Information Integration

Submitted to the
Department of Computer Science
College of Computing Sciences
New Jersey Institute of Technology

in Partial Fulfillment of
the Requirements for the Degree of
Master of Science in Computer Science
by
Saurabh Joshi

&
Azar Daruwalla

Project Advisor: Dr. James Geller

New Jersey Institute of Technology

APPROVALS:

Proposal Number: _____

Approved by: _____
(Project Advisor- Dr. James Geller)

Date Submitted: ____December 2, 2005_____

Final Evaluation: _____
(Graduate Advisor/Committee)

Signature: _____
(Saurabh Joshi)

Signature: _____
(Azar Daruwalla)

Abstract

We have developed a program, using WebL, which performs an analysis of the Internet and provides the user with information about Web sites and most of the Web pages on those sites. Information from these Web pages can be retrieved and tabulated to perform an additional independent analysis later. The primary focus of the program is to crawl all the three lettered sites from www.aaa.com to www.zzz.com and for this purpose an independent algorithm has been implemented. This algorithm generates the three lettered strings from aaa to zzz and then passes them on to the crawler which uses them to scroll through the Web sites. The program also keeps a track of the tables, which exists on these Web pages and then performs an analysis as to how many of these tables have been actually used to present tabulated information to the Web surfer and how many have been just used to just decorate or beautify the Web pages. Finally, the program returns a percentage of the tables used from the total number of tables encountered, which have been used for presenting tabulated information to the users along with other figures such as the total number of pages visited and total number of tables present on all the Web pages. All these numbers can be used to predict how many tables could be found when we consider larger portions of the Internet.

Table of Contents

1. INTRODUCTION:

The goal of the project "Information Integration" is to design a Web Crawler that will crawl through the Internet, integrating the information of the data tables that are present on the Web.

1.1 PROBLEM STATEMENT

The aim of the Information Integration project is to design a Web Crawler that will crawl through the Internet, get the information of the data tables that are present on the Web, then go one step further and retrieve information about the tables from the Web pages and then find out as to what percentage of tables on the Web pages are used to decorate the Web pages and how many are actually used to store relevant data. Finally, the total percentage of the real data tables needs to be calculated based on the information obtained about the total number of tables and the total number of Web pages visited.

1.2 Web Crawler

What is a Web Crawler : The definition for Web crawler is a Web indexing program that builds an index by following hyperlinks continuously from Web page to Web page. The Crawler needs to be designed to suite individual requirements.
To design this Web Crawler which we use as a tool in our project, we made use of a language designed by Compaq Systems Research Center, called as WebL, which is a convenient programming language for the Web [4].

1.3 WebL- Language Selection

WebL (pronounced "Webble") is a Web scripting language for processing documents on the World-Wide Web. It is well suited for retrieving documents from the Web, extracting information from the retrieved documents, and manipulating the contents of documents. In contrast to other general purpose programming languages, WebL is specifically designed for automating tasks on the Web. Not only does the WebL language have built-in knowledge of Web protocols like HTTP and FTP, but it also knows how to process documents in plain text, HTML and XML format.

The flexible handling of structured text markup as found in HTML and XML documents is an important feature of the language. In addition, WebL also supports features that simplify handling of communication failures, the exploitation of replicated documents on multiple Web services for reliability, and performing multiple tasks in parallel. WebL also provides traditional imperative programming language features like objects, modules, closures, control structures, etc. [4].

To give a better idea of how WebL can be applied for Web task automation, and also what makes WebL different from other languages, it is instructive to discuss the computational model that underlies the language. In addition to conventional features you would expect from most languages, the WebL computation model is based on two new concepts, namely *service combinators* and *markup algebra* [4].

Basic Features

- WebL is designed for rapid prototyping of *Web computations*. It is well-suited for the automation of tasks on the WWW.
- WebL's emphasis is on high flexibility and high-level abstractions rather than raw computation speed. It is thus better suited as a rapid prototyping tool than a high-volume production tool.
- WebL is implemented as a stand-alone application that fetches and processes Web pages according to programmed scripts.

Programming Language

- WebL is a high level, imperative, interpreted, dynamically typed, multithreaded, expression, language.
- WebL's standard data types include boolean, character, integer (64-bit), double precision floats, Unicode strings, lists, sets, associative arrays (objects), functions, and methods.
- WebL has prototype-like objects.
- WebL supports fast immutable sets and lists.
- WebL has special data types for processing HTML/XML that include pages, pieces (for markup elements), piece sets, and tags.
- WebL uses conventional control structures like if-then-else, while-do, repeat until, try-catch, etc.
- WebL has a clean, easy to read syntax with C-like expressions and Modula-like control structures.
- WebL supports exception handling mechanisms (based on Cardelli & Davies' service combinators) like sequential combination, parallel execution, timeout,

and retry. WebL can emulate arbitrary complex page fetching behaviors by combining services.

Protocols Spoken

• WebL speaks whatever protocols Java supports, i.e. HTTP, FTP, etc.

