A Declarative Language for Querying and Restructuring the Web

Laks V.S. Lakshmanan
Department of Computer Science
Concordia University
Montreal, Quebec
laks@cs.concordia.ca

Fereidoon Sadri
Department of Mathematical Sciences
University of North Carolina at Greensboro
Greensboro, NC
sadri@uncg.edu

Iyer N. Subramanian
Department of Computer Science
Concordia University
Montreal, Quebec
subbu@cs.concordia.ca

Abstract

World Wide Web is a hypertext based, distributed information system that provides access to vast amounts of information in the internet. A fundamental problem with the Web is the difficulty of retrieving specific information of interest to the user, from the enormous number of resources that are available. In this paper, we develop a simple logic called WebLog that is capable of retrieving information from HTML (Hypertext Markup Language) documents in the Web. WebLog is inspired by SchemaLog, a logic for multidatabase interoperability. We demonstrate the suitability of WebLog for (a) querying and restructuring Web information, (b) exploiting partial knowledge users might have on the information being queried, and (c) dealing with the dynamic nature of information in the Web. We illustrate the simplicity and power of WebLog using a variety of applications involving real-life information in the Web.

1 Introduction

The Setting: World Wide Web (WWW) is revolutionizing the information age. Its strong impact on the end user and the many potential benefits it augurs has spurred tremendous research on a whole gamut of issues related to storing, retrieving and manipulating information on the Web. The Web is emerging to be one of the most exciting topic of active research, bringing together researchers from diverse areas such as communications, electronic publishing, language processing, and databases.

The Problem: How can a user (seeking specific information she is interested in) retrieve and perhaps restructure information in the Web? In real life, she might also have clues on the information of interest to her, such as its likely location in the Web, its structure, some keywords, or patterns relevant to it. We would like to enable the user to specify such information as part of the query, make use of it to search the Web, and return a helpful response “satisfying” the query. The framework we propose is aimed at addressing such a need.

Most information in the Web is present in the form of Hypertext Markup Language (HTML) documents. In this paper, we investigate how information in HTML documents could be retrieved and restructured in a declarative way. In the subsequent discussions, we use the terms Web documents and HTML documents interchangeably – they mean the same for us.

Querying the Web – State of the Art: The compelling need for querying information in the Web has led to the development of a number of tools that based on some keywords specified by the user, search the Web and return information related to the keywords. Lycos, WWW (World Wide Web Worm), NetFind, and InfoSeek are some examples. These search tools make use of a pre-compiled catalog (also called a reference database) of information available in the Web to answer user queries. Searches can be performed on titles, reference hypertext, URL1 etc. These tools typically have two components: resource locator and search interface. The resource locator is run periodically to gather information from the Web and create the catalog. The search interface provides fast access to information in the catalog. In Section 5, we discuss in detail two popular search tools – WWW, and Lycos. More information on Web search tools in general can be found in [16, 8].

Currently available Web search tools suffer from the following drawbacks:

• Partial knowledge that a user might have on the information she is querying about is not fully exploited.

As an example, consider the user searching for call for papers (CFPs) of all database conferences. In particular, she is interested in knowing the submission deadline of various conferences. She also has the following information on CFPs.

(i) Most database related CFPs can be obtained by navigating links from the page of Michael Ley at http://www.informatik.uni-trier.de/ley/db/index.html.

(ii) Each CFP has a pattern of the form "... Date ... (date) ..." (or a ‘synonym’ of this pattern), where (date) is potentially the submission deadline.

We would like the user to be able to express such information when she specifies the query. The

---

*This research was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada, the Fonds Pour Formation De Chercheurs Et L'Aide A La Recherche of Quebec, and University of North Carolina at Greensboro.

1Uniform Resource Locator, an address that specifies the location of a resource in the Web.
query processor should also make use of this information to perform an efficient search.

- The restructuring ability these tools provide is limited, or none.

Continuing on the example above, the user might want to group together links to CFPs that have submission deadline in the same month. The querying tool should allow for specifying in an ad hoc manner the format in which the answer should be presented.

- There are no facilities for exploiting externally available libraries of string and document processing functions.

