Applications of a Web Query Language

Gustavo O. Arocena
University of Toronto
gus@db.toronto.edu

Alberto O. Mendelzon
University of Toronto
mendel@db.toronto.edu

George A. Mihaila
University of Toronto
georgem@db.toronto.edu

February 17, 1997

Abstract

In this paper we report on our experience using WebSQL, a high level declarative query language for extracting information from the Web. WebSQL takes advantage of multiple index servers without requiring users to know about them, and integrates full-text with topology-based queries.

The WebSQL query engine is a library of Java classes, and WebSQL queries can be embedded into Java programs much in the same way as SQL queries are embedded in C programs. This allows us to access the Web from Java at a much higher level of abstraction than a bare HTTP request.

We have used WebSQL for applications related to Web maintenance and for the definition of the content of domain-specific text indexes.

Using the library, we have also implemented a client-server architecture that allows us to perform interactive intelligent searches on the Web from an applet running on a browser.

1 Introduction

Current practice for finding documents of interest in the Web depends on *browsing* the network by following links and *searching* by sending information retrieval requests to "index servers". The limitations of browsing as a searching technique are well-known, as well as the disorientation resulting in the infamous "lost-in-hyperspace" syndrome. As far as keyword-based searching, one problem with it is that users must be aware of the various index servers (over a dozen of them are currently deployed on the Web), of their strengths and weaknesses, and of the peculiarities of their query interfaces. To some degree this can be remedied by front-ends that provide a uniform interface to multiple search engines, such as Multisurf[HGN$^+$95], Savvysearch [Dre96], and Metacrawler [SE95].

A more serious problem is that these queries cannot exploit the topology of the document network. In the hypertext literature, the need for such queries is well documented. For example, in an often-cited paper on the limitations of hypertext systems, Halasz says: [Hal88]

> Content search ignores the structure of a hypermedia network. In contrast, structure search specifically examines the hypermedia structure for subnetworks that match a given pattern.

and goes on to give examples where such queries are useful.

In the Web context, suppose for example that we are looking for an IBM catalog with prices for personal computers. A keyword search for the terms "IBM," "personal computer," and "price," using MetaCrawler, returns 92 references, including such things as an advertisement for an "I Bought Mac" T-shirt and the VLDB '96 home page. If we know the URL the IBM home page is `www.ibm.com`, we would like to be able to restrict the search to only sites located on the same server as the IBM home page and directly or indirectly reachable from this page. With currently available tools, this is not possible because navigation and query are distinct phases: navigation is used to construct the indexes, and query is used to search them once constructed.

The WebSQL query language, instead, allows us to combine query with controlled automated navigation. WebSQL was proposed in [Mih96], along with a description of its first implementation.

When talking about automated navigation, emphasis must be put on *controlled* navigation. There are currently tens of millions of documents in the Web, and growing; and network bandwidth is still a very limited resource. It becomes important therefore to be able to distinguish in queries between documents that are stored on the local server, whose access is relatively cheap, and those that are stored on remote serves, whose access is more expensive. An analysis of the cost of query execution based on the notion of "query locality" was presented in [MMM96].

There are other reasons why queries that combine content and structure are important. We see a query language not primarily as an end-user tool, but as an aid in application development, just as SQL is used in combination with a programming language for building database applications. Many applications on the Web require accessing the structure defined by the links. For example, typical tasks in web site management include checking for dangling links, finding frequently used paths, finding unreachable documents, etc. We give examples of use of WebSQL for this kind of tasks in section 4.

The structure of the rest of the paper is as follows: in section 2 we present an intuitive introduction to WebSQL and the data model behind it (for a complete description of the language, including a formal specification of its semantics, refer to [Mih96]), in section 3 we give examples of interactive use of WebSQL for performing "intelligent search", in section 4 we give examples of queries used in applications using embedded WebSQL, in section 5 we give a rough description of the current implementation of the query engine and of the several available user interfaces and in section 6 we present our conclusions.

2 A Relational View of the WWW

WebSQL proposes to model the web as a relational database composed of two (virtual) relations: Document and Anchor (see Figure 1). The Document relation has one tuple for each document in the web and the Anchor relation has one tuple for each anchor in each document in the web. This relational abstraction of the web allows us to use a query language similar to SQL to pose the queries.

