilities for gathered information, including special graphics, HTML browser presentation (e.g. Mosaic), and Unix directory tree representation.

4. Providing a view maintenance facility at a much higher level than robot maintained indexes.

At the present time, we have completed the design of the W3QL query language. The bulk of this paper is devoted to presenting the language and its capabilities. A prototype system will be constructed in the coming months. In the implementation, the following aspects will be emphasized:

1. Utilizing standard Unix services (e.g., file, grep, awk) for identification, filtering and formatting. This way, the amount of software which is written "from scratch" will be reduced.

2. Utilizing WWW robot constructed indexes and available libraries for optimizing query processing.

Related work

Due to the space limitations, we only briefly mention works that are very closely related to this paper. Improvements to hypertext management, such as query processing and views, were proposed in [8] and [2]. Analysis of files that have a strict inner-structure is treated in [1]. One can view the parsing, plus query evaluation, of [1] as a possible component of the condition library and the unparsing as a possible component of the formatting library (these libraries are components of W3QS). Several proposals for structure-specifying queries exist. The *Graphlog* graphic language is introduced in [4] to specify searched patterns. Application of the Graphlog query language to different systems, such as *NoteCards*, *gIBIS* and *HAM* are given in [4].

In [2], Beeri and Kornatzky define a logic-based language to state structure-specifying queries on a hypertext structure. Their formulae uniformly treat structural aspects, content aspects and boolean operations. Unlike [2], we separate structure and content specifications and concentrate only on forward links; we do not alter the global hypertext structure.

A query language on a dynamically changing hypertext structure is proposed in [12], but this language requires an understanding of the internal organization of each hypertext node.

Robot indexing initiatives are described in [11] [14]. An approach to information mining in the WWW is proposed in [5], but without defining a multi-purpose query language.

Paper organization

The paper is organized as follows. In section 2 we give an overview of the WWW and explain the difficulties in searching for data in the WWW. In section 3 we present the principles of W3QL, our query language. In section 4 we give examples of W3QL queries. In section 5 we describe the query system architecture. In section 6 we give a formal definition of the query language. In section 7 we present the library of functions used to implement search algorithms over the WWW and two programs (SQLPRINT and SQLCOND) used by the query system. Section 8 presents conclusions.

2 Difficulties in Searching the WWW

The World-Wide Web is succinctly described as a "Wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents" [10]. The interaction with the WWW is mainly based on *hypertext navigation*. Documents are usually presented to the user in *browser* clients. Clicking on an emphasized word of the hypertext leads to a request to a *HTTP server*, and the server then returns a new document. The documents are identified by their URL (Uniform Resource locators). A URL has three parts: The first part specifies the method of access to the data (e.g. ftp, HTTP), the second part is a network address of a server and the third part is a file name. For example, `http://www.cs.technion.ac.il/index.html` corresponds to the access page (the *home page*) of the WWW server at the Technion. Parameters may be passed to a HTTP server; for example:

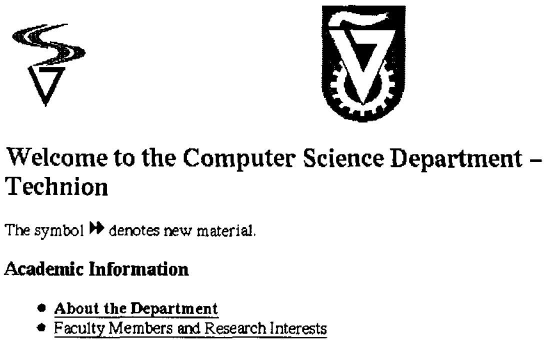
`http://webcrawler.cs.washington.edu/cgi-bin/WebQuery?flower`

queries the Webcrawler database with the keyword *flower* [14].

Documents written in the HTML format are standard ASCII files containing formatting codes [7]. For example, the HTML document in figure 1 is presented by the Mosaic browser as in figure 2. Currently, ac-

```
<TITLE>Technion - Computer Science Department Home Page</TITLE>

<PRE>
<IMG SRC="Gifs/csd.gif">          <IMG SRC="Gifs/technion.gif">
</PRE>

<H1>Welcome to the Computer Science Department - Technion</H1>

<P>
The symbol <IMG SRC="/Gifs/new.xbm"> denotes new material.

<H2>Academic Information</H2>
<UL>
<LI><A HREF="department.html"><B>About the Department</B></A>
<LI><A HREF="segel.html">Faculty Members
                                        and Research Interests</A>
...
```

Figure 1: Example of an HTML file at http://www.cs.technion.ac.il/index.html



Welcome to the Computer Science Department - Technion

The symbol ⏩ denotes new material.

Academic Information

- About the Department
- Faculty Members and Research Interests

Figure 2: Display of the HTML file in the Mosaic browser

cess to the WWW is based on navigationally-oriented browsers. This leads to the well-known "lost in cyberspace" phenomenon. Users are confronted with a large, unfamiliar, heterogeneous and constantly changing network. They have no systematic way to obtain information, because of the following reasons:

- There is no reliable road map for the WWW. The WWW is constantly growing and it becomes more and more difficult to locate specific information.

- It is difficult to analyze obtained information. The data found on the WWW is heterogeneous. Some files contain text, while others contain images, sounds or videos. These files are stored in various formats. Therefore, it is currently impossible to verify automatically whether a file satisfies a specific condition (For example, "Find all the images that contain a tree" or "Find all the articles written by A. Einstein").

