Web Languages and An Example: WebL

Quanfu Fan and Xiangtao Wang

April 30, 2003

1 Introduction

The World Wide Web has been growing into the largest information resource for human being since it emerged. Given the bewildering information on the web, the task of finding exactly what a user wants is rather tedious and even daunting. While a large number of search engines such as Google and Yahoo are terribly useful, users have to be patient to click enough links to find the data suitable for their tasks. Even more, a lot of times they expect to tailor and reorganize those data returned from different pages. For example, a routine thing that a lot of job seekers do every day is to visit various job web sites such as Monster and CareerBuilder to search jobs that match their skills. They would be more than happy if a program could help them retrieve and classify jobs from the web sites according to their interests. Unfortunately, despite the fact that there is a diversity of languages available for web programming, few of them are suitable for this task. In fact, there exist many such web-related tasks that are not able to automated, but have to be manipulated manually like routinely updating stuff on a website and comparing dozens of products from an on-line catalogue, etc. Consequently, there is a great call on a new language to automate web tasks and relieve users overloaded by the web.

Luca Cardelli has described this new language as 'web language'[6], which we think is literally misleading. As there is not a good name for it so far, we will borrow this term in this paper. Generally speaking, a web language is a script programming language that automates computation of documents on the web. Just like PERL easing people from text processing, web languages are designed to facilitate web-related tasks such as retrieval from web pages, computation on these data and modification of page contents as well as page publishing. Rather than providing powerful searching functionality as general search engines do, web languages are dedicated to performing computation conveniently and flexibly on the extracted data from one or several web documents. Some applications that have constructed with a web language are listed below[4],

- Customizable Web crawlers.
- Meta-search engines for popular search engines on the Web.
- Meta-newspapers that collect articles from several sites according to interests.
- Tools to build a newspaper from CDF-based descriptions [CDF97].
- Shopping robots that shop for the cheapest books at several electronic bookstores.
- Tools to extract financial information from stock pages.
- Tools to extract and compute project information from DIGITAL's intranet.
- Tools to concatenate Latex2HTML-generated documents for printing.
- Tools to validate links in Web pages.

In this paper, section 2 will examine what programming models and constructs are needed for a web language. Section 3 will study a real web language, WebL. Section 4 gives a simple application we write with WebL. Finally, a summary is given in section 5.

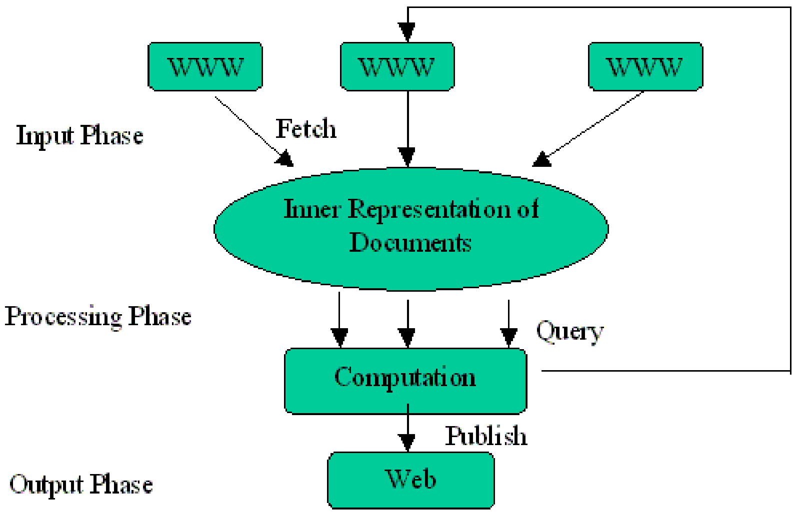

Figure 1: A Programming Model for Web Computation

2 Programming Models and Basic Constructs for Web Computation

There would be no argument that a web language should include some common language features such as types, functions and conditionals. Moreover, in order to make a more powerful script language suitable for web programming, some additional features are obviously desired to deal with the peculiarities of the web. As observed by Lucas Cardelli[6], some typical peculiarities presented by the web include wide-area distribution, lack of referential integrity and unreliable quality of service. These new observables have to be taken into account in the new programming model designed for a web language.

Basically, a typical web computation includes three major parts, input phase, processing phase and output phase, as depicted in Fig 1(Modified from figure 1 in [4]). One or several web pages are fetched back from the web and their contents are parsed into an inner data structure that supports a series of future computation. During the process of computation, some more pages would be fetched if needed. Finally, the computed results will be stored back to a web server for publishing.