• WebL can easily fill in Web-based forms and navigate between pages.

• WebL has HTTP cookie support.

• Programmers can define HTTP request headers and inspect response headers.

• Programmers can explicitly override mime types and DTDs used when parsing

Applications

WebL is a general purpose programming language, and can thus be used to build different applications.

Some of the applications that can be built according to Compaq's Manual are as follows :

• Web shopping robots,

• Page and site validators,

• Meta-search engines,

• Tools to extract connectivity graphs from the Web and analyze them,

• Tools for collecting URLs, host names, word frequency lists,

• Page content analysis and statistics,

• Custom servers and proxy-like entities,

• Locating and downloading multi-media content.

2.1 Hello World

Below is the first program we wrote to get an overall feel of the language. The primary purpose of writing this program was to learn the techniques involved in debugging, compilation and execution of the program and along with that the art of developing basic structures for the bigger programs.

Program “Hello World “

```
begin
        var hw;
        hw = "Hello World";
        PrintLn(hw);
end;
```

In order to edit this program the editor used was TextPad. For compiling the program first we needed to change the extension of the text file from “.txt” to “.webl”. Once the extension had been changed we needed to establish a remote connection to the AFS machines or to the AI machines at NJIT where WebL has already been installed. Once the connection has been established we needed to login with our respective UCID and AFS Passwords and then execute the files which we have saved in our AFS accounts by using the command “webl” followed by the file name.

The errors or exceptions which are generated can be appropriately trapped by using “Try Catch” blocks around the code piece sets where the errors are expected to occur. The basic structure for commenting is exactly that of the C language and in WebL programs we use “//” and “/* */” for comments. In WebL we can also write import statements to facilitate the reuse of code. WebL allows you to package commonly used routines to a module. An example module might be routines to process pages from a specific Web server. Client programs can access the routines by importing the module. This is indicated by the client writing an import statement specifying all the modules that a program requires. After importing a module, the variables declared in that module can be accessed. This is done by writing the module name, followed by an underscore character and the variable name. For example, the following program imports module A, accesses a variable and calls a function in that module:

```
Program
begin
        import A;
        PrintLn(“The value of x in A is “, A_x);
        A_Doit();
end;
```

The import statement may occur only at the beginning of a program. Imported variable references must always be explicitly qualified by the module name and an underscore. Note the choice of the underscore character allows us to separate the variable name space and module name space (i.e. a module and a variable might

have the same name).

One of the side-effects of importing a module is the loading of the module into memory. WebL keeps track of all loaded modules in a global module list. Before importing a module, the module list is checked to see whether the module has been loaded before — if so, the previously loaded module is reused. Thus a module can be loaded only once. There is no operation to unload a module.

2.2 Web Crawler

The following example implements a Web Crawler that prints the URL and title of all pages visited. The crawl is restricted to pages on the *yahoo.com* domain that have a URL that ends in a "/", ".htm" or ".html". The queue is initially seeded with two pages from where the crawl will progress in a breadth-first fashion. Note that the program finally goes to sleep while the crawl is in progress.

2.2.1 Web Crawler Base Class

```
/************** Basic Crawler**************/

import Str, WebCrawler;
var MyCrawler = Clone(WebCrawler_Crawler,
[.
        Visit = meth(s, page);
        var title = Text(Elem(page, "title")[0]) ? "This page has no title";
        PrintLn(page.URL, " title=", title);
end,
```

```
        ShouldVisit = meth(s, url)
        Str_StartsWith(url,‘http://www[.]yahoo[.]com‘)
        and
        Str_EndsWith(url, "(/)|([.]html?)")
end,
.]);
        MyCrawler.Start(3);// Only three threads are used.
        MyCrawler.Enqueue("http://www.yahoo.com/");
Stall();

/***************End of Crawler****************/
```

The creation of the crawler can be explained as follows:

In this example, we illustrate how to build a simple Web Crawler framework that can easily be customized. The basic idea is to define a generic *Crawler* object of which methods can be overridden to customize its behavior. By the way, our crawler implementation is provided as standard in WebL in a module called *Web-Crawler*.

First we define the generic *Crawler* object as follows:

```
1 import Str, Farm;
2
3 export var Crawler
4 [.
5       // Pages visited so far (and removed from queue)
```

```
6       // and pages waiting in the queue.
7       enqueued = [. .],
8
9       // Will contain the farm after the start method is called
10.
11      farm = nil,
12
13      // Method that should be overridden.
14      Visit = meth(s, page) PrintLn(page.URL) end,
15       ShouldVisit = meth(s, url) true end,
16
17      Enqueue = meth(s, url)
18 // First remove everything following #
19 var pos = Str_IndexOf("#", url);
20      if pos != -1 then
21      url = Select(url, 0, pos)
22 end;
23 lock s do
24      var present = s.enqueued[url] ? false;
25              if !present and s.ShouldVisit(url) then
26                      s.enqueued[url] := true;
27                      s.farm.Perform(s.ProcessPage(s, url))
28              end
```

```
29      end
30 end,
31
32 ProcessPage = fun(s, url)
33      try
34              var page = GetURL(url); // fetch the page
35              s.Visit(page);
36
37              // Process all the links from this page.
38              every a in Elem(page, "a") do
39              s.Enqueue(a.href) ? nil
40      end
41      catch E
42              on true do PrintLn(url, " err: ", E.msg)
43      end;
44 end,
45
46 Start = meth(s, noworkers)
47      s.farm = Farm_NewFarm(noworkers);
48 end,
49
50       Abort = meth(s) s.Stop() end
51 .];   // End of Program
```