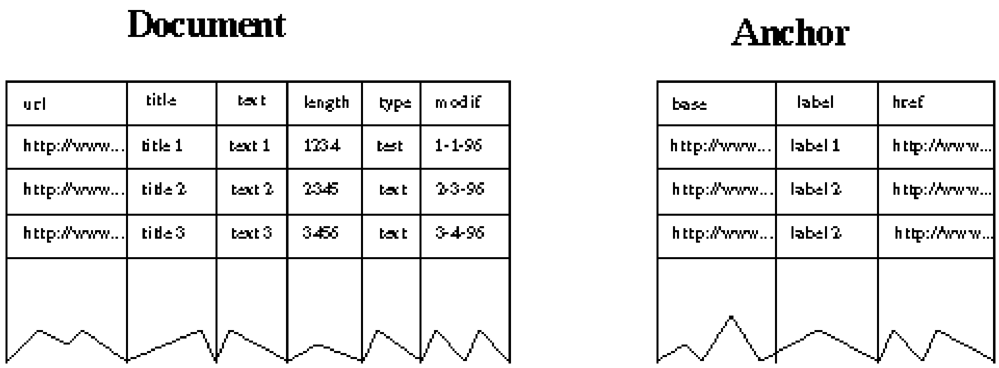

Figure 1: The Web According to WebSQL

If Document and Anchor were actual relations, we could use SQL to write queries. For example, the next SQL query would give us all the pairs of URLs of documents with the same title:

Query 1:

```
SELECT d1.url, d2.url
FROM Document d1, Document d2
WHERE d.title = d2.title
AND NOT (d1.url = d2.url)
```

But we cannot effectively compute queries like the previous one. Since the Document and Anchor relations are completely virtual and there is no way to enumerate all the documents in the web, we cannot operate on them as we would do with relations in a conventional relational database. For this reason, instead of operating directly on the virtual relations, WebSQL operates on materialized portions of them. A portion of the Document relation is materializable if we can somehow enumerate the URLs of all the documents it contains, and a portion of the Anchor relation is materializable if we can enumerate the URLs of the documents that contain the anchors in that portion. Once the materialized portions have been defined, we can manipulate them as traditional relations. For example, the next WebSQL query, which is a slight variant of Query 1, can be effectively computed:

Query 2:

```
SELECT d1.url, d2.url
FROM  Document d1 SUCH THAT d1 MENTIONS "something interesting",
Document d2 SUCH THAT d2 MENTIONS "something interesting"
WHERE d.title = d2.title
AND NOT (d1.url = d2.url)
```

The only difference between Query 2 and Query 1 is the addition of the "SUCH THAT ... MENTIONS ..." construct to the definition of each variable. This construct allows us to define a materialized portion of the web that contains all the documents that mention some keyword in their text. The implementation of MENTIONS consists of a query to one of the global index servers (AltaVista, Excite, HotBot, etc.).

3 Intelligent Search

As we explained in the introduction, queries to index servers are one of the two ways in which people search for information in the web. The other is manual navigation. WebSQL also allows us to automate the navigation process. For example, the query:

Query 3:

```
SELECT d.url, d.title
FROM  Document d SUCH THAT "http://www.somewhere.com" -> d
```

retrieves the title and the URL of all the documents that are pointed to from the document whose URL is "http://www.somewhere.com" and that reside in the same server as this one. The arrow between the URL and the variable specifies the path that must be followed in order to find the desired documents (in this case, a path of length one within the same server). Had we used '=>' instead of '->', we would have specified a path of length one but where the destination document is in a different server than the document of origin. We call '->' and '=>' local and global link types, respectively.