- In a hypertext environment, the organization of documents conveys information. Nevertheless, it is cumbersome for users to search for information related to the organization of the hypertext. For example, if a hypertext document is composed of an index which points to different book chapters, and the chapters contain references to a bibliography, it is difficult to search for two chapters that have references to the same article in the bibliography.

Some systems address these problems as follows:

- Indexes are built to allow searching for documents, usually based on keyword matching. These indexes are built by humans or by automatic tools called *robots* or *knowbots* [5][14][11]. This approach is useful, but it has some drawbacks:
  - In indexing, there is, perhaps unavoidable, replication of information.
  - Indexes summarize the data, i.e. maintain the portions of the information which are considered important [9]. It is complicated to summarize images, graphics or sound data. So, indexes are most appropriate for text data.
  - Indexes do not do well in capturing the hypertext structure of indexed data.
  - Indexes become rapidly obsolete.

- There are some tools that help navigating the WWW. They are based on a graphical representation of some part of the network [6]. For example, a graph may represent the part of the network that the user is exploring. However this approach is limited:
  - This technique is an aid to navigation, but it does not provide a powerful information retrieval facility.
  - A graph can be useful for only a very small portions of the WWW. The whole WWW graph is too intricate for graphic representation to be useful.

To summarize, while some help exists, there is no comprehensive facility for querying the WWW.

3 W3QL: Declarative WWW Resource Finding

There are two basic types of hypertext queries [8].

- Content queries. These queries are based on the content of a single node of the hypertext. A condition that such a node must satisfy to be selected is stated.

- Structure-specifying queries. The information conveyed in the hypertext organization itself is queried. The entity selected by this type of query is a set of nodes (and links) from the hypertext structure that satisfy a given graph pattern [12][4][2].

Of course, one may combine these basic types and state more complex queries [2].

3.1 Content queries in W3QL

The information found on the WWW is mostly stored as unstructured data (files) while database systems are mainly concerned with structured data (such as tables). However, in order to give a database-like access to the WWW, information that is found in files must be queried. Three basic sorts of files exist:

1. Files that have a *strict inner structure*, such as BibTeX files or a Unix environment file, are in fact file representations of the content of a database

(in the BibTeX case, a bibliographic database). In such files, the semantics of the data is clearly linked to the syntax of the file. For example, the content of a BibTeX field is defined by the name of the field. Therefore, the database schema corresponding to such files is naturally conveyed by the grammar of the language in which the file is written [1].

2. *Semi-structured files* are text files that contain formatting codes (e.g. Latex or HTML). In such files, most of the semantic information is not coded in a formal way. Therefore, it is difficult to give a database abstraction for such files. However, it is possible to use the formatting codes to analyze their semantic content. For example, Latex files contain a *title* attribute that gives some indications about the subject of the file, and HTML files have specific attributes that give a semantic meaning to some of their parts (e.g. the **REL** attribute in anchors). Therefore, a group of standard attributes may be defined for semi-structured file format. The non-structured information of these files may be accessed using natural language analysis techniques.

3. In *raw files* the relation between the meaning of the file and its inner structure is difficult or impossible to ascertain. Such are executable files, pure text files and image and sound files.

The WWW contains mostly semi-structured files.
A standard W3QS program, called **SQLCOND**, is used to evaluate boolean expressions which are similar to the where-clause of a SQL query. **SQLCOND** will enable utilizing the information that is conveyed in file formats. Using **SQLCOND**, a user can select nodes from the WWW that satisfy certain conditions. For example:

```
node.format = Latex and
                    node.author = "A. Einstein".
```

Special attention must be paid to HTML files. In particular, we would like to state conditions about the anchors of HTML files in order to build structure-specifying queries.

3.2 Structure-specifying Queries in W3QL

The WWW can be viewed as a graph: each URL constitutes a node, and there is an edge from a node a to a node b if:

1. the file with URL a is in HTML format, and

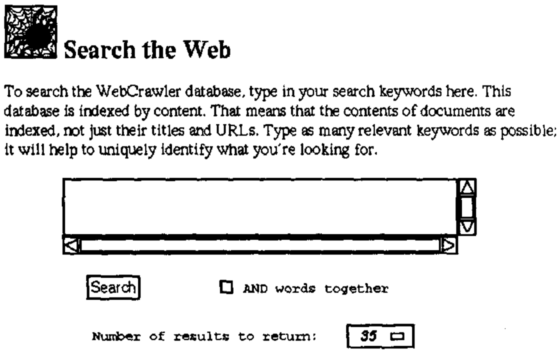
2. the file with URL a contains at least one anchor that points to node b.

A *path* is a set of nodes $\{v_1, \ldots, v_k\}$ such that (v_i, v_{i+1}), $1 \leq i < k$, are edges of the graph.
A *graph pattern* is a graph in which the nodes and the edges are annotated with conditions. These conditions correspond to the content queries defined above. The answer to a structure-specifying query is a set of subgraphs of the WWW, where:

1. Each subgraph must be *similar* to the query pattern. Similarity is defined formally below.

2. The nodes and the edges of the subgraph must satisfy the conditions specified for their corresponding images (see figure 3).

Browsers allow an extended use of HTML and of the HTTP protocol which is known as *form completion*. Forms are HTML files that contain elements, such as menus or answer fields, that the user can complete (see figure 4). Completed forms are returned to the server. This technique is used, for example, to register users or to send queries to online databases.