Various resources for processing documents and strings are available in the form of external functions. The querying medium should be able to exploit these readily available tools for allowing powerful and natural ways of querying the Web.

- Query result quality is compromised by the highly dynamic nature of Web documents.

In most search tools, once the page is visited, it is marked read and never visited again (unless explicitly asked to). But, by its very nature, information in each document is ever expanding along with the Web. Thus, soon after an update, the catalog could become out of date. The search strategies thus do not take into account the highly dynamic nature of the Web.

Besides these shortcomings, the search interface provided by these tools is highly restrictive. A typical interface would ask for keywords in the document the user is interested in and a choice of the kind of search to be done (e.g., title search, URL search, keyword search etc). Thus, the search possibilities are circumscribed by the limited choices provided by the search interface, as opposed to the possibilities opened up by ad hoc querying.

Database researchers have also investigated issues related to querying information present in documents ([17, 1, 4, 2]). Their approach to the problem is to provide a "database view" of the structured information present in documents. Most works in this direction follow the idea of "mapping" the underlying grammar of the document to an appropriate database scheme. Thus, when the document is parsed using this grammar, corresponding objects would be created in the database. These works exploit the well studied optimization techniques in databases to improve "document query processing". In Section 5, we discuss in detail some of the recent works in this direction.

Though the Web is a collection of structured documents, the nature of interrelationships among these documents raises new issues that are yet to be addressed in a convincing manner by the above proposals. Information gathering in the Web lays its emphasis on navigation via hyperlinks that relate documents to one another. The above proposals do not account for the notion of hyperlinks and the associated aspects of navigation. Restructuring the relationship among the various documents is another issue yet to be addressed satisfactorily. Also, the multimedia nature of Web documents, calls for novel ideas and techniques to address the problem of querying the Web.

Our Strategy: We adopt a pragmatic approach to the problem of querying and restructuring information in the Web. We propose a declarative query language called WebLog. Some of the highlights of WebLog are:
(a) providing a declarative interface for querying as well as restructuring (the language is logic based),
(b) accommodating partial knowledge user may have on the information being queried (WebLog has a rich syntax and semantics),
(c) providing ways for seamlessly integrating libraries of document analysis and string processing algorithms, thus putting their combined power to work for querying and restructuring the Web (the language has facilities for specifying foreign functions in the form of "built-in predicates"), and
(d) recognizing the dynamic nature of Web information (query processing need not be done on catalogs, but on the Web itself by taking advantage of "navigation landmarks" the user has the flexibility to specify).

The framework we propose is general enough to handle multimedia documents in the Web. It is inspired by SchemaLog, a simple but powerful logic language proposed by Lakshmanan, Sadri, and Subramanian [9, 10] for interoperability in multidatabase systems. Indeed, the syntax of the language is almost identical to that of SchemaLog.

The rest of the paper is organized as follows. In Section 2, we provide an overview of HTML. The conceptual model of HTML documents is presented in Section 3. Section 4 discusses the syntax and semantics of WebLog. The role of built-in predicates is discussed in Section 4.2. Section 5 discusses related research in Section 5, and conclude in Section 6.

2 HTML Overview

In this section, we present the main features of the Hypertext Markup Language (HTML), the language of most Web documents today. A detailed discussion is clearly beyond the scope of this paper. Interested readers are referred to [3].

HTML is the language used for creating hypertext documents on the Web. It has the facility for creating documents that contain hyperlinks which are pointers from keywords appearing in the document to a destination. At its simplest, the destination is another HTML document. The destination could also be a resource such as an external image, a video clip, or a sound file. Hyperlinks are the most important constituent of HTML documents. They are composed of two anchors -- a source anchor (henceforth called hyperlink) that specifies the start of the hypertext link and a destination anchor (henceforth called href) that is a pointer to the document to be linked from the source. The display that is the result of viewing a HTML document using a browser (such as NCSA Mosaic or NetScape) is called a page. The hypertext appears as a highlighted text in the page. Activation (usually clicking) of hypertext is interpreted as a request for displaying the destination document. We call this process navigation. For reasons that will become
obvious in Section 3, we associate an id called \texttt{hlink-id} with each hyperlink. A hlink-id has two components, the hypertext, and the href.