Next, we will look closely at each phase and investigate what programming constructs are necessary for a web language.

2.1 Input Phase

The input phase should know how to fetch back various documents from the web for future processing. The major goal of this phase is to provide reliable web services as much as possible, disregard of the unpredictable and capricious web-computing environment.

First of all, a web language needs to understand all kinds of web protocols such as HTTP, S-HTTP and FTP etc. The language should decide which protocol it speaks to when given a URL. In fact, web protocols are quite simple, but the hardwares (network etc.) and softwares (web server etc.) those protocols interact with are likely to vary unpredictably[6], making the model of computation of the web complicated. For example, a page fetch could fail easily and frequently due to various reasons such as a high load on the server, redirection of a page or intolerably heavy network traffic. As a matter of fact, web's geographical distribution, network latency and bandwidth etc. become more important issues in a web language than CPU and memory, which are basic limiting factors for other languages. The programming models for web languages should be able to provide a special exceptional-handling mechanism to respond well to different modes of failure occurring on the web.

In addition, the language needs to handle correctly the two different ways that a URL carries request parameters, GET and POST. The parameters requested by a GET explicitly follow a URL and are easily identified. For those by a POST, however, a programmer has to deduct them from the web form where the request originates from. What's worse, in some cases, parameters are passed by using hidden fields. This has brought a challenging problem to our programming model, i.e. could that possible to identify without much manual efforts the parameters in a page, the URL of which is from computation? How this problem is addressed is crucial to the success of a web language as failing to find a robust way to reach resources would be likely to force a user to give up eventually.

2.2 Processing Phase

The processing phase of a web computation involves extracting data from web pages and performing computation on those data. How to represent page contents is one of the most fundamental issues to be addressed in this phase. As text document (either HTML or XML), a web page can have a lot of representation. However, a good data structure, which allows a quick walk through the whole document, will greatly benefit the subsequent queries and computation. For example, XML DOM, a tree view of an XML document[2], appears be a good candidate as information can be conveniently retrieved and manipulated by navigating the tree.

Once an inner representation of a web page is built, an extraction rule will be set up for retrieve information. Even though the web is often regarded as a 'distributed' database, it doesn't have any programming language to support web information queries as a relational databases does. An extraction rule expresses a user's interest to some pieces of information, guides a navigation to those information in the inner data structure and specifies how to put them together. A good example is Xquery, a language for querying XML documents[3]. Unfortunately, even with perfect extraction rules, we may still end up with an unsuccessful search due to the poor structure of web pages. For example, tables are often used not to reflect the document structure, but to enforce presenting effects.

Web computation is defined as some operations on the extracted data such as joining data with the same attribute, filtering data or modifying data. To facilitate programming, a web language can offer a mechanism to map the extracted information into some user-defined data structures[7]. Thus, programmers can concentrate more on how to control the computation than on how the information is stored in the inner data structure. For example, a job on a web site can be expressed as (like a Java class),

```
Job {
        String job_title;
        String job_category;
        String job_description;
        String position;
        int salary;
        Date postedDate;
        ....
}
```

With such a structure, programming with a web language becomes no difference with programming with other languages.

2.3 Output phase

The output phase is responsible for returning the computed data to users as desired. One preferable way is to store the processed data back to the web, i.e. generating an HTML document or even publishing it via a web server. Another more appealing feature is that the language can communicate with a web server directly, exporting the computed data to the server for dynamically creating new pages as CGI or JSP does. In this sense, what a web language does sounds more like programming the web as a database.

3 WebL

WebL is a scripting language designed to facilitate retrieving and manipulating information from web.[5] As a high level language, WebL provides traditional imperative programming language features, such as prototype-objects, dynamic typing, sets and lists, modules, etc.. To well suit for web service characteristics and web information structures, WebL computation model has two innovate concepts, namely service combinators and markup algebra.

3.1 Service Combinators

World Wide Web is a global distributed database. An important difference from normal information source is that web service is not always reliable. The web page is possibly unavailable, unexpectedly terminated, dropping to unacceptable slow speed, or mirrored geographically. The unpredictable nature of the web requires different exception handling mechanism. Service combinator is WebL's solution to get more reliable web access to the unreliable web structures. It adds certain robustness to web programming and computation.

