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 our interest, 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.*

Keywords: World Wide Web, Structured documents, HTML, Logic, Intelligent browsing, Mixed-media access, Document processing.

*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 À La Recherche of Quebec, and University of North Carolina at Greensboro.

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 a specific information she is interested about)* retrieve *and perhaps* restructure *information in the Web?* In real life, she might also have clues on the information of her interest, such as its likely location in the Web, its structure, some keywords, or patterns relevant to it. We would like to help the user specify such information as part of the query, make use of it to search the Web, and return a helpful response that would satisfy 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 remaining discussions, we use the terms Web documents and HTML documents interchangably – 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 the user specifies, search the Web and return information related to the keyword. Lycos, WWWW (World Wide Web Worm), NetFind, and InfoSeek are some of them. 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, URL[1] 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 WWWW, and Lycos. More information on Web search tools in general can be found in [SEKN92, Kos].

Currently available Web search tools suffer from the following drawbacks.

- *Partial knowledge that a user might have on the information she is querying 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

[1] Uniform Resource Locator, an address that specifies the location of a resource in the Web.

the following information on CFPs.

(i) All database related CFP's 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 ... $< date >$ ..", 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 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 the format in which the answer should be presented.

- *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.

Database researchers have also investigated issues related to querying information present in documents ([SHT+77, ACM93, CAS94, ACM95]). 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 interrelationship 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), and *(c)* 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 multi-media documents in the Web. It is inspired by SchemaLog, a simple but powerful logic language proposed by Lakshmanan, Sadri, and Subramanian [LSS93, LSS95] 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. We disucss 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 in clearly beyond the scope of this paper. Interested readers are referred to [BC95].

HTML is the language used for creating hypertext documents on the Web. It has the facility for creating documents that contain *hyperlinks* that 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 *hypertext*) 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

<title> ... </title>	Document Title
....	Most prominent header
.	
...	Least prominent header
<hr>	Horizontal line
<pre> ... </pre>	Preformattd text
 ... 	Emphasis
 ... 	bold font
...	Unordered List
....	Ordered list
<htext>	hyperlink to "url"
<img src="url"[alt=][align=]	link to image file

Figure 1: Common Tags in HTML

appears as a highlighted text in the page, activation (usally clicking) of which is interpreted as a request for the destination document. We call this process *navigation*. For reasons that will become obvious in Section 3, we associate an id called *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 *head* and the *body*. The head contains meta-information about the document. It is specified using the tag ‘< *title* >’. The 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 correponding page.

3 Conceptual Model

In this section, we describe the *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 <HR> (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 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 <HR> or <UL> to be the rel-infon delimiter. In 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.

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 `<title>, <b>, <em>`) 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 $\mathcal{T}$ be the set of all tags that adorn tokens in a HTML document, $\mathcal{S}$ be the set of strings, and $\mathcal{H}$ be the set of hlink-ids. A rel-infon is a partial, set valued mapping I,

$$I \ : \ \{`occurs', `hlink'\} \ \cup \ \mathcal{T} \ \rightarrow \ 2^{\mathcal{S} \cup \mathcal{H}}$$

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 occuring 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 <ul> to be the rel-infon delimiter, $b \rightarrow `conferences'$ 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) in all rel-infons 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.

4 *WebLog* – Syntax and Semantics

This section discusses the syntax and semantics of *WebLog*. The role of built-in predicates in a *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. $\mathcal{A}, \mathcal{B}$, ... denote arbitrary well-formed formulas and A, B, ... denote arbitrary atoms.

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

Every symbol in $\mathcal{S} \cup \mathcal{V}$ is a term of the language. If $f \in \mathcal{G}$ 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:

$\langle\mathrm{url}\rangle[\langle\mathrm{rid}\rangle : \langle\mathrm{attr}\rangle \twoheadrightarrow \langle\mathrm{val}\rangle]$
$\langle\mathrm{url}\rangle[\langle\mathrm{rid}\rangle : \langle\mathrm{attr}\rangle \rightarrow \langle\mathrm{val}\rangle]$
$\langle\mathrm{url}\rangle[\langle\mathrm{attr}\rangle]$
$\langle\mathrm{url}\rangle$