Search the Web

To search the WebCrawler database, type in your search keywords here. This database is indexed by content. That means that the contents of documents are indexed, not just their titles and URLs. Type as many relevant keywords as possible; it will help to uniquely identify what you're looking for.

Figure 4: Example of a form

Three main components are involved in supporting structure-specifying queries:

- Pattern definition. A pattern definition sub-language is used to describe the searched patterns. The sub-language is simple enough to be practical but it is able to express non-trivial queries. We intend to build a graphical interface for pattern specification.

- Search engine. Finding sets of nodes in the WWW that satisfy a pattern is not a trivial task. We do not intend to solve this problem as a general proposition. But, we do define a library of functions that facilitate the implementation of search algorithms. We also provide some basic vanilla-flavored search algorithms, for example breadth first search. The engine will use available indexes.

- Form completion. We need to deal with forms and other menu driven nodes in the WWW. A search engine must be able to find its way automatically through menus. Solving this problem as a general proposition is beyond the scope of this work,

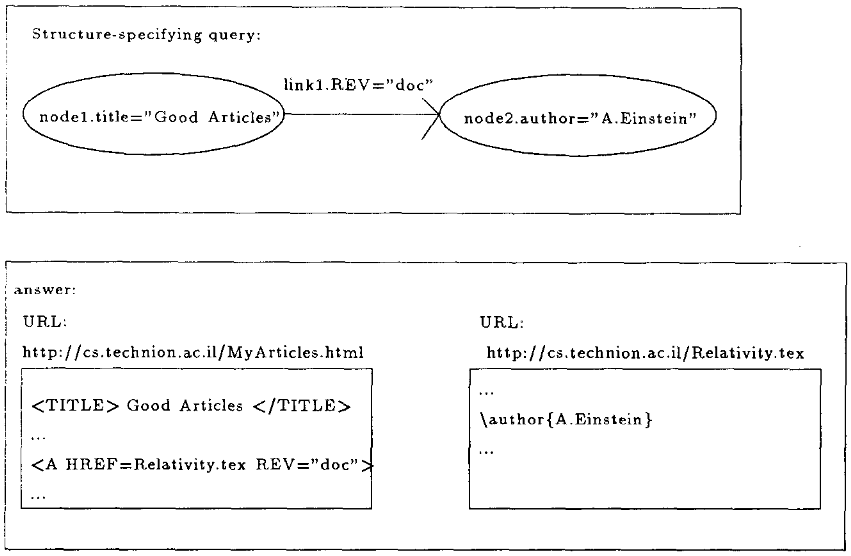

Structure-specifying query:
link1.REV="doc"
node1.title="Good Articles"
node2.author="A.Einstein"
answer:
URL:
http://cs.technion.ac.il/MyArticles.html
<TITLE> Good Articles </TITLE>
...
<A HREF=Relativity.tex REV="doc">
...
URL:
http://cs.technion.ac.il/Relativity.tex
...
\author{A.Einstein}
...

Figure 3: Example of a structure-specifying query

but we define some basic tools that automatically complete forms. The idea is to utilize keywords and experience gained in the past form completion activities.

3.3 Views: Dealing with Changing Information

We apply the concept of database views to the WWW. Views simplify the information shown to the user, letting the user only focus on what he/she needs. A view facility would greatly simplify work with the WWW. We intend that views be automatically refreshed in order to reflect changes. For example, a user can maintain a hypertext bibliography pointing to all the articles whose subject is "Physics". This bibliography will be maintained as new articles appear (or disappear) on the network.

Currently, information changes are difficult to locate on the WWW. Therefore, maintaining truly up-to-date views of the WWW is nearly impossible. So, we limit change monitoring to sites which are "related" to the query answer, and periodically start new searches in order to update the list of sites relevant to the query. This approach to view maintenance results in fairly accurate views and reasonable costs.

3.4 Query Results

The default format for a query answer is a table. The table contains the URLs of the nodes and links' attribute values that satisfy the query, organized in rows. This table is stored in a file and is presented using a browser. This allows the user to easily utilize the documents returned as the answer for the query (see Table 1).

node1.URL	link	node2.URL
DavidFiles.html	REV="article"	Relativity.tex
OdedFiles.html	REV="article"	Relativity.tex
OdedFiles.html	REV="article"	BlackBody.tex

Table 1: Example of an answer table. Result of a search of all the home pages, at the Technion site, that point to Latex files whose author is A. Einstein (We show here relative URLs, the complete URLs of these nodes begins with `http://cs.technion.ac.il`).

The files that constitute the answer to the query are saved in a directory structure (as describded in section 6). The directory structure may be operated upon using standard Unix commands. These can be specified in the Select Clause of the query. For example,
`cp */1 FirstResult`
will make a copy the first result row files into the directory `FirstResult`.

A standard W3QS program, called `SQLPRINT`, allows the formatting of results as a projection on files attributes. For example, `SELECT SQLPRINT n.author` where *n* is a Latex file, means that the query result contains only the author attribute of the Latex file.

4 Examples

Here are some examples of queries and views that can be expressed in W3QL.

4.1 Search for Articles

The query asks for articles written in Latex format by "A. Einstein". The query states that the search is performed on some indexes. These indexes are held at known sites on the WWW. Generally, these indexes are queried by completing a form, and the answer is an HTML page which contains the URLs of pages that might contain the information.

```
1     SELECT cp n2/* result;
2     FROM n1,l1,n2;
3     WHERE
4       n1 in ImportantIndexes.url;
5       FILL n1.form AS IN ImportantIndexes.fil
6                   WITH keyword = ''A. Einstein'';
7       SQLCOND (n2.format = Latex)
                AND (n2.author = ''A. Einstein'');
```

Line 4: The search is done using indexes. The file ImportantIndexes.url contains the list of the URLs of the indexes being searched. The URL of the file that corresponds to n1 must be in this file.