<table>
<tr><th>Regexp</th><th>Meaning</th></tr>
<tr><td>->->=></td><td>Path of length three composed of two local followed by one global link</td></tr>
<tr><td>(->|=>)</td><td>Path of length one, either local or global</td></tr>
<tr><td>->*</td><td>Local paths of any length</td></tr>
<tr><td>=> -> *</td><td>One global link followed by any number of local links</td></tr>
<tr><td>= |#>|-></td><td>Local path of length zero or one</td></tr>
</table>

Figure 2: Examples of Path Regular Expression

Another example of path specification is '->=>', which specifies paths of length two, composed of a local link followed by a global link. In general, link types can be combined in full regular expressions, yielding what we call Path Regular Expressions. The alphabet for these regular expressions is ->, =>, #>. The only symbol we have not mentioned yet is '#>', which represents an internal link within the same document. Table 2 gives more examples.

Using path regular expressions we can specify interesting search patterns. For example, the next query searches for pages related to databases in the web site of the Department of Computer Science of the University of Toronto:

Query 4:

```
SELECT d.url
FROM  Document d SUCH THAT "http://www.cs.toronto.edu" ->* d,
WHERE d.text CONTAINS "database"
OR d.title CONTAINS "database"
```

We can also combine queries to index servers with navigational queries. In the following query, we search for job opportunities for software engineers.

Query 5:

```
SELECT d1.url, d1.title, d2.url. d2.title
FROM Document d1 SUCH THAT d1 MENTIONS "employment job opportunities",
     Document d2 SUCH THAT d1 =|->|->-> d2
WHERE d2.text CONTAINS "software engineer"
```

We first query an index server to find pages that mention the keywords "employment job opportunities" and then, from each of these pages, we follow the local paths of length zero, one or two to find pages that contain the keywords "software engineer".

In the previous queries we have only used the Document relation. The information in anchors can also be useful for specifying searches. For example, suppose we want to find the pages describing the publications of some research group. If we know the URL of the home page of this group and we assume that the anchors pointing to the pages describing the publications contain the word "papers" in their label, we could write the query:

Query 6:

```
SELECT a1.href, d2.title
FROM Document d1 SUCH THAT "http://www.university.edu/~group" ->* d1,
     Anchor a1 SUCH THAT base = d1,
     Document d2 SUCH THAT a1.href -> d2,
WHERE a1.label CONTAINS "papers"
```

With query 6 we explore the pages in the web site in search for pages containing anchors with the word "papers" in their label. Once we find them, we add a tuple to the answer containing the URL and the title of the document pointed to from this anchor. The next query illustrates a different search strategy for the same problem: instead of looking at the text in the anchors, we look at the URL they refer to, and add to the answer the URL and the title of those documents that contain anchors pointing to compressed Postscript files.

Query 7:

```
SELECT d1.url, d1.title
FROM Document d1 SUCH THAT "http://www.university.edu/~group" ->* d1,
     Anchor a1 SUCH THAT base = d1,
WHERE filename(a1.href) CONTAINS "ps.gz"
      OR filename(a1.href) CONTAINS "ps.Z";
```

Observe that the path regular expression '–>*' in Query 6 matches paths whose length is one unit shorter than those matched by the same expression in the last query.

3.1 User-defined Link Types

We will now see how to extend the alphabet of path regular expressions to introduce predicates that test for properties of links other than locality to a server or to a document. This will allow us to express richer conditions on paths. WebSQL provides a syntax for specifying user-defined link types. Suppose we want to know which documents of a set of documents mention the word "Canada" in their titles. If these documents are arranged in a list such that the label of the link from a document to the next is the string "Next", we can define a link type as follows:

```
DEFINE LINK [next] AS label CONTAINS "Next";
```

A link will match the link type [next] only if its label contains the word "Next". Once we have defined the link type, we can use it in a path regular expression as we did with the predefined link types. The query that looks for the information we want is:

Query 8:

```
SELECT d.url
FROM Document d SUCH THAT "http://the.starting.document" [next]* d,
WHERE d.title CONTAINS "Canada";
```

4 Other Applications

As explained in the introduction, WebSQL was not only meant for being used interactively but also, and mainly, for developing applications and tools that could benefit from accessing the Web at a high level of abstraction. Both the interactive and the embedded uses of the language involve searches, but both kind of searches are of different nature. In interactive searches, as we saw above, we are likely to exploit the language capabilities to query index servers, test conditions on different parts of the documents, and to use path regular expressions to explore the neighborhood of certain documents. On the other hand, the searches we are likely to do when using embedded queries exploit the three elementary tasks WebSQL can do: a) given a URL, retrieve the corresponding document from the web; b) iterate over all the anchors in a document that has been retrieved and c) specify the elements of a collection of documents by means of a path regular expression.