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

We also permit *molecular formulas* of the form
$\langle\mathrm{url}\rangle[\langle\mathrm{rid}\rangle : \langle\mathrm{attr}_1\rangle \twoheadrightarrow \langle\mathrm{val}_1\rangle, \ldots, \langle\mathrm{attr}_n\rangle \twoheadrightarrow \langle\mathrm{val}_n\rangle]$
as an abbreviation of the corresponding well-formed formula
$\langle\mathrm{url}\rangle[\langle\mathrm{rid}\rangle : \langle\mathrm{attr}_1\rangle \twoheadrightarrow \langle\mathrm{val}_1\rangle] \wedge \cdots \wedge \langle\mathrm{url}\rangle[\langle\mathrm{rid}\rangle : \langle\mathrm{attr}_n\rangle \twoheadrightarrow \langle\mathrm{val}_n\rangle]$.
In spirit, this is similar to the molecules in F-logic [KLW94], and more recently in SchemaLog [LSS95].

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. Built-ins 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 explicity defined in the program, but are implicitly known by the system. Such relationships are expressed using special predicates called *built-in predicates*. Some examples of built-ins in datalog are, the arithmetic predicates ($<, >, =$ etc). In the following, we discuss some of the frequently used built-in predicates in *WebLog*.

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.

htext($<$ *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* $>$.

substring($<$ *string* $>, <$ *string* $>$)*:* Pattern matching is an important need in our setting. 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.).

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 occuring in the first argument, the second string denotes a unique hlink-id. The htext component of this hlink-id is the string itself, while the href component is system generated. This predicate is useful for generating new hlink-ids 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 could be easily defined.

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.**[2] Technically, programming predicates can be easily incorporated in *WebLog* by introducing a separate sort of predicate symbols and then interpreting them "classically". We shall freely make use of programming 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`[3]) 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 $url[rid : attr \twoheadrightarrow value]$ 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 that it treats hyperlinks as 'first class citizens'. This provides the facility for navigating 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.

(Q_1) *We are interested in collecting all citations (hyperlinks) refering 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.

$$ans.html[title \rightarrow \text{`}all\ citations\text{'},\ hlink \twoheadrightarrow L,\ occurs \twoheadrightarrow T] \longleftarrow leyurl[hlink \twoheadrightarrow L],\ href(L, U),\ U[title \rightarrow T]$$

Variable L in the first subgoal ranges over all hyperlinks in $leyurl$. The Built-in predicate *href* is used to

[2] For programming predicates we use the conventional syntax $\langle pred\text{-}name\rangle(\langle arg_1\rangle, \ldots, \langle arg_n\rangle)$. Note that this introduces ambiguity in the syntax of *WebLog*, as a programming predicate could now be confused with a functional term! We can remove this ambiguity by requiring functional terms to conform to the syntax $f < t_1, \ldots, t_m >$. For the sake of clarity and simplicity of exposition, we ignore this point. The intended meaning of *WebLog* expressions will always be clear from the context.

[3] Inorder to avoid repeating this long URL, from now on we will refer to it as *leyurl*.

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,

(Q_2) *Find all documents in the Ley server that have information related to 'Interoperability'.*

The following *WebLog* program expresses this query.

$$ley\_server\_pages(http://www.informatik.uni\text{-}trier.de/~ley/db/index.html)$$
$$ley\_server\_pages(U) \longleftarrow ley\_server\_pages(V),\ V[hlink \twoheadrightarrow L],\ href(L, U),$$
$$substring(U, http://www.informatik.uni\text{-}trier.de/~ley/)$$
$$interesting\_urls(U) \longleftarrow ley\_server\_pages(U),\ U[occurs \twoheadrightarrow I],\ synonym(I, \text{'}Interoperability\text{'})$$

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 occurrances of keywords related to 'Interoperability' (captured using the predicate *synonym*) in the Ley server documents and returns the relevant URLs in the relation *interesting_url*.

4.3.3 Querying Patterns

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

(Q_3) *Suppose we know that a paper on Coral*[4] *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

[4] A deductive database system from University of Wisconsin, Madison

following rule in *WebLog* helps find out the year.

$$ans(Y) \longleftarrow ley\_server\_pages(U), U[title \rightarrow \text{`}coral', occurs \twoheadrightarrow S], len(S, 40),$$
$$substring(S, \text{`}VLDBJournal'), substring(S, Y), isa(Y, year).$$

ley_server_pages is the relation containing the URL's of documents in Ley server (obtained using program for query Q_2). 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 year is of type *year*. Thus, the answer relation would contain all integers that appear in a string of length 40 that has *VLDB Journal* as the substring. One of these integers must be the year in which this paper appeared.