Line 2: The structure-specifying query is composed of: n1, the node that corresponds to the queried index, l1 the edge to the article returned by the index, and n2 the node that contains the article.

Line 5: Indexes are queried by completing a form. The form is completed *automatically* by using the information found in the file ImportantIndexes.fil, and the keyword portion of the query.

Line 7: The condition that n2 must satisfy in order to be selected.

Line 1: The selected articles are saved as files in the user's directory `result`.

4.2 Dynamically Maintained Views

This example refines the previous one. Now, a list of pointers to the articles is maintained, instead of saving the articles themselves. The list is defined as a view which is updated every week.

```
1     SELECT CONTINUOUSLY SQLPRINT n2.URL;
2     FROM n1,l1,n2;
3     WHERE
4       n1 IN ImportantIndexes.url;
5       FILL n1.form AS IN ImportantIndexes.fil
6                   WITH keyword = ''A.Einstein'';
7       RUN learnformat IF n1.form
                UNKNOWN IN ImportantIndexes.fil;
8       SQLCOND (n2.format = ''Latex'')
                AND (n2.author = ''A.Einstein'');
9     EVALUATED EVERY week;
```

Line 7: If a new index is inserted in ImportantIndexes.url and its format is unknown, the program `learnformat` is called. This program asks the user to describe how the form must be completed in order to complete it automatically the next time this form is encountered.

Line 9: This view is re-evaluated every week.

Line 1: A file containing the URLs of the articles is returned to the user using the SQLPRINT program. This list is continuously updated.

4.3 Search for Hypertext Patterns

This query refers to the hypertext structure described in figure 5. The query returns all the articles cited in the first chapter of the book. Each chapter include several pointers to the bibliography.
For example,

```
<A HREF=''http://cs.technion.ac.il/References.html#ref2''>
[Relativity]
</A>
```

means that the link **[Relativity]** leads to the label `ref2` in the file "References.html". In the reference file the labeled link looks like:

```
<A HREF=''http://cs.technion.ac.il/Relativ.tex'' NAME=''ref2''>
[Relativity, A. Einstein]
</A>.
```

The link **[Relativity, A. Einstein]** points to the article **Relativ.tex**.

```
1  SELECT cp art/* result;
2  FROM ind,l1,chap,l2,ref,l3,art;
3  WHERE
4   SQLCOND (ind.url =
5    ''http://cs.technion.ac.il/BookIndex.html'')
6    AND (chap.url = /.Chapter-1.html/)
7    AND (l2.HREF = /.\#$l3.NAME/);
8   USING BFS
```

Line 2: The from-clause describes the hypertext pattern that is searched for,

- ind is the index file,
- l1 is the pointer to the chapter,
- chap is the chapter file,
- l2 is the edge to the reference file,
- ref is the reference file,
- l3 is the edge to the article,
- art is the article that is returned in line 1.

Line 4-5: The URL of the index is given.

Line 6: We use a Perl [15] regular expression. The URL of the chapter ends with the string "Chapter-1.html" (".'' means "any sequence of characters").

Line 7: The HREF argument of the link to the reference must satisfy the Perl regular expression. This expression is interpreted as follows: . means that HREF begins any sequence of characters, **#$l3.NAME** is the label part of the URL (the prefix $ is used specify that a variable substitution must occur in the regular expression). In our example, `l2.HREF = ''http://cs.technion.ac.il/References.html#ref2''` while `l3.name = ''ref2''`. If the label of the link to the article, l3, is found in a link l2 from chapter 1, this means that the article is cited in chapter 1 and so it must be returned.

Line 8: The pattern is searched using the BFS algorithm.