Below we show examples of the kind of things that can be done by combining these three basic operations. All these examples have to do with tasks that are likely to be routinely done when maintaining a web site.

4.1 Finding Broken Links

The following query allows us to find the "broken links" in a page:

Query 9:

```
SELECT a.href
FROM Anchor a SUCH THAT base = "http://the.document.to.test"
WHERE protocol(a.href) = "http" AND doc(a.href) = null;
```

This query scans all the http anchors in a page and returns the value of the HREF field for those that point to inexistent documents (when a URL is valid, the function doc returns the tuple describing the document; otherwise it returns null). With a slightly different query, we can find all the broken links in pages that can be reached by recursively following links from the starting URL:

Query 10:

```
SELECT d.url, a.href
FROM Document d SUCH THAT "http://the.starting.document" ->* d,
Anchor a SUCH THAT base = d
WHERE protocol(a.href) = "http" AND doc(a.href) = null;
```

This query gives us a list of pairs of URLs such that the second one is a broken link in the page pointed to by the first one. Observe that, with respect to the above query, we have only added a component to the FROM clause for iterating over all the local documents that can be reached from the starting one, and a field to the result that identifies the document where the broken link occurs.

4.2 Defining the Content of a Full-text Index

Suppose you have a large set of HTML files that constitute the documentation for some software product and you want to build a full-text index for it. This documentation is arranged in a hierarchy where the root node is the table of contents, with links that point to the chapters, the front page of each chapter has links pointing to sections, and so on. By means of the following user-defined link type, we can restrict a search in such a way that only links that point to documents that are deeper in a hierarchy are traversed. We make use of the fact that the hierarchical organization is reflected in the path component of the URLs.

```
DEFINE LINK [Deeper] AS
 server(href) = server(base) AND path(href) CONTAINS path(base);
```

The [Deeper] link type ensures that the path of the document of origin is contained in the path of the destination document. Using Deeper, we can restrict the navigation to documents in the tree by going strictly from the root to the leaves (that is, avoiding back links, cross links and links to documents outside the hierarchy), as shown in the next query:

Query 11:

```
SELECT d.url, d.text
FROM Document d SUCH THAT "http://the.document.to.test" [Deeper]* d;
```

Query 11 retrieves a pair (url, text) for each document in the hierarchy. This is all we need to build the full-text index.

4.3 Finding References from Documents in Other Servers

As a last example, assume that you have a page with some links to pages in other sites and you want to know if your site is referenced from those pages or from the pages referenced by them. The following query would do it:

Query 12:

```
SELECT d.url
FROM Document d SUCH THAT "http://the.starting.doc" => d,
     Document d1 SUCH THAT d =>|->=> d1,
     Anchor a SUCH THAT base = d1
WHERE server(a.href) = "your.server"
```

5 Implementation

This section presents the current implementation of the WebSQL compiler, query engine, and user interfaces.

Both the WebSQL compiler and query engine are implemented as a set of Java [SM] classes, which form the *WebSQL class library*. The library can be used from any Java program.

The WebSQL system architecture is depicted in Figure 3.

The Compiler and Virtual Machine. The WebSQL compiler parses the query and translates it into a nested loop program in a custom-designed object language. The object