HTML allows for preparing documents for Web browsing by embedding control codes (tags) in ASCII text to designate titles, headings, paragraphs, and hyperlinks. Figure 1 contains some of the commonly used tags in a HTML document.

Conceptually, an HTML document consists of two parts: the \texttt{head} and the \texttt{body}. The \texttt{head} contains meta-information about the document. It is specified using the tag \texttt{\langle title\rangle}. The \texttt{body} consists of the document contents that includes headings, text, images, voice, video etc and hyperlinks. Users can navigate over the various documents by activating the hyperlink that would be of interest. A sample HTML document is presented in the Appendix along with its corresponding page.

3 Conceptual Model

In this section, we describe the \texttt{WebLog} model of HTML information. The conceptual model suggests simple, yet powerful ways of querying HTML documents. It also facilitates making use of common knowledge users might have about a document.

Each page (and hence a document) consists of a heterogeneous mix of information about the topic mentioned in the 'title' of the document. In practice, a typical document would consist of "groups of related information" that are spatially close together in the page. Information within each such group would be homogeneous. For instance, information enclosed within the tag \texttt{\langle b\rangle} (horizontal line) in a document could form a group of related information. Similarly, the tags header, paragraphs, lists etc could play the role of delimiting one group from another.

We would like to distinguish between groups of related information appearing in a page. We call each such group, a rel-infon. A page is thus a set of rel-infons.

The notion of what constitutes a rel-infon is highly subjective. We believe this choice should be left to the user, who will define it based on her needs. For example, in the HTML document presented in the Appendix, we could consider either the tag \texttt{\langle b\rangle} or \texttt{\langle ul\rangle} to be the rel-infon delimiter. The former case, the granularity we obtain for a rel-infon is at the level of distinguishing information present in the Ley server, and elsewhere. In the latter case, the granularity is finer – information appearing under 'conferences', 'journals' etc would be considered as corresponding to different rel-infons. We argue that this flexibility of specifying rel-infon granularity is a source of great power in expressing queries.

From the perspective of querying and restructuring HTML documents, the information that would be of utmost interest in a page are keywords or more generally strings, hyperlinks, and tags that adorn strings. We would like to provide 'first-class status' to all these concepts in our model.

A rel-infon has several attributes. The attributes come from a set consisting of strings 'occurs', 'hlink', and various tags (such as \texttt{\langle title\rangle}, \texttt{\langle b\rangle}, \texttt{\langle em\rangle}) that adorn strings in a HTML document. The attributes of a rel-infon map to 'values' that are strings, except for the hlink attribute that is mapped to a hlink-id.

Formally, let \( T \) be the set of all tags that adorn tokens in a HTML document, \( S \) be the set of strings, and \( \mathcal{H} \) be the set of hlink-ids. A rel-infon is a partial, set valued mapping \( I \):

\[
I : \{\text{occurs}, \text{hlink}\} \cup T \rightarrow 2^{\mathcal{H}}
\]

Intuitively, the attributes are meant to play the following role in modeling a rel-infon. The attribute 'occurs' is mapped to the set of strings occurring in a rel-infon. 'hlink' is mapped to the set of hlink-id's of hyperlinks appearing in a rel-infon, and the tag attributes, if defined, are mapped to the tokens they adorn in the rel-infon. For instance in the document in the Appendix, if we consider \texttt{\langle ul\rangle} to be the rel-infon delimiter, \( b \rightarrow \text{'conferences'} \) is a legal 'attribute/ value pair' in the rel-infon on conferences. The tag attribute 'title' is a special one; it is mapped to the same string (the title of the document) regardless of the rel-infon in the document.

Each rel-infon also has a unique id. This id could be the a token appearing in a header associated with a rel-infon, the most prominent 'keyword' in a rel-infon, or even the byte offset of the start of the rel-infon from the beginning of the page. Our model does not commit to a specific choice and offers some flexibility.

4 WebLog – Syntax and Semantics

This section discusses the syntax and semantics of \texttt{WebLog}. The role of built-in predicates in a \texttt{WebLog} programming environment is also discussed.