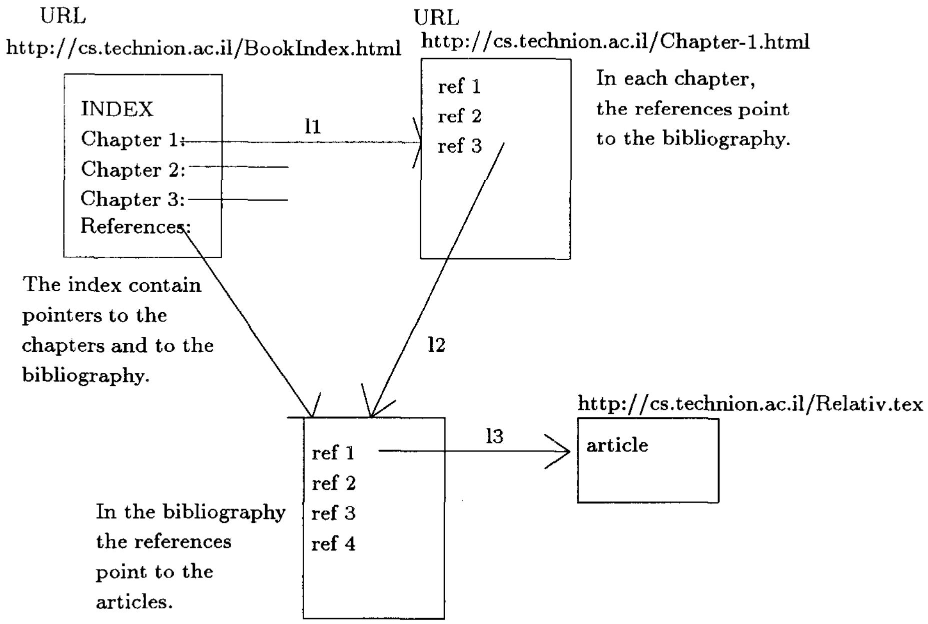

URL
http://cs.technion.ac.il/BookIndex.html
URL
http://cs.technion.ac.il/Chapter-1.html
INDEX
Chapter 1:
Chapter 2:
Chapter 3:
References:
l1
ref 1
ref 2
ref 3
In each chapter, the references point to the bibliography.
The index contain pointers to the chapters and to the bibliography.
l2
http://cs.technion.ac.il/Relativ.tex
ref 1
ref 2
ref 3
ref 4
l3
article
In the bibliography the references point to the articles.

Figure 5: Example of an HTML book

5 The W3QS Architecture

W3QS architecture reflects the following design principles:

- All the different modules should be easy to modify or enhance.
- The system should use existing UNIX utilities as much as possible.
- The system must present an intuitive interface to the naive user but also provide a full programming environment for the more sophisticated user. Advanced users should be able to use the query system to test information retrieval algorithms, or to manage hypertext information on their servers.

5.1 The System

The main modules of the system are (see figure 6):

- The query processor receives the query, and uses the algorithm clause (for example: USING BFS) to obtain a search program (1) which is found in the *RSP library*.
- The *remote search program* (RSP) executes the search. The RSP uses search processes to fetch information from the WWW (3). The RSP uses programs from the *condition library* (that contains, for example, SQLCOND) to select the information that corresponds to the query (4). When the RSP is done, it stores relevant information on the local disk (5).
- When the search ends, the query processor uses the *format library* (that contains, for example, SQLPRINT) to return the result to the user (6,7).

The libraries contain programs which are used during query processing. The functionality of W3QS can be extended by adding new programs to the libraries. The RSP library contains search programs, the condition library contains condition evaluation programs, and the format library contains result-formatting programs.

6 The W3QL Query Language

W3QL is a SQL-like language. Its syntax and semantics are described below.

6.1 Syntax

The core grammar for W3QL queries is described in table 2. Capital letters denote grammar terminals and lower case letters denote non-terminals. * means zero or more repetitions of a construct. [ ] means optional.

- `node_name` and `link_name` are C strings.
- `unix_program` is the name of an executable file.
- `file_name` is the name of a UNIX file.
- The `arguments` are separated by blanks.
- `regexp` is a Perl regular expression as defined in [15].
- `assignment` has the form `x=y`. For example, `topping = ''Anchovies''` or `time = 120`.

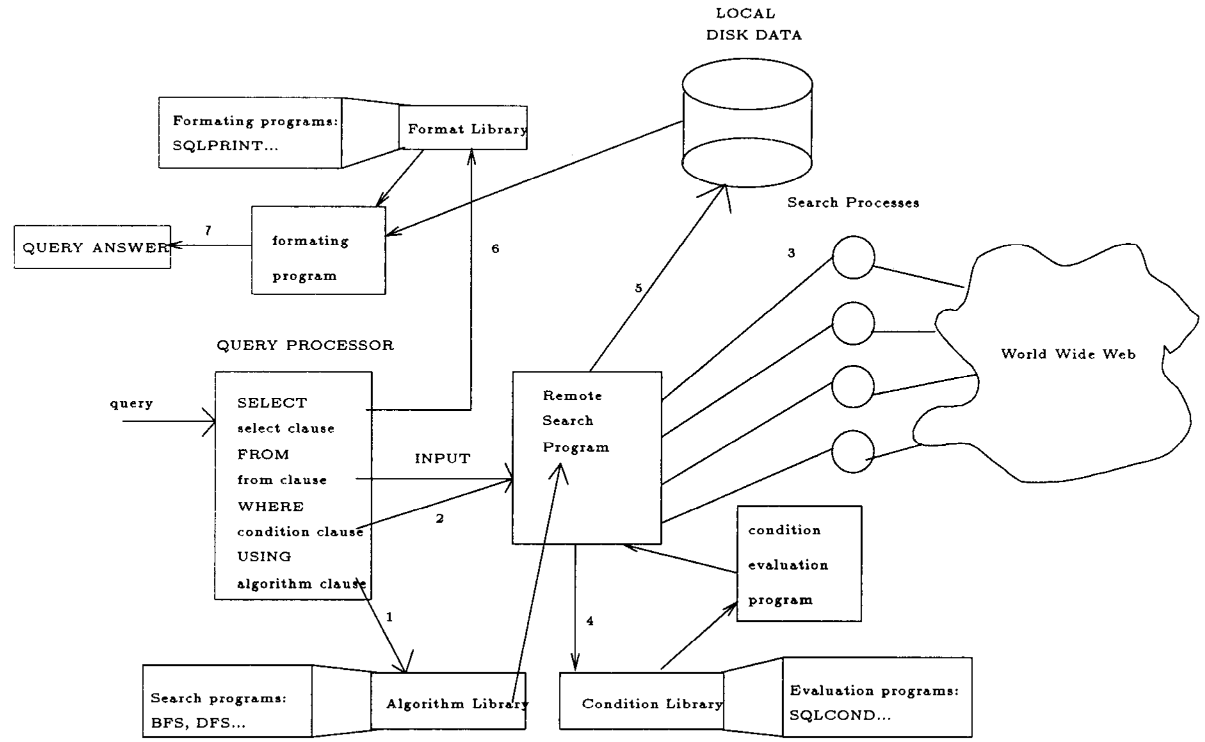


Figure 6: The system architecture

query:= SELECT [CONTINUOUSLY] [select_clause]
FROM from_clause
WHERE condition_clause
[USING algorithm_clause]
[EVALUATED EVERY time_unit]