First we need to keep track of all pages visited so far with an associative array (aka a WebL object called enqueued) where the fields are the visited URLs, and the value is either true or false (line 7). Note that an alternative implementation could use a set instead of an object without any performance penalty [7]. Farm (line 1) introduces a technique for programming and controlling several concurrently executing threads. Lines 14 and 15 define the two methods that need to be overridden to customize the crawler. The *Visit* method is called each time a new page is visited, and the *Should- Visit* method indicates whether a specific URL should be crawled or not.

The *Enqueue* method (lines 17-30) adds a URL to the queue of pages to be fetched. The first task is to strip off any references from the URL (lines 19-22). Line 24 then checks if we visited the page already. Note the use of the ? service combinator to catch the exception should the URL not be in the visited array. If the URL is not present, and we should visit this page (line 25), we remember that we have seen the page (line 26), and then pass the job of retrieving the page to a farm object (line 27). Eventually, when a worker on the farm reaches a new job, the *ProcessPage* function is invoked (lines 32-44). After the page is fetched (line 34), we call the method *Visit* to let the crawler process the page (line 35). Lines 38-40 take care of enqueing all the anchors found on the page.

WebCrawler

Clone applies to the generic crawler and our own object that contains the modifications to the generic crawler we would like to make (lines 3-16).

```
1 import Str, WebCrawler;
2
3 var MyCrawler = Clone(WebCrawler_Crawler,
4       [.
5             Visit = meth(s, page)
6             var title = Text(Elem(page, "title")[0]) ? "N/A";
7             PrintLn(page.URL, " title=", title);
8       end,
9
10      ShouldVisit = meth(s, url)
11            Str_StartsWith(url, ‘http://www-yahoo.com‘)
12            and
13            Str_EndsWith(url, "(/)|(.html?)")
14
15            end,
16      .]);
17
18      MyCrawler.Start(2);
19      MyCrawler.Enqueue("http://www.yahoo.com/");
20      MyCrawler.Enqueue("http://www.yahoo.com/");
21      while !MyCrawler.Idle() do Sleep(10000) end
```

Our particular implementation of the *Visit* method extracts and prints the URL and title of the page (lines 5-8). The *ShouldVisit* method (lines 10-15) restricts crawling to host names of the form "*www.yahoo.com"* and URLs that end either in "/" or ".html".

Lines 18-20 start up the crawler with two workers and enqueue two starting point URLs. Line 21 goes into a loop that checks every 10 seconds whether the workers have become idle, in which case the crawler terminates.

2.2.2 Implementation:

As our basic aim is to develop a Web Crawler to first crawl through the Web and extract the pages we crawl. Certain features for crawling have been provided by the programming language itself. The Web Crawler that we have developed exports a single object called Crawler that implements a low performance multi-threaded Web Crawler. To use the Web Crawler, the methods Visit( ) and ShouldVisit( ) must be overridden by the programmer. The Visit( ) method is called by the Crawler each time a page is visited, and the ShouldVisit( ) method returns true when a specific URL must be crawled. The Crawler is activated by the Start( ) method which takes as argument an integer specifying how many threads should perform the crawl. At this point the Crawler has no pages to crawl yet. Pages are inserted into the crawler queue with the Enqueue( ) method. As each page in the queue is processed, the crawler extracts all the anchor (“<A>”) tags in that page, and calls the ShouldVisit( ) method to determine if the page

referred to by the anchor should be crawled or not. The Abort() method can be called at any time to terminate the crawl.

2.2.3 Problem Faced

When we wrote the crawler to scroll through the web sites there were some problems with the execution of the crawler and the exceptions generated are as follows:

NoSuchField Exception - Object does not have such field.

NotAFunctionOrMethod Exception - Left-hand side is not callable.

ArgumentError Exception - Number of actual and formal arguments do not match.

FieldError Exception - Unknown field or illegal field assignment.

NotAVariable Exception - Left hand side is not a variable.

FieldDefinitionError Exception - Could not define field.

IndexRangeError Exception - Index is out of range.

NotAnObject Exception - Left hand side is not an object.

We studied all these exceptions and then the problems in the code were resolved accordingly. Also we faced problems when the connection to the host servers were lost or when sometimes the Web servers did not return the correct MIME type information for certain documents, which makes it impossible for WebL to parse the document.
The exceptions even though handled properly would make the working thread sleep for 100 seconds thereby resulting in the loss of connection to the server.