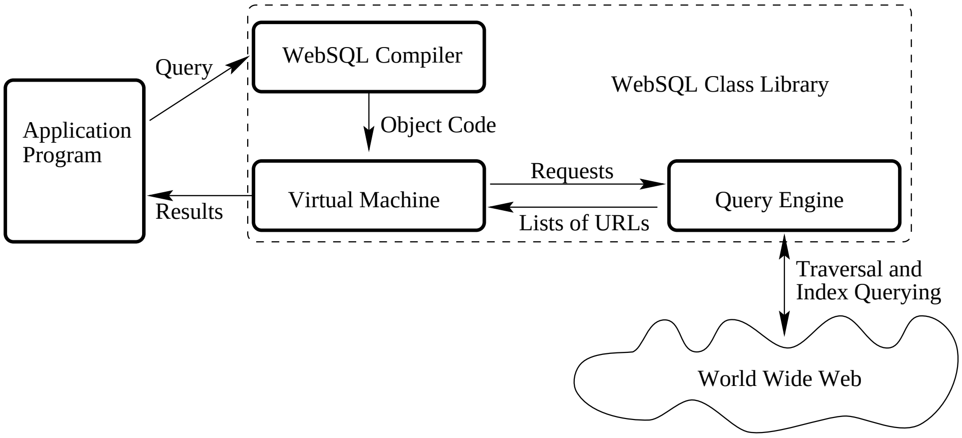

Figure 3: The Architecture of the WebSQL System

program is executed by an interpreter that implements a stack machine. Its stack is heterogeneous, that is, it is able to store any type of object, from integers and strings to whole vectors of Document and Anchor tuples. The evaluation of the range specified in the FROM clause is done via specially designed operation codes whose results are vectors of Document or Anchor tuples.

The Query Engine. Whenever the interpreter encounters an operation code corresponding to a range specifying condition, the query engine is invoked to perform the actual evaluation. Depending on the type of condition, this involves either sending a request to index servers or a depth-first traversal of a sub-part of the document network.

We have developed three different interfaces that allow us to use the language interactively. The simplest interface is an HTML form connected to a CGI script. The user can either fill in the form to assemble a query or type a complete WebSQL query directly. When the Submit button is pressed, the query is sent to the CGI script that invokes a stand-alone Java application running on our server. This application parses the query, and if no errors are found, hands it in to the query execution engine which produces the result as a list of tuples that gets formatted into an HTML table and is shipped back to the user. This interface, although slow and with limited user interaction, has the advantage that it can be used with any browser.

For Java-aware browsers (such as Netscape 3.0), we have developed a much more user-controlable front-end under the form of a Java applet. Figure 4 shows a snapshot of this applet running in a Netscape browser. Among the features of this applet we can mention:

- many typical queries can be generated by providing a URL or a keyword and then chosing a template;
- the result tuples are computed (and showed in a separate frame) incrementally, either one by one or in groups of five;
- the execution can be interrupted at any time;

- if the query results contain URLs, one can click on them and the corresponding document is shown in a separate browser window;