Services, which mean any computations on remote web servers in this content, either succeed or fail. The basic service for web programming is to fetch a page of given web address, which can be illustrated by the following two built-in functions:

GetURL(url, [. parameter1=value1, parameter2=value2, ...])
PostURL(url, [.parameter1=value1, parameter2=value2, ...])

GetURL and PostURL are two functions to fetch pages with HTTP GET and HTTP Post protocol respectively. Parameters like page url and query arguments pass to the function, and a page object returns or the functions fails. The following expression searches "WebL" from google search engine:

var page = GetURL("http://www.google.com/search", [. q="WebL" .]);

Five service combinatiors are provided by WebL to make web computation more reliable: sequential execution, concurrent execution, time-out, repetition and non-termination.

1. Sequential execution
   The combinator "?" prevents service failure by adding an alternative service in case of the first service fails. The following expression first attempts to search "WebL" from google, and attempts to search from altavista in case of failure of google:
   var page = GetURL("http://www.google.com/search", [. q="WebL" .]) ?
   GetURL("http://www.altavista.com/web/results", [. q="WebL" .]);

2. Concurrent execution
   The combinator "|" reduces service failure by executing two services concurrently, returning the result of first successful service or failure if both services fail. The following expression attempts to search "WebL". Both google and altavista start concurrently, and result returns from first successful one.
   var page = GetURL("http://www.google.com/search", [. q="WebL" .]) |
   GetURL("http://www.altavista.com/web/results", [. q="WebL" .]);

3. Time-out
   The timeout combinator sets a time limit on a service. The service either succeeds with the time limit, or fails. The following expression attempts to search "WebL" from either google or altavista, but sets 10000 milliseconds as timeout limit.
   Timeout(10000, GetURL("http://www.google.com/search", [. q="WebL" .]) |
   GetURL("http://www.altavista.com/web/results", [. q="WebL" .]));

4. Repetition
   The repetition combinator allows service to be repeatedly invoked until succeeds. The following expression repeatedly attempts searching "WebL" by alternating from google and altervista, until reaches

timeout 10 seconds.

*Timeout(10000, Retry(GetURL("http://www.google.com/search", [. q="WebL".]) |
GetURL("http://www.altavista.com/web/results", [. q="WebL" .])));*

5. Non-termination
   The stall combinator never returns. It is typically used together with timeout combinatory to wait for certain time. The following expression repeatedly tries to fetch the URL, but waits 10 seconds between attempts:

   *var page = Retry(GetURL("http://www.google.com/search", [. q="WebL" .]) ?
   Timeout(10000, Stall()));*

3.2 Markup Algebra

Web documents are composed in HTML or XML format. To extract and process web documents, WebL uses a mechanism called markup algebra to make use of web document structures. WebL markup algebra understands nested element structures of web document, which makes element extraction fairly easy and high-level. In addition, WebL also provides search capability according to string pattern. WebL's mixture view web document processing makes WebL more suitable for web programming than any plain text processing language like Perl, or XML specific tool like DOM.

WebL markup algebra is based on the concept of piece and piece set. A piece is a region of page, which has start position and end position, including all tags and text segments in this region. For example, $<tr><td>text1</td><td>text2></td></tr>$ can be a piece, $<td>text1</td>$ can also be a piece. A collection of pieces is called piece set. Piece set has characteristics of both set and list: no duplication but ordered.

3.2.1 Search functions

WebL provides several functions to facilitate information retrieving from web pages. You can do element search, pattern search, PCData search, sequence search and paragraph search etc.. Among these built-in search functions, element search and pattern search are two basic search methods.

Element search utilizes the HTML or XML structure of web page. It searches through the given search scope to find all pieces matching begin and end tag pair of stated element. For example, the following expression returns all hyperlinks as a piece set of given page P:

var links = Elem(P, "a");

Pattern search treats web page as if it is plain text documents ignoring all tags, just as what we see from browser. It returns a piece set of given character pattern or regular expression groups, similar to what Perl can do. We can use the following expression to get all occurrences of string "WebL":

var weblSets = Pat(P, "WebL");

3.2.2 Piece set operators

WebL piece sets have both property of sets and lists. Like normal sets, piece sets have normal set union, set difference and set intersection operations.