We overcame this problem by time limiting the crawler. The thread which was handling the exception was initially made to sleep for 100 seconds before the next Web

address was generated but later in order to prevent the problems caused by loss of connectivity, with the Web server the sleeping time was reduced to 5 seconds.

2.2.4 Controlling the Crawler

Initially when the algorithm for the Web address generation was not implemented the prime task was to control the crawler on any particular site. Just for experimenting we chose www.yahoo.com as the starting Web site for the crawler. When the crawler started crawling through the www.yahoo.com Web site it then visited all the links that existed on the www.yahoo.com homepage and then on all the links that existed on the Web page represented by the first link. In order to test if the data retrieved was accurate, first we had to restrict the crawling only to one specific Web site. Hence we decided to restrict the Web Crawler to Dr. James Geller's Web page. http://web.njit.edu/~geller/ where there was just one table that we needed to retrieve information from. Once it was verified that the program was returning accurate results we tried it on those Web pages which had very few links. In this way we could manually check the results and then implement the logic in the entire program. Still after restricting the crawl to a particular Web page there were problems for the crawler, because crawling from one page to another was like a chain reaction which was not stoppable. There were other problems such as looping where the crawler would crawl from Web site www.abc.com to www.xyz.com and then back to Web site www.abc.com which formed an endless loop. Similarly when the crawler tried to dig into all the Web pages on a particular Web site it would go from one folder on the Web site to another and eventually it would time out before returning the

results. Therefore another strategy was developed where we developed a function which mapped the integers to letters of the English language and then generated three lettered combinations which were passed on to the crawler as a Web site for the crawler to crawl through. This way a lot of problems of looping, timeout and chain reaction were automatically resolved and the desired output was obtained.

2.3 Information Extraction from WebPages

The following are WebL functions commonly used for analyzing Web pages.[4]

Function Description

The functions contain the keywords "piece" and "pieceset" which are explained as follows.

Piece:

A *piece* is a WebL value type that denotes a region of a page. Each piece refers to two tags: the *begin tag* that denotes the start of the region, and the *end tag* that denotes the end of the region. The region includes both the begin and end tag, and everything between them in the page.

Piece Sets:

As its name indicates, a *piece set* is a collection of pieces belonging to the same page. It is a set in the sense that a piece can belong only once to a piece set but a piece can be a member of several piece sets. A piece set is also a list because pieces in a piece set are ordered. The piece ordering in a piece set is based on the begin tag and end tag positions of a piece in a page. The pieces are ordered according to the left-to-right order of the

begin tags. Piece sets play a very important part in WebL. They form the basis of many of the operations that extract data from Web pages.

Elem(p: page): pieceset

Returns all the elements in a page.

Elem(p: page, name: string):pieceset

Returns all the elements in page *p* with a specific *name*.

Elem(q: piece): pieceset

Returns all the elements that are contained (nested) in piece *q*.

Elem(q: piece, name: string):pieceset

Returns all the elements with a specific *name* contained in piece *q*.

Para(p: page, paraspec: string):pieceset

Extracts the paragraphs in *p* according to the paragraph terminator specification

Para(p: piece, paraspec: string):pieceset

Extracts the paragraphs in *p* according to the paragraph terminator specification

Pat(p: page, regexp: string): pieceset

Returns all the occurrences of a regular expression pattern in page *p*.

Pat(q: piece, regexp: string):pieceset

Returns all the occurrences of a regular expression pattern located inside the piece *q*.

PCData(p: page): pieceset

Returns the "parsed character data" of the page. This corresponds to the individual seqences of text on the page, as delimited by markup tags.

PCData(p: piece): pieceset

Returns the “parsed character data” of the piece. This corresponds to the individual seqences of text inside the piece, as delimited by markup tags.

Seq(p: page, pattern: string):pieceset

Matches all the occurrences of a sequence of elements identified by search” on pattern.

Seq(p: piece, pattern: string):pieceset

Matches all the occurrences of a sequence of elements identified by pattern inside the piece *p*.

All these functions were tried and tested at different times and then finally the Elem( ) function was used as it was giving the most useful results of all the functions. Sometimes there were a lot of exceptions and it was very difficult to debug the program, as there was inadequate knowledge available and also, as the language is no longer supported, the documentation available is limited. The errors generated did not explicitly specify which parts of the code were causing the problem and this is why we decided to move onto JAVA to make the debugging process is simpler.

2.4 Shift to Java Coding

We had by now started running small snippets of WEBL code on Web pages from www.aaa.com to www.aaz.com to test our program, but through out the trial and error phase we encountered situations where the program would die. At certain times we felt this was the limitation of the WEBL language itself but at other times it was caused by the network connection freezing, since we were running the program by accessing our college AI servers and those programs were running for hours and days. So, during the testing phase we decided to give Java a try, which we could execute from our respective home systems and we were also counting on Java's stability. We started researching Java Web crawlers and ways to extract information while crawling. Since we had a time constraint, we had to be careful not to spend too much time on Java and then make a decision of not continuing with it. [6]

One of the programs that we wrote to test the functionality of Java to implement the crawler is shown in Appendix A.