4.3.4 Restructuring

A primitive instance of restructuring using *WebLog* can be seen in the program for query Q_1 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 *WebLog*.

(Q_4) *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 *leyurl*. We also know that CFP's have patterns of the form '..submission...< *date* >..'. With this knowledge, Q_4 can be expressed the following way.

$$traverse(L) \longleftarrow leyurl[occurs \rightarrow \text{`}Conference',\ hlink \twoheadrightarrow L]$$
$$traverse(L) \longleftarrow traverse(M),\ href(M, U),\ U[occurs \rightarrow \text{`}Conference'],\ U[hlink \twoheadrightarrow L]$$
$$cfp.html[title \rightarrow \text{`}allcfps',\ hlink \twoheadrightarrow L,\ occurs \rightarrow \text{`}submndate',\ occurs \twoheadrightarrow D] \longleftarrow$$
$$traverse(L),\ href(L, U),\ U[occurs \rightarrow \text{`}Interoperability'],\ U[occurs \twoheadrightarrow P],$$
$$len(P, 20),\ substring(P, S),\ synonym(S, \text{`}submit'),\ substring(P, D),\ isa(D, date).$$

traverse(L) asserts the fact that the page cited via hyperlink L is traversed (from a page 'descending' from the *leyurl* page). Rule (1) initiates the navigation from the *leyurl* 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.

(Q_5) *Suppose we would like to restructure the newly generated 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 programming predicate *dates2weeks* that converts a date to its corresponding week in the year.)

$$\begin{aligned}
&U[title \twoheadrightarrow W, hlink \twoheadrightarrow L,\ occurs \rightarrow \text{`}date\text{'},\ occurs \twoheadrightarrow D] \longleftarrow \\
&\qquad cfp.html[hlink \twoheadrightarrow L,\ occurs \twoheadrightarrow S],\ substring(S, D),\ isa(D, date), \\
&\qquad dates2weeks(D, W),\ newlink(W, M),\ href(M, U) \\
&cfp_by_week.html[title \rightarrow \text{`}byweek\text{'},\ occurs \rightarrow \text{`}weekNo.\text{'},\ hlink \twoheadrightarrow M] \longleftarrow \\
&\qquad cfp.html[occurs \twoheadrightarrow S],\ substring(S, D),\ isa(D, date), \\
&\qquad dates2weeks(D, W),\ newlink(W, M)
\end{aligned}$$

The first rule generates as many HTML documents as there are distinct number of week numbers corresponding to the dates in 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 newlink generates a unique hyperlink, in whose location the CFP citations are added. The second rule is used to generate a page that is an 'interface', containing links to the new set of pages that are generated in the first rule.

5 Related Work and Discussion

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

WWWW is a search tool developed at the University of Colorado by Oliver McBryan ([McB94]). WWWW has a resource locator that scours 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 *egrep* program to query the catalog, and provides the option for searching each of these databases.

Lycos ([ML94]), developed at the Center for Machine Translation, Carnegie Mellon University is currently the most popular Web search tool ([Poi95]). The Lycos resource locator (called, *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.

The problem of extracting data from files has been addressed by database researchers since the early days of the field ([SHT+77]). 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 developments such as the Web has stimulated fresh investigation of issues related to querying and updating structured data stored in documents ([ACM93, CAS94, ACM95]). In [ACM93], 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 ([CAS94]) 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. More recently, Abiteboul, Cluet, and Milo ([ACM95]) 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 are also identified in Section 1.

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 allows 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). This framework is also general enough to support multimedia information.

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.

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 [LSS93, LSS95]. We presented a conceptual model for HTML documents and studied its syntax and 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 Web information.

Part of the future research in this direction involves studying the formal semantics of *WebLog*. We are currently investigating efficient implementation strategies for realizing *WebLog*. 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. Our ongoing work addresses these and related issues.

References

[ACM93] Abiteboul, S., Cluet, S., and Milo, T. Querying and updating the file. In *Proc. of the Conf on Very Large Databases (VLDB)*, 1993.

[ACM95] Abiteboul, S., Cluet, S., and Milo, T. A database interface for file update. In *ACM SIGMOD International Conference on Management of Data*, 1995.