4.1 Syntax

We use strings starting with a lower case letter for constants and those starting with an upper case letter
for variables. As a special case, we use $t_i$ to denote arbitrary terms of the language. $A, B, \ldots$ denote arbitrary well-formed formulas and $A, B, \ldots$ denote arbitrary atoms.

The vocabulary of WebLog consists of pairwise disjoint countable sets $\mathcal{F}$ (of function symbols), $\mathcal{S}$ (of non-function symbols), $\mathcal{V}$ (of variables), and the usual logical connectives $\neg, \land, \lor, \exists, \forall$.

Every symbol in $\mathcal{S} \cup \mathcal{V}$ is a term of the language. If $f \in \mathcal{F}$ is a $n$-place function symbol, and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.

An atomic formula of WebLog is an expression of one of the following forms:

$$\langle \text{url} \rangle \langle \text{attr} \rangle \langle \text{val} \rangle$$

where $\langle \text{url} \rangle$, $\langle \text{attr} \rangle$, $\langle \text{rid} \rangle$, and $\langle \text{val} \rangle$ are terms of $\text{WebLog}$. We refer to them as $\text{url term}$, $\text{attr term}$, $\text{rid term}$, and $\text{val term}$ respectively. The rid term intuitively stands for the re-infor $\langle \text{rid} \rangle$ and is optional. The well-formed formulas (wff's) of WebLog are defined as usual: every atom is a wff; $\neg A, A \land B, A \lor B, (3X)A$, and $(\forall X)A$ are wff's of $\mathcal{L}$ whenever $A$ and $B$ are wff's and $X$ is a variable.

We also permit molecular formulas of the form

$$\langle \text{url} \rangle \langle \text{attr} \rangle \langle \text{val} \rangle$$

as an abbreviation of the corresponding well-formed formula $\langle \text{url} \rangle \langle \text{attr} \rangle \langle \text{val} \rangle$. In spirit, this is similar to the molecules in Ilog F-logic [7] and recently in SchemaLog [16].

Example: http://www.com[X : title→Web, hlink→L, occurs→example] is an atomic formula in WebLog. Here, the url address is the url term, title, hlink, and occurs are the attr terms, and the remaining terms are the val terms.

Next, we present the semantics of WebLog using examples. The examples make use of "built-in predicates" that are tailor-made for the Web setting. Such predicates play a significant role in a WebLog programming environment. Before we present the semantics, we discuss some of the commonly used built-in predicates.

4.2 Built-in Predicates

In any application, it is often useful (or necessary) to express certain general relationships whose semantics is well understood in the context of the application. Thus, these relationships need not be explicitly defined in the program, but are implicitly known to the system. Such relationships are expressed using special predicates called built-in predicates. Some examples of built-ins in database are the arithmetic predicates (<, >, =, etc).

In the context of querying the Web, there is a natural need for string processing and for conventional and multimedia document analysis. In real life, algorithms for such analysis are available as standalone, external functions. With an aim towards exploiting the availability of these external resources, we treat built-in predicates as (abstractions of) external functions. This novel approach (a) combines the power of search, deduction, and external functions, and (b) makes it easy to (continually) incorporate new external functions even as they are deployed. Built-ins are a source of great power for WebLog. Naturally, our treatment of built-in predicates calls for the notion of a 'legal binding pattern' – the external function's pre-condition with regard to an argument being either bound or free – with which a built-in could be invoked.

In the following, we identify some useful built-in predicates for WebLog applications. All of these built-ins could be invoked with a binding pattern of either (bound, free) or (bound, bound).

$href<$ hlink-id >, < url >): This predicate captures the relationship between a hlink-id and the destination anchor in its corresponding hyperlink. The second argument stands for the URL of the destination.

$hext$ (< hlink-id >, < string >): This is the counterpart of $href$ that captures the relationship between a hyperlink and its source anchor. The second argument stands for the hypertext that is the source anchor in the hyperlink corresponding to the < hlink-id >.

$string<$ < string >, < string >): Pattern matching is an important requirement in document querying. The Binary built-in predicate substring, is useful in this context. The first argument of substring is the source string and the second argument is a substring in the source.