Pseudo code summary of the program: [5]

1) Get the user's input: the starting URL and the desired file type.
2) Add the URL to the currently empty list of URLs to search. While the list of URLs to search is not empty,

3) Get the first URL in the list.
4) Move the URL to the list of URLs already searched. Check the URL to make sure its protocol is HTTP,
5) If not, break out of the loop, back to "While".
6) See whether there's a robots.txt file at this site that includes a "Disallow" statement.
7) (If so, break out of the loop, back to "While".)
8) Try to "open" the URL (that is, retrieve that document from the Web).
9) If it's not an HTML file, break out of the loop, back to "While."
10) Step through the HTML file. While the HTML text contains another link, Validate the link's URL and make sure robots are allowed (just as in the outer loop).
11) If it's an HTML file, If the URL isn't present in either the to-search list or the already searched list, add it to the to-search list.
12) Else if it's the type of the file the user requested, add it to the list of files found.

After spending some time on the Java Web Crawler, we decided to go back to WEBL, since unlike WEBL, Java did not have any pre-created class such as the Crawler class which is used in our program to crawl through the Web. WEBL is a language which is designed to develop Web Crawlers, as compared to Java which is a much vast programming language and learning Java itself requires a complete semester. To create an efficient Web Crawler that would extract information about tables on each Web page

would definitely be quite time consuming. Also WEBL provided other programmer friendly methods such as Elem() which could be used to extract information on tables. Due to the above reasons we decided to revert back to WEBL.

2.5 Analysis

We will now discuss our main program shown in Appendix B.

First we developed a function that takes an integer as an input and returns the corresponding character as an output. For example if the input is 1 the output is 'a', so if the input to the method is 111 the output would be aaa. We now crawl the Web page www.aaa.com. Then we run the main program consisting of three nested loops, a loop for each character from www.aaa.com to www.zzz.com. The third most character from left is updated most frequently. Every time the third most character completes one round from a to z the second character is incremented by one. Once the second character completed a round from a to z, increments the first character by one.

Example:

aaa, aab, aac ….. aaz, aba, abb, abc ….. abz, ….. azz, baa, bab, bac ……. zzz.

If we wanted to crawl www.aaaa.com to www.zzzz.com then we would have 4 loops instead.

After much consideration and much analysis we decided on using the Elem function to access all table tags and tr tags in the Web page. (The Elem() function is explained in Section 2.3). The Elem() function helps us in extracting all the information about the table. HTML stands for **H**yper **T**ext **M**arkup **L**anguage. An HTML file is a text file

containing small markup tags. The markup tags tell the Web browser how to display the page.

Example:

```
<html>
<head>
<title>Title of page</title>
</head>
<body>
This is my first homepage. <b>This text is bold</b>
</body>
</html>
```

Tables are defined with the <table> tag. A table is divided into rows (with the <tr> tag), and each row is divided into data cells (with the <td> tag). The letters "td" stands for "table data," which is the content of a data cell. A data cell can contain text, images, lists, paragraphs, forms, horizontal rules, etc.

```
<table border="1">
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
```

</tr>

</table>

The above HTML code looks as follows in a browser:

Row 1, cell 1	row 1, cell 2
Row 2, cell 1	row 2, cell 2

We have imported the “WebCrawler” class the helps us crawl the Web. The variable “MyCrawler” is a clone of a “WebCrawler” object providing it with all the functionality provided by the “WebCrawler” class. Within the loops we keep a count of the Web pages traversed, a count of all the tables within the tables and all the rows within those tables. We spawn 3 threads to work in parallel, crawling and extracting information from the Web pages. To control the crawler we are only crawling all HTML pages.

At the end of crawling from www.aaa.com to www.zzz.com we are providing information about the total number of Web pages, the total number of tables in those pages, the total number of real tables in them and finally how many of those tables are only for layout.

Our conclusion from observing various Web pages is that if a table has more than 10 rows, it can reasonably be said that those tables are real tables containing data while the others are more likely for layout.

3. Observations and Recommendations

During the course of the whole project we have had many set backs and faced many problems. Firstly when running the crawler from www.aaa.com to www.aaz.com, the crawler died regularly at www.aay.com. There were many observations made in regards to the program dying, one being that WEBL is a language that has become obsolete and has not been updated for some time now. Secondly we were never able to run WEBL on our systems owing to our paucity of knowledge about it. We were able to figure out how to run our programs on our college AI server machines. But since we were running the programs from home there used to be situations where the network connection would freeze, leaving the program at some particular point where we did not have much information about the total number of Web pages or tables. Also, WEBL does not have features like Java in regards to writing to a File or accessing some kind of database, making life little more difficult.