[BC95] Berners-Lee, T. and Connolly, D. Hypertext markup language – 2.0 (work in progress), 1995. URL:http://www.w3.org/hypertext/WWW/MarkUp/MarkUp.html.

[CAS94] Christophides, V., Cluet, S. Abiteboul, S., and Scholl, M. From structured documents to novel query facilities. In *ACM SIGMOD International Conference on Management of Data*, 1994.

[KLW94] Kifer, M., Lausen, G, and Wu, J. Logical foundations for object-oriented and frame-based languages. *Journal of ACM*, 1994. (Tech. Rep., SUNY Stony Brook, 1990).

[Kos] Koster, M. World wide web wanderers, spiders, and robots. URL: http://web.nexor.co.uk/mak/doc /robots/robots.html.

[LSS93] Lakshmanan, L.V.S., Sadri, F., and Subramanian, I. N. On the logical foundations of schema integration and evolution in heterogeneous database systems. In *Proc. 3rd International Conference on Deductive and Object-Oriented Databases (DOOD '93)*. Springer-Verlag, LNCS-760, December 1993.

[LSS95] Lakshmanan, L.V.S., Sadri, F., and Subramanian, I. N. A logical language for interoperability in multi-database systems. Technical report, Concordia University, Montreal, March 1995. Submitted for publication, March 1995. (A preliminary version appeared in International Conference on Deductive and Object Oriented Databases, December 1993.).

[McB94] McBryan, O.A. Genvl and wwww: Tools for taming the web. In *Proc. of the First Intl. WWW Conf.*, May 1994.

[ML94] Mauldin, M.L. and Leavitt, J.R. Web agent related research at the center for machine translation, August 1994. SIGNIDR Meeting, McLean, Virginia.

[Poi95] Point Communications Corp. Lycos: the catalog of the internet, 1995. URL: http://www.pointcom.com/jpegs/reviews/6-28-031.htm.

[SEKN92] Schwartz, M.F., Emtage, A., Kahle, B., and Neuman, B. C. A comparison of internet resource discovery approaches. *Computing Systems*, 5(4), 1992. URL: ftp://ftp.cs.colorado.edu/pub/cs/techreports /schwartz/PostScript/RD.Comparison.ps.Z.

[SHT+77] Shu, N. C., Housel, B.C., Taylor, R.W., Ghosh, S.P., and Lum, V.Y. Express: a data extraction, processing, and restructuring system. *ACM Transactions on Database Systems*, 2, 2, June 1977.

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 as well as its corresponding page viewed using the browser NCSA Mosaic are presented.

```
<html><head><title>Database Systems &amp; Logic Programming</title></head>
<body><h1>Database Systems &amp; Logic Programming</h1>
An experimental
<a href="intro.html">bibliograpy server</a> by
<a href="http://www.informatik.uni-trier.de/~ley/addr.html">Michael Ley</a>,
Universit&auml;t
<a href="http://www.uni-trier.de/trier/trier_eng.html">Trier, Germany</a>.<br>
<b><a href="about/call.html">Call for Contributions</a></b>

<hr>

<h2>Information on this server</h2>
<ul>
<li><b>Conferences</b>
<ul>
<li><a href="conf/index.a.html">Index: All conferences on this server</a>
<li>... <a href="conf/index.html">on Database Systems</a>
(<a href="conf/sigmod/index.html">SIGMOD</a>,
<a href="conf/vldb/index.html">VLDB</a>,
...)
<li>... <a href="conf/index.l.html">on Logic Programming</a>
(<a href="conf/iclp/index.html">ICLP</a>,
<a href="conf/slp/index.html">SLP/NACLP</a>,
...)
</ul>
<li><a href="journals/index.html"><b>Journals</b></a>
(<a href="journals/tods/index.html">TODS</a>,
<a href="journals/tois/index.html">TOIS</a>,
...)
</ul>

<hr>

<h2>Links to related services</h2>
<ul>
<li><a href="../organizations.html">Computer Science Organizations</a>
(<a href="http://info.acm.org/">ACM</a> -
<a href="http://bunny.cs.uiuc.edu/README.html">SIGMOD</a> -
<a href="http://info.sigir.acm.org/sigir/">SIGIR</a> -
<a href="http://www.cs.mu.oz.au/~ad/alp/info-alp.html">ALP</a> -
etc.)
<li><a href="http://www.comlab.ox.ac.uk/archive/logic-prog.html">WWW
Virtual Library: Logic Programming</a> (by Jonathan Bowen, Oxford)
<li><a href="http://web.cs.city.ac.uk/archive/constraints/constraints.html">City
University Constraints Archive</a>
</ul>

<hr>
<address>
<a href="http://www.informatik.uni-trier.de/~ley/addr.html">Michael Ley</a>
(ley@uni-trier.de)
19-Jul-95
</address>
</body>
</html>
```