Pieces in piece sets are ordered, which provides positional set operators for piece sets. Pieces can be referred by its position in piece set, like p[0] as first piece in piece set p. Pieces can also be extracted using before, after, directlybefore, directlyafter operations, which provide high flexibility to programmers to manipulate the markup contents. Let's look at the following table:

$<table>$
$<tr><td><a>A</td>$
$<td><a>B</td>$
$</tr>$
$<tr><td><a>C</td>$
$<td><a>D</td>$

$< /tr >$

$< /table >$

We can compute the retrieved web content as following:

// get piece $< a > C < /a >$

Elem(T, "a") directlyafter Elem(T, " tr")[0]

Web contents in HTML or XML format have nested element tree structure. To facilitate information computation according to the hierarchical relationship, WebL has some hierarchical operators, such as inside, contain, directlyinside and directlycontain. For example, the following expression extracts all rows containing keyword "WebL":

var Elem(Elem(P, "table")[2], "tr") contain Pat(P, "WebL");

3.2.3 Page modification

The purpose of web programming language is to make use of information from web. It is natural to have the output in web format again. WebL has this capability by providing page modification functions as part of markup algebra.

WebL creates a new page through NewPage function, which returns a template page with tags. Then pieces are inserted into specified position of a page using function InsertAfter and InsertBefore. Function Delete and Replace are used to delete a piece from a page, or replace a piece from a page.

4 An Example

WebL is very powerful in terms of automating web tasks. To illustrate the efficiency of how WebL retrieves and computes web information, let us think about the scenario of job searching. In todays job market, it is essential to get job information as early as possible. A good practice is to frequently search related web sites, such as Monster, CareerBuilder, Flipdog, and even Microsoft, IBM. The task needs to be automated if you decide to look at 100 related job websites every day.

The example WebL program chose Monster and Flipdog as two sample websites to automate job searching task. First the program gets a page object with certain query standards, then put all information of posted jobs into an object set using element search, finally new page is created to show all job information from both Monster and Flipdog in modified structures. For simplicity, this job search tool only searches job in Phoenix, Arizona area, and only accepts one argument as searching keyword.

WebL was developed in Java, and the WebL classes and resources are bundled in a Java JAR file called WebL.jar. The simple job search WebL program can be run as the following:

Java -jar WebL.jar JobSearch.webl $< keyword >$

Where keywords can be PHP, Java, Oracle, Accountant, etc..

5 Conclusion

We have shown a programming model suitable for web computation and discussed some basic constructs needed for a web language. Also, we have studies a real web language, WebL, which is featured by two novel programming constructs, service combinators and algebra marks. Some hands-on experience has been gained by implementing a simple job-seeking tool with WebL. We feel that there is an urgent need for web languages to free users from the heavy and tedious web-related tasks and WebL has given a good try in this direction. In the meanwhile, we have realized that automating web-related tasks is still quite challenging for computer scientists. Through programming WebL, we found some difficulties to process web documents:

- Many HTML web pages are poorly structured. Programmer needs to look at the HTML source code very carefully to magically figure out how to retrieve information. Web page structure may be different when different keywords are used for searching, and web page structure may be changed after couple of days. To understand HTML code and modify application according to web page structure change is time-consuming.

- Many web pages encode the search keywords, which prevents programming uniformly. For example, monster.com uses 356 as code for location Phoenix. This kind of encoding makes it almost impossible to have location as an argument to the program.
- HTML Post protocol has implicit parameter passing, which makes it more difficult to program the web information retrieving.

Some of them seem unconquerable unless new standards are applied to web documents like using XML to separate presentation and content.

References

[1] World Wide Web Consortium, Extensible Markup Language (XML), http://www.w3.org/XML/

[2] World Wide Web Consortium, The Document Object Model (DOM), http://www.w3.org/DOM/

[3] World Wide Web Consortium, XQuery 1.0: An XML Query Language, http://www.w3.org/TR/xquery/

[4] Thomas Kistler and Hannes Marais WebL - A programming language for the Web. In Computer Networks and ISDN Systems (Proceedings of the WWW7 Conference), Volume 30, pages 259-270. Elsevier, April 1998

[5] WebL On-line Manual, http://research.compaq.com/SRC/WebL/htmldocu/WebL.html

[6] L. Cardelli and R. Davies, Service combinators for Web computing, Research Report 148, Digital Equipment Corporation Systems Research Center, Palo Alto, California, June 1999

[7] Arnaud Sahuguet and Fabien Azavant, Building Intelligent Web Applicationa Using Lightweight Wrappers, Data and Knowledge Engineering, 2000