Since we hardly knew what each function did in WEBL, we had to crawl the Web and then execute each method to verify what it does and then make some kind of decision on how to move ahead. The other main problem with WEBL was with the syntax. The syntax is quite different compared to Java and not consistent at some places.

Our recommendation in regards to the project would be to use our logic to write the same program in Java.

4. Conclusion

In general the project revolved around extracting information about tables from the Internet. We have been successful in crawling through a limited part of the Internet to find out as to how many Web pages exists in a limited domain such as www.aaa.com to www.zzz.com. Another result obtained was the number of the tables that exist on the pages visited by the crawler. Then an analysis was performed mathematically to find out how many tables had more than ten rows of data. In general it was observed that the tables with more than ten rows were actually used for storing data on the Web pages, while those tables with fewer than ten rows were used for the layout of the Web pages. After determining the number of tables used for layout and the number of tables actually used for data representation the percentage of real tables in the total table was found out to aid the research.

The output of the program we have written, provides us with the information needed for the analysis.

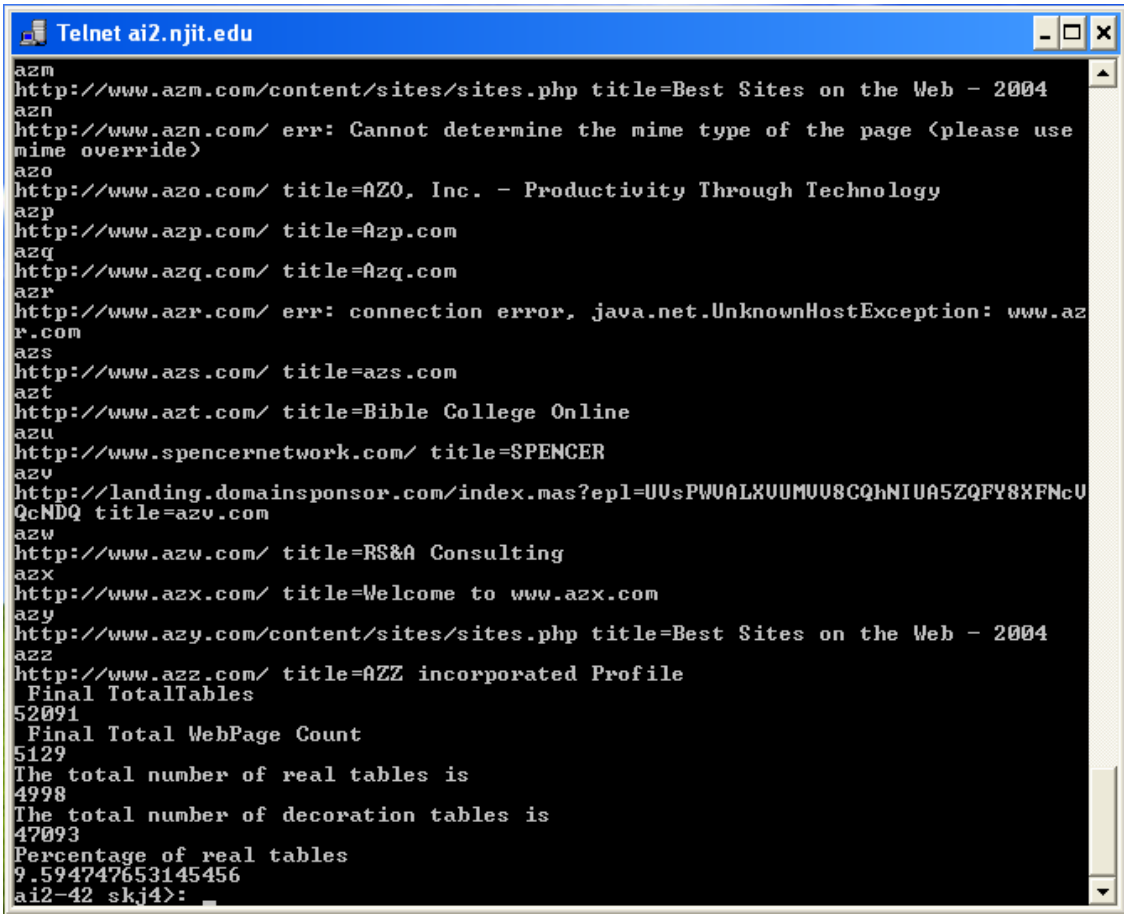


Figure 1: Sample Output

The output shown in Figure 1 shows that the Web crawler crawled through the Web sites www.aaa.com to www.aaz.com. In this run the crawler scrolled through 5129 Web pages and found a total of 52091 tables on these Web pages. Also out of these 52091 tables 4998 tables were used to display tabulated data on the Web pages and 47093 tables were actually used just for designing the layout of the Web pages. Thereby, out of the 52091 tables only 9.59% of the tables were used to represent actual data on the pages. Similarly we have all the other results tabulated for an overall analysis