Figure 2: Sample HTML Code

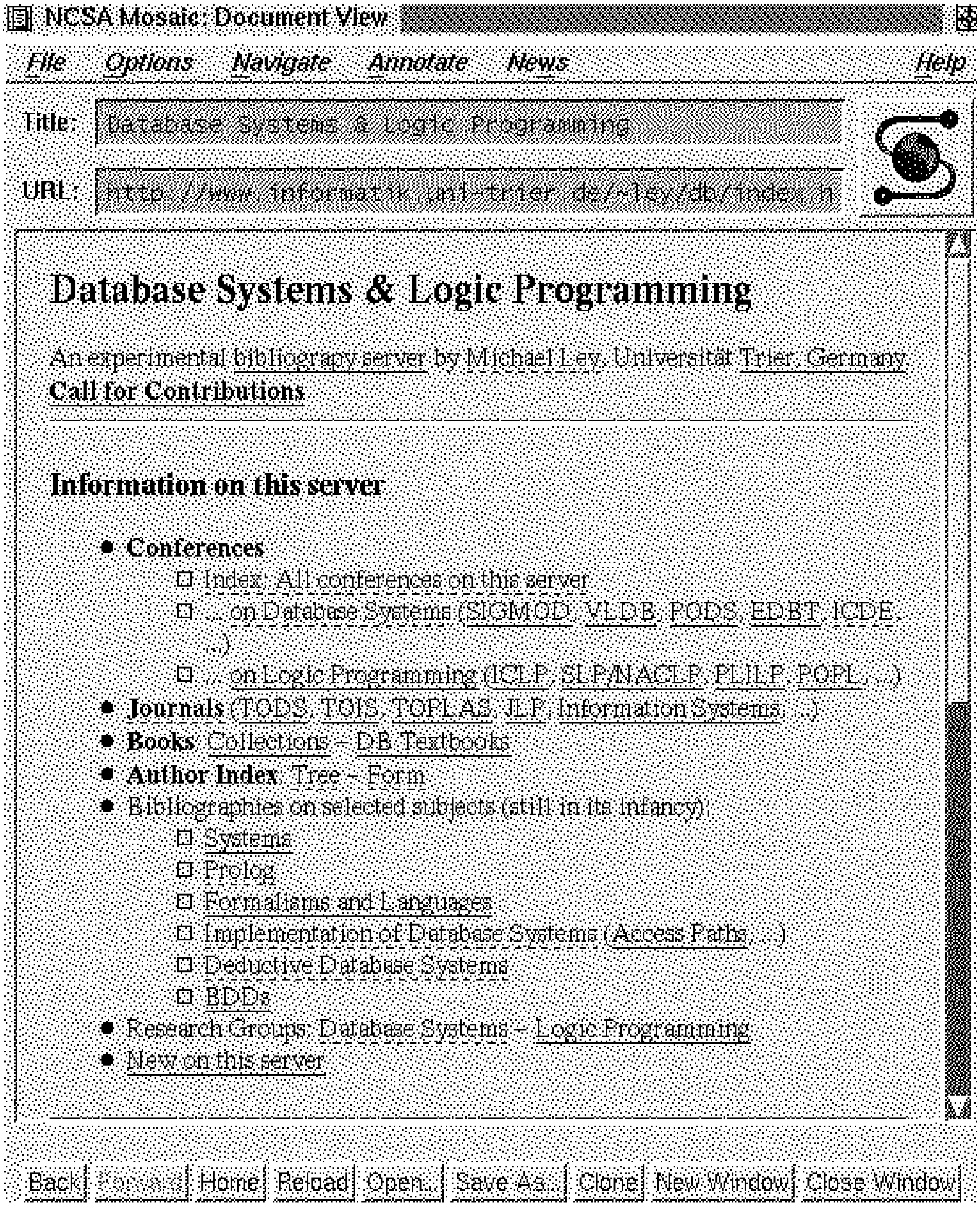

Figure 3: The page at *leyurl*