select_clause:= statement*
statement:= unix_program argument*;
from_clause:= path*
path:= step* node_name;
step:= node_name, link_name, | (node_name, link_name), link_name,
condition_clause = condition*
condition:= statement = regexp; | node_name IN file_name; | FILL node_name AS IN file_name WITH assignment* ; | RUN unix_program IF node_name UNKNOWN IN file_name;
algorithm_clause:= statement

Table 2: Grammar of the query language

- **`time_unit`** is a period of time, e.g. **`week`** or **`5 minutes`**.

6.2 Semantics

6.2.1 Condition Clause

The *condition-clause* imposes conditions on the nodes and edges of the from-clause, and also provides the RSP navigation directives. The following condition types are possible:

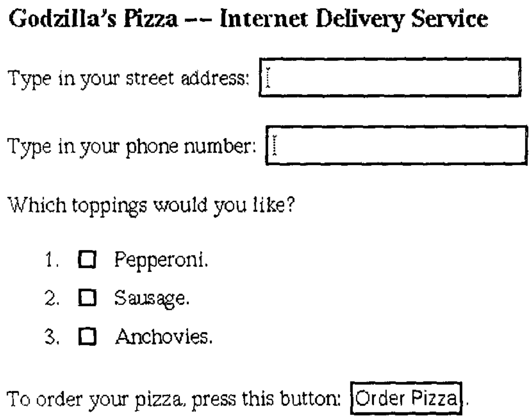
1. **`unix_program argument* = regxp`**. To evaluate this type of condition the RSP invokes an external UNIX program. The program input is defined by the arguments. An argument can be a node name, a link name or a node name with **`.form`** appended to it. The RSP runs the program, the arguments are replaced by file names. Here are some simple examples:

   - The statement **`diff n1 n2`** is evaluated by running **`diff templ temp2`**, where templ and temp2 contain the data extracted from the WWW nodes that are mapped to n1 and n2.
   - The **`.form`** termination is used to state that a condition applies to a node that contains a form.

   The program output is then compared to a Perl regular expression. If the output satisfies the regular expression the condition is *true*.

2. `node_name IN file_name`. This condition is *true* if the node name is mapped to a URL that is found in the file `file_name`. This type of condition can be used in two ways. It can be a directive to the RSP, meaning that the RSP must begin the search from a specific list of nodes, or it may be a condition, meaning that some node in the searched pattern must be located in a specific domain of the network as defined by the content of file_name.

3. `FILL node_name AS IN file_name WITH assignment*`. The WWW node mapped to node_name contains a form that must be filled automatically by the RSP using the data found in file_name and the assignments. For example, if the form in figure 7 was completed in the past, the condition `FILL n1 AS orderpizza.fil WITH topping=''Anchovies''` means that all the entries from the last completion activity (address and phone number), that were saved in the file `orderpizza.fil`, remain the same, only the topping is possibly different.



Godzilla's Pizza -- Internet Delivery Service

Type in your street address:

Type in your phone number:

Which toppings would you like?

1. □ Pepperoni.
2. □ Sausage.
3. □ Anchovies.

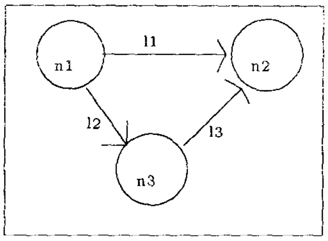
To order your pizza, press this button: Order Pizza.

Figure 7:
A form at *http://www.ncsa.uiuc.edu/SDG/Software-/Mosaic/Docs/fill-out-forms/example-3.html*

4. `RUN unix_program IF node_name UNKNOWN IN file_name`: The RSP calls an external program if it encounters an unknown form. This program may immediately ask the user to complete the form, may save this form as an unknown form to be completed later by the user, or may try to guess, using the form default values and the RSP's experience in form completion activities.

6.2.2 The Pattern Graph

The *from-clause* of the query describes the pattern graph that the RSP searches for. A pattern graph is described by a set of paths. The graph corresponding to a set of paths is built by identifying the nodes having the same name (see figure 8).



This graph can be described as either:

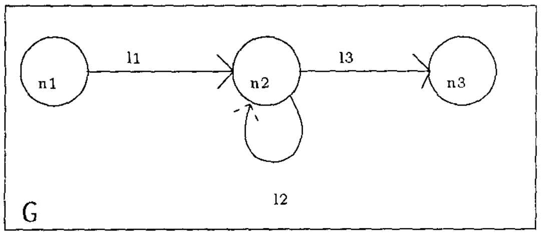
n1,l1,n2;
n1,l2,n3,
n3,l3,n2;

or

n1,l1,n2;
n1,l2,n3,l3,n2;

Figure 8: Example of a pattern graph.

In order to pose queries without knowing exactly how the hypertext is organized, we allow the definition of a special sort of paths called *unbounded length paths* (see figure 9). Unbounded length paths are represented as self-loops in the graph pattern.