$isa<$ < string >, < type >): This predicate is useful for type checking – a need often felt while querying patterns in documents. The first argument of isa is some object that is represented as a string. The second argument is its type (e.g. int, float, string, date, url, year etc.). While we do not anticipate a need for sophisticated type checking as in OO databases or in OO programming, the algorithms implementing this foreign function could use a predefined library to assign types to values. For example, the string following the keyword date: is assigned the type date.

$len<$ < string >, int): The second argument of this predicate is the length of the string in its first argument.

$newlink<$ < string >, < hlink-id >): The second argument is a unique hlink-id corresponding to the string occurring in the first argument. The hext component of this hlink-id is the string itself, while the href component is system generated. This predicate is useful for generating new hlinks in WebLog.

Besides these commonly used built-ins, we would freely make use of other useful predicates (such as synonym, homonym etc.) whose semantics would be clear from the context. In particular, built-ins that facilitate querying multimedia documents as abstractions of appropriate media processing algorithms would be extremely useful.

Programming Predicates: In the context of queries as well as view definitions, it will be convenient to have the facility for predicates which do not refer to any document, but exist only in the context of a program. We call such predicates programming predicates. We shall freely make use of programming predicates.2

2For programming predicates we use the conventional syntax (pred-name)((arg1), \ldots, (argn)).
predicates in the examples of Section 4.3.

4.3 Semantics

In this section, we informally present the semantics of WebLog. We make use of real life examples in our presentation. Through these examples we will illustrate the power of WebLog to (a) navigate hyperlinks, (b) search titles as well as the document for keywords, (c) recognize patterns appearing in documents, and (d) perform restructuring. We use the Ley server originating at the "Database Systems & Logic Programming" page (URL: http://www.informatik.uni-trier.de/~ley/db/index.html) for our illustration.

The semantics closely follows the conceptual model. The url term stands for the URL of a page, and similarly the attr term, rid term and val term stand for the concept they are named after. '+' and '-' in the syntax, enclose attribute/value pairs in the context of a single rel-infon in the document. An assertion url[ rid : attr → value] is interpreted as saying value belongs to the set of values associated with attr in the context of the rel-infon rid in the page url. We will illustrate these notions via examples.

4.3.1 Hyperlink Navigation and Searching Titles

One of the novel features of WebLog is its ability to actually navigate across HTML documents using a WebLog program. Hyperlinks can also be queried like ordinary data, and used for restructuring. The following example illustrates these ideas.

(Q1) We are interested in collecting all citations (hyperlinks) referring to HTML documents, that appear in the Database Systems & Logic Programming page. We would also like this collection to contain the title of the document the citation refers to. The following WebLog program expresses this need.

ans.html[title→all citations',hlink→L, occurs→T]  
   -- leyurl[hlink→L, href(L,U), U[title→T]].

Variable L in the first subgoal ranges over all hyperlinks in leyurl. The Built-in predicate href is used to navigate over the citations in the page at leyurl. The rule generates a new HTML document ans.html that is a collection of all citations in leyurl, annotated with the title of the cited document.

The navigation in this example is simple; there is just one level of traversal. Navigation of a more general kind is illustrated in the next example.

4.3.2 Querying Keywords in Documents

Ley server is the collection of documents originating in the leyurl. These documents have the property that (i) they can be reached by navigating links originating in the leyurl, and (ii) their URL will have the prefix http://www.informatik.uni-trier.de/~ley/. Suppose we would like to make use of this knowledge to express the query,

(Q2) Find all documents in the Ley server that have information related to 'Interoperability'.

The following WebLog program expresses this query.

ley_server_pages(http://www.informatik.uni-trier.de/~ley/db/index.html)  
   ← ley_server_pages(U) ← ley_server_pages(V),  
   V[hlink→L, href(L, U), substring(U,  
   http://www.informatik.uni-trier.de/~ley/),  
   interesting_urls(U) ← ley_server_pages(U),  
   U[occurs→T], synonym(T, 'Interoperability').

Rules (1) and (2) help identify the documents belonging to Ley server. The recursive rule (Rule (2)) essentially captures the properties (i), and (ii), known to the user. Navigation is done via recursion. Rule (3) searches for occurrences of keywords related to 'Interoperability' (captured using the predicate synonym) in the Ley server documents and returns the relevant URLs in the relation interesting_urls. The predicates ley_server_pages(U) and interesting_urls(U) are programmatic predicates; href, substring, and synonym are built-ins.