Web Sites	Total Web Pages Crawled	Total Tables	Total Number of tables used for storing actual data	Total Number of tables used for layout	Percentage of Tables used for storing data
www.aaa.com-www.aaz.com	5129	52091	4998	47093	9.59
www.baa.com-www.bzz.com	6751	57485	5412	52073	9.41
www.caa.com-www.czz.com	7281	69423	4389	65034	6.32
www.daa.com-www.dzz.com	6126	46400	3897	42503	8.39
www.eaa.com-www.ezz.com	5957	56321	4315	52006	7.66
www.faa.com-www.fzz.com	5128	62343	6189	56154	9.92
www.gaa.com-www.gzz.com	3904	56512	5123	51389	9.06
www.haa.com-www.hzz.com	4858	43525	4393	39132	10.09
	45134	444100	38716	405384	

Web Sites	Total Web Pages Crawled	Total Tables	Total Number of tables used for storing actual data	Total Number of tables used for layout	Percentage of Tables used for storing data
www.iaa.com-www.izz.com	7132	65775	5219	60556	7.93
www.jaa.com-www.jzz.com	7264	84951	4388	80563	5.16
www.kaa.com-www.kzz.com	4467	44186	5897	38289	13.34
www.laa.com-www.lzz.com	5919	63423	4188	59235	6.60
www.maa.com-www.mzz.com	8914	54565	6249	48316	11.45
www.naa.com-www.nzz.com	13782	35792	6713	29079	18.75
www.oaa.com-www.ozz.com	7561	60137	6718	53419	11.17
www.paa.com-www.pzz.com	8291	68038	5931	62107	8.71

Web Sites	Total Web Pages Crawled	Total Tables	Total Number of tables used for storing actual data	Total Number of tables used for layout	Percentage of Tables used for storing data
www.qaa.com-www.qzz.com	7445	53528	4807	48721	8.98
www.raa.com-www.rzz.com	9124	54962	3926	51036	7.14
www.saa.com-www.szz.com	6786	71092	7894	63198	11.10
www.taa.com-www.tzz.com	5640	51791	6641	45150	12.82
www.uaa.com-www.uzz.com	8128	63541	3528	60013	5.55
www.vaa.com-www.vzz.com	5903	42775	3423	39352	8.02
www.waa.com-www.wzz.com	4981	52289	4112	48177	7.86
www.xaa.com-www.xzz.com	13457	74193	6174	68019	8.32
www.yaa.com-www.yzz.com	9852	70108	5927	64181	8.45

Web Sites	Total Web Pages Crawled	Total Tables	Total Number of tables used for storing actual data	Total Number of tables used for layout	Percentage of Tables used for storing data
www.zaa.com-www.zzz.com	4983	43857	3729	40128	8.50
Total	**184763**	**1499103**	**134180**	**1364923**	**8.95**

Table 1: Output Table

The output table in Table 1 shows the complete summary of the program outputs, generated by the crawler while crawling through the Web sites www.aaa.com to www.zzz.com. In this run the crawler scrolled through 184763 Web pages and found a total of 1499103 tables on these Web pages. Also out of these 1499103 tables 134180 tables were used to display tabulated data on the Web pages and 1364923 tables were actually used just for designing the layout of the Web pages. Thereby, out of the 1499103 tables only 8.95% of the tables were used to represent actual data on the pages.

5. Future Work

As future work we recommend using Java, which is much more stable, syntactically consistent and not obsolete as WEBL. Using Java, one could also take other factors into account, such as the total number of columns, to make much more accurate assumptions in regards to which tables are real data table or just for layout. Using Java one could also write all the information that was collected in regards to the total number of Web pages, total number of tables in those Web pages and which of those tables have actual data in them, into a file to make analysis easier.

The same crawler could then be used to extract certain kinds of information from tables, Example: Information about tables that have pictures, tables that have pie charts or bar diagrams or specific information within tables such as cost of a particular product or any sensitive information. It could also search for certain patterns of data in the tables or to do a page content analysis and statistics. One could also add bridges between the WEBL and Java code, It is possible to call Java objects directly from WEBL code, without extending the WEBL system.

References:

1) Hannes Marais, Automating the Web with WebL. Bay Area Roundtable, June12, 1998.

2) Hannes Marais, Concepts and applications of the WebL Programming Language. SRI, November 9, 1998 and UC Davis, November 12, 1998.

3) Hannes Marais and Tom Rodeheffer. Automating the Web with WebL. In Dr. Dobb's Journal, January 1999.

4) 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, also appeared as SRC Technical Note 1997-029

5) Web site http://java.sun.com/

6) Cay S. Hortsman, Gary Cornell. Core Java – Advanced Features Sun Microsystems Press 2nd Edition.

7) WebL - A programming language for the Web. In *Computer Networks and ISDN Systems* (Proceedings of the WWW7 Conference), Volume 30, page 154.