This graph is describded as:

n1 l1 (n2 l2) l3 n3

Figure 9: Example of a pattern-specifying graph with an unbounded length path.

Formally, a *pattern-specifying graph* is a directed graph $G(V, E)$ where:

- There is at most one edge between any two nodes.
- There is at most one edge from a node to itself (self-loop).

A subgraph of the WWW $G'(V', E')$ is said to be *similar* to a graph pattern $G(V, E)$ if there is a mapping F such that:

- $F : V \cup E \mapsto 2^{V' \cup E'}$
- if $v \in V$ has no self-loop: $F(v) = \{v'\}, v' \in V'$
- if $v \in V$ has a self-loop: $F(v) = \{v'_1, \ldots, v'_n\}, v'_i \in V'$ and for $1 \le i \le n-1$, $(v'_i, v'_{i+1}) \in E'$. This also defines $First(F(v)) = v'_1$, $Last(F(v)) = v'_n$ (if $F(v) = \{v'\}$, $First(F(v)) = Last(F(v)) = v'$).

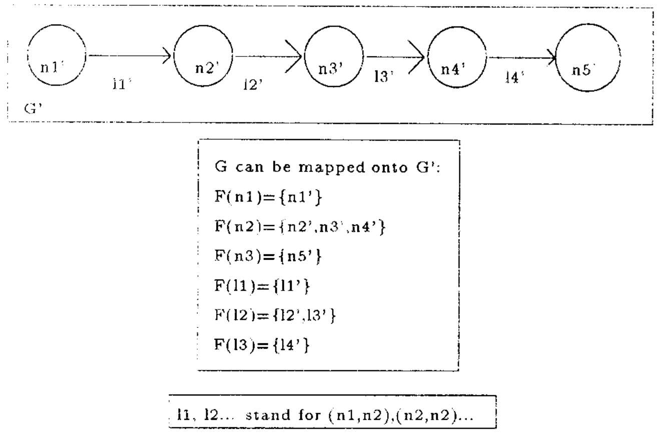


Figure 10: Example of a mapping.

- for all $v_1, v_2 \in V$ s.t. $v_1 \neq v_2$, $F(v_1) \cap F(v_2) = \emptyset$.
- $F((v_i, v_j)) = \{(Last(F(v_i)), First(F(v_j))\}$, $v_i, v_j \in V$, $v_i \neq v_j$.
- $F((v_i, v_i)) = \{(v'_1, v'_2), \dots, (v'_{k-1}, v'_k)\}$ where $F(v_i)$ is $\{v'_1, \dots, v'_k\}$ and $(v'_i, v'_{i+1}) \in E'$, $1 \leq i \leq k-1$.

Example: See figure 10.

The semantics of a W3QL query is as follows. Locate all subgraphs of the WWW that are similar to the pattern specified in the from-clause. Each subgraph, whose nodes satisfy the where-clause, defines a *row* in the returned table. The row contains the nodes and links used to exhibit similarity.

The following query searches for images in GIF format appearing in HTML files following only links with the REL argument equals to "example". REL is an HTML argument which is used to add meaning to hypertext links.

```
SELECT cp n3/* result;
FROM (n1,l1),l2,n3;
WHERE
SQLCOND (n1.format = HTML) AND
               (l1.REL="example") AND
(n3.name="*.gif");
```

6.2.3 Select Clause

This optional part of the query defines the form in which the query results should to be processed. Once the RSP finishes searching the network, the RSP saves the nodes that were extracted from the WWW and that are needed to be processed, in the local disk, organized as a directory tree (see figure 11). The select clause may run any UNIX program on these files so as to build the result of the query. For example, the following query saves all the lines from the selected nodes, which correspond to the node name n1, that contain the string "zoo", in the file "zoo.txt".

```
SELECT grep zoo n1/* > zoo.txt;
FROM n1,l1,n2;
WHERE
...
```

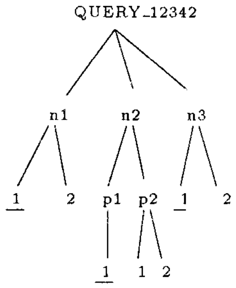


Figure 11: The file tree built for the pattern: n1,l1,(n2,l2),l3,n3. The RSP found two sets of nodes that form a solution to the query. The files that constitute the first set are: n1/1, n2/p1/1, n3/1 (underlined in the figure). The files that constitute the second set are: n1/2, n2/p2/1, n2/p2/2, n3/2. Note that p1 is a single node path and p2 is a two node path (both corresponding to (n2,l2) in the query).

6.2.4 Algorithm Clause

The *algorithm-clause* prescribes, to the query processor, which procedure should execute the search. These procedures will be implemented using the functions defined in the RSP Construction Kit. If the algorithm clause is not found in a query, a default search algorithm will be used.

7 The RSP construction kit and SQLCOND

7.1 The RSP Construction Kit Functions

The algorithm library includes data types and functions to assist programmers in easily implementing WWW search algorithms. The library has three groups:

- Structure-specifying graph functions. These functions build and use an internal representation of the structure-specifying graph. The functions include mapping a node to a URL, testing whether a URL satisfies a condition, etc.
- Access to, and description of parts of, the WWW. These functions help manage the processes that extract information from the WWW, instruct a process to bring a page, examine a page to find its links, etc.
- Extension to the library. The extension to the library includes functions to clone RSPs and

functions for inter-RSP communication in order to develop distributed search algorithms.