4.3.3 Querying Patterns

The following example illustrates how patterns appearing HTML documents can be conventionally queried in WebLog. It also demonstrates the handling of types in our framework.

(Q3) Suppose we know that a paper on Coral has appeared in the VLDB Journal. We do not know which year this paper appeared, but would like to find this information.

We know that a bibliography of papers on Coral can be found in a document with title 'coral', accessible from the Ley server. We of course, know that the year would appear in the bibliography entry. The following rule in WebLog helps find out the year.

ans(Y) ← ley_server_pages(U),  
   U[title→'coral', occurs→S], len(S, 40),  
   substring(S, 'VLDB Journal'),  
   substring(S, 'Y', year).

ley_server_pages is the relation containing the URL's of documents in Ley server (obtained using the program for query Q2). The body of the above rule expresses the user's knowledge that (a) the title of the bibliography document is 'coral', (b) It has some string (whose length we specify is 40) that has substrings 'VLDB Journal' and the string that stands for the year. We could also express our knowledge that the string corresponding to the year is of type year. Thus, the answer relation would contain all years that

3In order to avoid repeating this long URL, from now on we will refer to it as leyurl.

4A deductive database system from University of Wisconsin, Madison
appear in a string of length 40 that has \textit{VLDB Journal} as the substring. One of these integers must be the year in which this paper appeared.

4.3.4 Restructuring

A simple instance of restructuring using \textit{WebLog} can be seen in the program for query $Q_4$ in Section 4.3.1, where the answer is presented as an HTML document containing the appropriate citations. In this section, we illustrate via examples, the sophisticated restructuring capabilities of \textit{WebLog}.

($Q_4$) \textbf{We would like to compile the citations of CFP's of all conferences having `interoperability' as a topic of interest. We would also like to include information on the submission deadline in this compilation.}

We are aware that the CFP's can be obtained by navigating the tree of pages that has a root in the rel-infon containing the string `conference', in \textit{leagurl}. We also know that CFP's have patterns of the form `.submission...< date >.' With this knowledge, $Q_4$ can be expressed the following way.

\begin{verbatim}
traverse(L) — —
leagurl[occurs—'Conference', hlink—L].
traverse(L) — —
traverse(M), href(M, U),
U[occurs—'Conference', U[hlink—L].
cfp.html[U, title—'allcfps', hlink—L,
occurs—'submission date', D] — —
traverse(L), href(L, U),
U[occurs—'Interoperability'], U[occurs—P],
len(P, 20), substring(P, S),
synonym(S, 'submit'), substring(P, D),
isa(D, date).
\end{verbatim}

\begin{verbatim}
traverse(L) asserts the fact that the page cited via hyperlink $L$ is traversed (from a page 'descending' from the \textit{leagurl} page). Rule (1) initiates the navigation from the \textit{leagurl} page via all hyperlinks that are present in the rel-infon having the keyword `Conference'. Rule (2) is recursive, and is guaranteed to terminate soon because of the presence of the keyword `Conference' in the second subgoal. Thus, only pages having this keyword would be traversed. The last rule uses the following idea to generate the `result page'. If a page has the keyword `Interoperability' and some rel-infon in the same page has a pattern that mentions `submit' (or some synonym of that) along with a date in it, we infer it must be a CFP page that is of interest to us. Note the use of rid term $L$ in the rule. It helps in 'grouping' together the date and hlink information belonging to the same rel-infon in the reconstructed output. Note also the use of function symbol '.' (concatenation) in the head. It helps in appropriately positioning the strings in the output.

($Q_5$) Suppose we would like to restructure the newly generated \textit{cfp.html} further in such a way that all conferences having a deadline in the same week are grouped together in a page.