A A Simple Job Search Tool Written with WebL

```
// JobSearch.webl queries Monster.com and Flipdog.com to search jobs in Phoenix, Arizona for a given ke
// usage: java -jar WebL.jar JobSearch.webl <keyword>

import Str, Browser;

var QueryMonster = fun(query)
	var P = GetURL("http://jobsearch.monster.com/jobsearch.asp",[. q = query, lid = 356 .]);

	var jobTable = Select ( Elem(P, "table"),
				fun(a) Text(Elem(Elem(a, "tr")[0], "td")[0])  == "Date" end
			);

	var all = Elem(jobTable[0], "tr");

	var rows = all - all[0];

	var results = [];

	every r in rows do
		var cc = Children(Elem(r, "td")[2])[0];
		results = results + [
			[.
				dd = Text(Elem(r, "td")[0]),
				title = Text(Children(Elem(r, "td")[2])[0]),
				ref = cc.href ? "",
				com = Text(Elem(r, "td")[3]),
				loc = Text(Elem(r, "td")[4])
			.]
		]
	end;
	results
end;

var QueryFlipdog = fun(query)
	var P = GetURL("http://www.flipdog.com/js/jobsearch-results.html",
			[. loc = "AZ-Phoenix Area", srch = query .]);

	var jobTable = Elem(P, "table") contain Pat(P, "Web Jobs:");

	var rows =  Elem(jobTable[0], "tr") contain Pat(P, "at");

	var results = [];

	every r in rows do
		var cc = Elem(Elem(r, "td")[2], "a")[0];
		results = results + [
			[.
				dd = Str_Trim(Text(Elem(r, "td")[3])),
				title = Str_Trim(Text(Elem(Elem(r, "td")[2], "a")[0])),
				ref = cc.href ? "",
```

```
                                com = Str_Trim(Text(Elem(Elem(r, "td")[2], "a")[1]) ? ""),
                                loc = Str_Trim(Text(Elem(r, "td")[4]))
                        .]
                ]
        end;
        results
end;



var Anchor = fun(text, url)
        "<a target = blank href='" + url + "'>" + text + "</a>"
end;

var displayMonster = fun(R, jobs)
        var statRow = NewHTMLPiece("<H4>" + ToString(Size(jobs)) + " jobs found from Monster! </H4>");
        InsertBefore(BeginTag(Elem(R, "table")[0]), statRow);

        every job in jobs do
                var pce = NewHTMLPiece("<tr><td>" + Anchor(job.title, job.ref) +"</td><td>" +
                                        job.com  + "</td><td>" + job.loc + "</td><td>" +
                                        job.dd +  " </td><td>" +
                                        Anchor("Monster", "http://www.monster.com") + "</td></tr>");
                InsertBefore(EndTag(Elem(R, "table")[0]), pce);
        end;

        R
end;

var displayFlipdog = fun(R, jobs)
        var statRow = NewHTMLPiece("<H4>" + ToString(Size(jobs)) + " jobs found from Flipdog! </H4>");
        InsertBefore(BeginTag(Elem(R, "table")[0]), statRow);

        every job in jobs do
                var pce = NewHTMLPiece("<tr><td>" + Anchor(job.title, job.ref) +"</td><td>" +
                                        job.com  + "</td><td>" + job.loc + "</td><td>" +
                                        job.dd +  " </td><td>" +
                                        Anchor("FlipDog", "http://www.flipdog.com") + "</td></tr>");
                InsertBefore(EndTag(Elem(R, "table")[0]), pce);
        end;

        R
end;

var R = NewPage("<html><body><table></table></body></html>", "text/html");
var headRow = NewHTMLPiece("<H1><center>Job search agent --- WebL demo</center></H1>");
InsertBefore(BeginTag(Elem(R, "table")[0]), headRow);

var titleRow = NewHTMLPiece("<tr><td>Job Title</td><td>Company Name</td><td>Location
                </td><td>Date</td><td>Source</td></tr>");
InsertBefore(EndTag(Elem(R, "table")[0]), titleRow);
```

```
var jobsMonster= QueryMonster(ARGS[1]);
var R1 = displayMonster(R, jobsMonster);

var jobsFlipdog = QueryFlipdog(ARGS[1]);
var R2 = displayFlipdog(R1, jobsFlipdog);

Browser_ShowPage(Markup(R2));
```