Figure 12 shows an example of a simple search algorithm implemented using the RSP Construction Kit. In this case, the structure-specifying graph is simply a single node.

```
SEARCH(PNODE *pn, WNODE *start, int level,
                                int limit)
  /* PNODE is a pattern specifying graph node,
     WNODE is a WWW node */
 {
  if (level >= limit) return;
  Assign(pn,start);
            /* start satisfies the conditions */
  if (TestCondition(pn)) AddToResult(pn);
  while (NextAnchor(start))
  SEARCH(pn,WWWGet(Anchor(start)),level+1,limit);
 }
```

Figure 12: A simple search algorithm

7.2 SQLCOND

The SQLCOND program uses the attributes of information found in a WWW node. This program uses the file name and the **file** UNIX utility (as in [9]) to "guess" the type of the file (e.g. Latex, Postscript). Then, a table is used to identify the attributes of the file. For example:

Format	Attribute	Begin Token	End Token
Latex	author	\author{	}
	title	\title{	}
HTML	title	<TITLE	</TITLE>
	anchor	<A	</A>
	HREF	<A. HREF="	"

We use the parsing capability of Perl to make a simple and quick analysis of the information found in files. This is a naive, yet effective, mechanism that may, sometimes, result in wrong answers. In that respect, our query processing is approximate.

SQLCOND permits stating conditions about node contents, or join conditions, for example **n1.author = n2.author**. In particular, SQLCOND permits stating conditions on the different nodes of an unbounded length path. The conditions that can be stated are a subset of the condition clauses defined in Perl [15].

8 Conclusions

We have designed and are currently implementing W3QS, a high level querying and displaying facility for the WWW. We have specified the syntax and semantics of a high level SQL-like query language called W3QL. W3QL is based on interfacing to user written programs and utilizing standard Unix services (e.g., file, grep, awk and W3QS standard programs) for identification, filtering and formatting. This makes W3QL extensible and customizable. This also makes the software construction task manageable.

We plan to provide W3QS with advanced display facilities for extracted information, including special graphics, HTML browsers visual presentation, and Unix directory tree representation. W3QS will include a view maintenance facility at a higher level than robot maintained indexes. We plan to utilize WWW robot-constructed indexes in query processing optimization.

At the present time, we have completed the design of the query language. A prototype W3QS system will be constructed in the coming months. The construction status and other developments concerning W3QL may be found in **http://www.cs.technion.ac.il/~konop**. This paper itself is accessible in this address.

References

[1] Abiteboul, S., Cluet, S., and Milo, T., Querying and updating the file. In *Proceeding of the 19th VLDB conference* (Dublin, Ireland, 1993).

[2] Beeri, C., and Kornatzky, Y., A logical query language for hypertext systems. In *Proceeding of the European Conference on Hypertext* (1990), Cambridge University Press, pp. 67–80.

[3] Carnegie Mellon University.,
Lycos, The Catalog of the Internet.
At http://lycos.cs.cmu.edu, 1995.

[4] Consens, M. P., and Mendelzon, A. O., Expressing structural hypertext queries in graphlog. In *Hypertext'89* (1989).

[5] De Bra, P. M. E., and Post, R. D. J., Searching for arbitrary information in the www: The fish search for mosaic.
In *Electronic Proceedings of the Second World-Wide Web Conference '94: Mosaic and the Web*
At http://www.ncsa.uiuc.edu/SDG/IT94-/Proceedings/WWW2_Proceedings.html, 1994.

[6] Dömel, P., Webmap: A graphical hypertext navigation tool.
In *Electronic Proceedings of the Second World-Wide Web Conference '94: Mosaic and the Web*
At http://www.ncsa.uiuc.edu/SDG/IT94-/Proceedings/WWW2_Proceedings.html, 1994.

[7] Graham, I. S., Html- documentation and style guide.
At http://www.utirc.utoronto.ca/HTMLdocs-/NewHTML/htmlindex.html, 1994.

[8] HALASZ, F. G., Reflections on notecards: Seven issues for the next generation of hypermedia systems. *Communication of the ACM 31*, 7 (1988).

[9] HARDY, D. R., AND SCHWARTZ, M. F., Customized information extraction as a basis for resource discovery. Tech. Rep. CU-CS-707-94, University of Colorado, 80309-0430, Mar. 1994.

[10] HUGHES, K., Entering the world-wide web: A guide to cyberspace. At ftp://ftp.eit.com, directory /pub/web.guide, 1994.

[11] MCBRYAN, O. A., Genvl and wwww: Tools for taming the web. In *Proceedings of the First International World Wide Web Conference* (May 1994), O. Nierstrasz CERN.

[12] MINOHARA, T., AND WANATABE, R., Queries on structure in hypertext. In *Foundation of data organization and algorithms, FODO'93* (1993), Lomet, Ed., Springer-Verlag, pp. 394–411.

[13] NETSCAPE COMMUNICATION CORPORATION., The Netscape Home Page.
At http://home.netscape.com, 1995.

[14] PINKERTON, B., Finding what people want: Experiences with the webcrawler.
In *Electronic Proceedings of the Second World-Wide Web Conference '94: Mosaic and the Web* At http://www.ncsa.uiuc.edu/SDG/IT94-/Proceedings/WWW2_Proceedings.html, 1994.

[15] SCHWARTZ, R. L., *Learning Perl.* A Nutshell Handbook. O'Reilly & Associates,Inc, 1993, ch. Regular expressions, pp. 83–98.