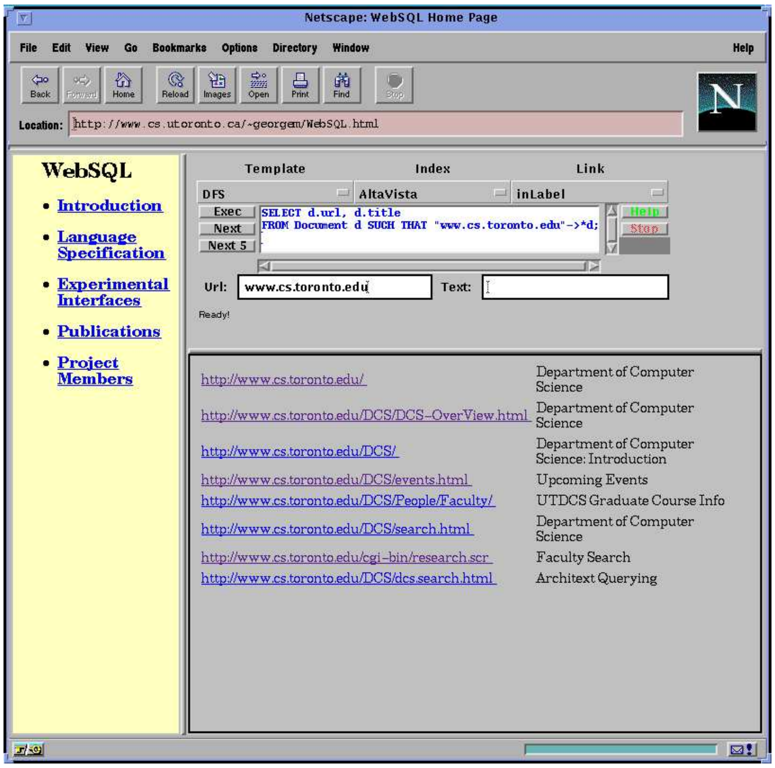

Figure 4: The WebSQL Applet

Due to security restrictions in accessing remote servers, an applet cannot compute the results of a query directly. Therefore, we designed this applet using the client-server paradigm. Thus, we have split the logic of the application into two separate processes: the applet itself, responsible for the user interaction, and a server process (running on the same machine as the CS department Web server hosting the HTML page containing the applet), that does the actual processing. The applet communicates with the server through a custom designed protocol that allows the user total control over the query execution.

Finally, we have yet another interface, this time a stand-alone Java application that, once installed on the user's machine, runs locally and provides similar functionality as the applet.

Also, we are making available (in compiled form) the entire library of Java classes needed to develop applications on top of our system, together with Java source files for some sample applications using WebSQL. This library can be thought of as an application programming interface (API) to the WebSQL engine. Our declared intention is to provide a tool for programatic access to the Web, in a fashion similar to the use of embedded SQL for regular databases.

All these interfaces, together with the complete syntax specification of the language and a comprehensive collection of examples are publicly available from the WebSQL home page (`http://www.cs.toronto.edu/~georgem/WebSQL.html`).

6 Conclusions

We have overviewed the main features of WebSQL, a declarative language for querying the World Wide Web introduced in [Mih96] and [MMM96] and we illustrated some practical applications of the language. Thus, we have shown how WebSQL can be used to automate routine Web site maintenance procedures, such as finding broken links in a collection of documents, finding references in documents in other servers to local pages, and defining the scope of a full-text index.

We described our prototype Java-based implementation of a WebSQL query execution engine together with several user interfaces and the embedded WebSQL application programming environment. Although fully accessible from the Web, our complex user interfaces are not meant to replace simple keyword-based search engines. Just as SQL is by and large not used by end users, but by programmers who build applications, we see WebSQL as a tool for helping build Web-based applications more quickly and reliably.

In our data model the text of an HTML document is a monolithic object, and therefore its analisys and interpretation is limited to simple text matching techniques. This is due to the fact that HTML is optimized for presentation, not for describing semantic content, which makes information extraction from arbitrary documents impossible. We are currently working at several extensions of the language that will make use of the document strucure when it is known.

There is also a great deal of scope for query optimization. We do not currently attempt to be selective in the index servers that are used for each query, or to propagate conditions from the WHERE to the FROM clause to avoid fetching irrelevant documents. It would also be interesting to investigate a distributed architecture in which subqueries are sent to remote servers to be executed there, avoiding unnecessary data movement.

References

[Dre96] Daniel Dreilinger. Savvysearch home page. 1996. `http://guaraldi.cs.colostate.edu:2000/`.

[Hal88] Frank G. Halasz. Reflections on Notecards: Seven issues for the next generation of hypermedia systems. *Comm. of the ACM*, 31(7):836–852, 1988.

[HGN+95] Masum Z. Hasan, Gene Golovchinsky, Emanuel G. Noik, Nipon Charoenkitkarn, Mark Chignell, Alberto O. Mendelzon, and David Modjeska. Visual Web surfing with Hy+. In *Proceedings CASCON '95*, Toronto, November 1995. IBM Canada. `ftp://db.toronto.edu/pub/papers/cascon95-multisurf.ps.Z`.

[Mih96] G. A. Mihaila. WebSQL - an SQL-like query language for the world wide web. Master's thesis, University of Toronto, 1996.

[MMM96] A. Mendelzon, G. Mihaila, and T. Milo. Querying the World Wide Web. In *Proc. of PDIS'96*, 1996. to appear.

[SE95] Erik Selberg and Oren Etzioni. Multi-service search and comparison using the MetaCrawler. In *Proceedings of the Fourth Int'l WWW Conference*, Boston, December 1995. `http://www.w3.org/pub/Conferences/WWW4/Papers/169`.

[SM] Sun Microsystems. Java (tm): Programming for the internet. `http://java.sun.com`.