The following two rules help obtain the desired effect. (We assume a built-in predicate \texttt{dates2weeks} that converts a date to its corresponding week in the year.)

\begin{verbatim}
U[title—'W', hlink—L, occurs—'date', D] — —
cfp.html[L, hlink—L, occurs—S],
substring(S, D), isa(D, date),
dates2weeks(D, W), newlink(W, M),
href(M, U).
cfp.by.week.html[W, title—'byweek', occurs—
'weekNo.', hlink—M] — —
cfp.html[occurs—S], substring(S, D),
isa(D, date), dates2weeks(D, W),
ewlink(W, M).
\end{verbatim}

The first rule generates as many HTML documents as are distinct number of week numbers corresponding to the dates in \textit{cfp.html}. Thus, links to CFP's having deadline in the same week are put together in the same page. For each week number, subgoal \texttt{newlink} generates a unique hyperlink, in which the location of the CFP citations are added. The \texttt{dates2weeks} predicate generates a page that is an `interface', containing links to the new set of pages that are generated in the first rule. In this new set, the rid term \textit{W} is used to group together (as one rel-infon), the hlinks of the cfp pages having the deadline in the same week.

5 Related Work and Discussion

This section discusses some of the approaches and tools available for Web querying.

\textit{WWW} is a search tool developed at the University of Colorado by Oliver McBryan ([14]). \textit{WWW} has a resource locator that scour the Web inspecting all resources. Each HTML file found is indexed with its title string. Each URL referenced in a HTML file is indexed by the clickable hypertext associated with the URL, the name of the HTML file referring the URL, and its title. The information that is gathered by the locator is stored in four types of search databases -- (1) citation hypertext, (2) citation URL, (3) HTML titles, and (4) HTML address databases. The search interface makes use of the Unix grep program to query the catalog, and provides the option for searching each of these databases.

\textit{Lycos} ([13]), developed at the Center for Machine Translation, Carnegie Mellon University is currently the most popular Web search tool ([15]). The Lycos resource locator (called, \textit{web explorer}) searches the Web every day, building a database of all the Web pages it finds. The index of the catalog is updated every week. The explorer, written in Perl and C, provides the following information on each document to the catalog -- title, headings and sub headings, 100 most weighty words, first 20 lines, and number of words. The search engine takes a user query, performs a retrieval from the catalog, returning a list sorted according to a "match score". The engine is a C program that uses a disk-based inverted file retrieval system and a simple sum of weights to score documents.

As discussed in Section 1, these tools suffer from the drawbacks that (a) ad hoc querying allowed against the rigid interface is limited, (b) provisions for restructuring is limited, (c) querying is done on a catalog that is difficult to keep up-to-date, and (d) there is little provision for exploiting users' partial knowledge.
The problem of extracting data from files has been addressed by database researchers since the early days of the field ([17]). Recent advancements in information modeling (e.g., semantically richer models such as the object-oriented model), and query processing (e.g., relational query optimization techniques) together with the need created by "helpmates" such as the Web has stimulated fresh investigation of issues related to querying and updating structured data stored in documents ([1, 4, 2]). In [1], Abiteboul, Cluet, and Milo make use of the grammar of a document to "map" it to an appropriate object-oriented database. They introduce the notion of a structuring schema which consists of the database schema and the grammar annotated with database programs that specify how terminals and non-terminals in the grammar are mapped to the schema. When the document is parsed, for each grammar rule that is fired, an appropriate instance, dictated by the annotation, is created in the database. They adopt well-studied database optimization techniques to efficiently perform this translation. Christophides, Abiteboul, Cluet, and Scholl ([4]) use a similar idea to map SGML (Standard General Markup Language) documents into object-oriented databases. Such a mapping requires extending the object query languages (calculus, and SQL-like languages), and they study these formal extensions. Interestingly, in their model (like in ours) navigation is given a first class status using "Meta variables". Abiteboul, Cluet, and Milo ([2]) study the (inverse) problem of propagating updates specified logically on a database, to a file that actually stores the structured data. They investigate optimization techniques suitable for this "reverse translation".

Shortcomings of these proposals from a Web querying standpoint were identified in Section 1. On the other hand, we anticipate a natural use for techniques in these works in WebLog applications. We can use the techniques of [1] to "materialize" relevant information into databases - O or otherwise - which can then be queried using WebLog. We plan to investigate this issue in future.

Recent months have seen a spate of work in the area of querying the Web. W3QL ([6]) is a SQL-like query language designed specifically for the Web. This language has features for querying HTML documents and hyperlinks, but lacks restructuring facilities. HyperFile ([5]) is an interesting data and query model useful in the context of querying the Web. The MultiSurf project ([12]) is an attempt at integrating text browsing of a local database with hypertext browsing of the Web. [11] presents the Hy+ system for graphical presentation and visual querying of structured data. Web visualization is the emphasis of Hy+.

Salient features of our framework include the following. The declarative query interface of WebLog facilitates simple and natural ways of querying Web information. Its rich syntax and semantics allow for expressing powerful queries that incorporate knowledge that users might have on the information being queried. WebLog provides first class status to hyperlinks. This novelty contributes to two major advantages - (a) the user can specify partial information on the traversals to be done to answer a query, and (b) the query processor can exploit this information to query directly on the Web in an efficient way (rather than query the catalog which might be out of date and huge in size). This framework is also general enough to support multimedia information, as long as appropriate algorithms for media information processing and extracting their 'meta-data' are available for abstraction as built-in predicates.

We note that our proposal is not meant to replace the existing search facilities in the Web. We could build on the existing tools to realize our framework. In this sense, the work presented in this paper complements the tools that are available for Web searching. For 'global' searches where the user has little knowledge about the information she wants (in terms of possible locations etc.), and wants to query the whole Web blindly, we still advocate the use of search engines. On the other hand, there are numerous circumstances where the user has partial knowledge of the information required. In such cases, we anticipate the use of WebLog as an attractive alternative to search engines. Restructuring and ad hoc querying are unique advantages of WebLog.

6 Conclusions

The objective of this work has been to provide a declarative system for Web querying and restructuring. With this in mind, we have developed a simple logic called WebLog, inspired by the multidatabase interoperability language, SchemaLog ([9, 10]). We presented a conceptual model for HTML documents and studied its syntax and (informal) semantics. One of the novelties of WebLog is that it provides a first class treatment of hyperlinks in HTML documents, and makes use of this to express powerful forms of navigation across HTML documents. The examples illustrate the power of WebLog for conventional as well as novel ways of querying and restructuring Web information.

Part of the future research in this direction involves developing the formal semantics of WebLog. We are currently in the process of implementing WebLog. In this context, we are investigating various useful foreign functions and their efficient incorporation in the WebLog engine. Given the close relationship of SchemaLog to WebLog, it would be interesting to investigate if this language could be used for Web querying as well as conventional database querying, thus providing for a truly integrated interface for interoperability across disparate information sources. Our ongoing work addresses these and related issues.

References


A Appendix

In this section we present a sample HTML document at URL: http://www.informatik.uni-trier.de/ley/db/index.html. The document (Figure 2) as well as its corresponding page viewed using the browser NCSA Mosaic (Figure 3) are presented.
Database Systems & Logic Programming

An experimental
A bibliography server by
Michael Ley
University of Trier
Trier, Germany.

Call for Contributions

Information on this server

Conferences

Index: All conferences on this server

SIGMOD
VLDB

Logic Programming

SLP
NAACL

Journals

TOIS

Links to related services

Computer Science Organizations

ACM
SIGMOD
ALP

Virtual Library: Logic Programming

City University Constraints Archive

Figure 2: Sample HTML Code
Database Systems & Logic Programming

An experimental bibliography server by Michael Ley, Universität Trier, Germany

Call for Contributions

Information on this server

- Conferences
  - Index: All conferences on this server
  - Database Systems (SIGMOD, VLDB, PODS, EDBT, ICDE)
  - Logic Programming (ICLP, SLP, NACL, PLILP, POPL, ...)
- Journals (TOIS, TOPLAS, Formal Languages, Information Systems, ...
- Books: Collections - DB Textbooks
- Author Index: Trees - Forms
- Bibliographies on selected subjects (still in its infancy)
  - Systems
  - Prolog
  - Formalisms and Languages
  - Implementation of Database Systems (Access Paths, ...
  - Deductive Database Systems
  - BDDs
- Research Groups: Database Systems - Logic Programming
- News on this server

Figure 3: The page at leyurl