Appendix A – Java Code

```
import java.applet.Applet;
import java.text.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.net.*;
import java.io.*;

public class WebCrawler extends Applet implements ActionListener, Runnable {
    public static final String SEARCH = "Search";
    public static final String STOP = "Stop";
    public static final String DISALLOW = "Disallow:";
    public static final int    SEARCH_LIMIT = 50;

    Panel   panelMain;
    List    listMatches;
    Label   labelStatus;

    Vector vectorToSearch;
```

```
Vector vectorSearched;
Vector vectorMatches;

Thread searchThread;
TextField textURL;
Choice    choiceType;

public void init() {
        panelMain = new Panel();
        panelMain.setLayout(new BorderLayout(5, 5));
        Panel panelEntry = new Panel();
        panelEntry.setLayout(new BorderLayout(5, 5));

        Panel panelURL = new Panel();
        panelURL.setLayout(new FlowLayout(FlowLayout.LEFT, 5, 5));
        Label labelURL = new Label("Starting URL: ", Label.RIGHT);
        panelURL.add(labelURL);
        textURL = new TextField("", 40);
        panelURL.add(textURL);
        panelEntry.add("North", panelURL);
        Panel panelType = new Panel();
        panelType.setLayout(new FlowLayout(FlowLayout.LEFT, 5, 5));
        Label labelType = new Label("Content type: ", Label.RIGHT);
```

```
panelType.add(labelType);
choiceType = new Choice();
choiceType.addItem("text/html");
choiceType.addItem("audio/basic");
choiceType.addItem("audio/au");
choiceType.addItem("audio/aiff");
choiceType.addItem("audio/wav");
choiceType.addItem("video/mpeg");
choiceType.addItem("video/x-avi");
panelType.add(choiceType);
panelEntry.add("South", panelType);

panelMain.add("North", panelEntry);

// list of result URLs
Panel panelListButtons = new Panel();
panelListButtons.setLayout(new BorderLayout(5, 5));

Panel panelList = new Panel();
panelList.setLayout(new BorderLayout(5, 5));
Label labelResults = new Label("Search results");
panelList.add("North", labelResults);
Panel panelListCurrent = new Panel();
```

```
panelListCurrent.setLayout(new BorderLayout(5, 5));
listMatches = new List(10);
panelListCurrent.add("North", listMatches);
labelStatus = new Label("");
panelListCurrent.add("South", labelStatus);
panelList.add("South", panelListCurrent);

panelListButtons.add("North", panelList);

// control buttons
Panel panelButtons = new Panel();
Button buttonSearch = new Button(SEARCH);
buttonSearch.addActionListener(this);
panelButtons.add(buttonSearch);
Button buttonStop = new Button(STOP);
buttonStop.addActionListener(this);
panelButtons.add(buttonStop);

panelListButtons.add("South", panelButtons);

panelMain.add("South", panelListButtons);
add(panelMain);
setVisible(true);
```

```
        repaint();

        // initialize search data structures
        vectorToSearch = new Vector();
        vectorSearched = new Vector();
        vectorMatches = new Vector();

        // set default for URL access
        URLConnection.setDefaultAllowUserInteraction(false);
    }

    public void start() {
    }

    public void stop() {
        if (searchThread != null) {
            setStatus("stopping...");
            searchThread = null;
        }
    }
    public void destroy() {
    }
```

```
boolean robotSafe(URL url) {
        String strHost = url.getHost();
                String strRobot = "http://" + strHost + "/robots.txt";
        URL urlRobot;
        try {
           urlRobot = new URL(strRobot);
        } catch (MalformedURLException e) {
           return false;
        }
        String strCommands;
        try {
           InputStream urlRobotStream = urlRobot.openStream();
           byte b[] = new byte[1000];
           int numRead = urlRobotStream.read(b);
           strCommands = new String(b, 0, numRead);
           while (numRead != -1) {
                if (Thread.currentThread() != searchThread)
                   break;
                numRead = urlRobotStream.read(b);
                if (numRead != -1) {
                   String newCommands = new String(b, 0, numRead);
                   strCommands += newCommands;
```

```
            }
        }
        urlRobotStream.close();
    } catch (IOException e) {
        // if there is no robots.txt file, it is OK to search
        return true;
    }

    // assume that this robots.txt refers to us and
    // search for "Disallow:" commands.
    String strURL = url.getFile();
    int index = 0;
    while ((index = strCommands.indexOf(DISALLOW, index)) != -1) {
        index += DISALLOW.length();
        String strPath = strCommands.substring(index);
        StringTokenizer st = new StringTokenizer(strPath);

        if (!st.hasMoreTokens())
            break;
        String strBadPath = st.nextToken();
        if (strURL.indexOf(strBadPath) == 0)
            return false;
    }
```

```
        return true;
    }
    public void paint(Graphics g) {
        //Draw a Rectangle around the applet's display area.
        g.drawRect(0, 0, getSize().width - 1, getSize().height - 1);

        panelMain.paint(g);
        panelMain.paintComponents(g);
        // update(g);
        // panelMain.update(g);
    }

    public void run() {
        String strURL = textURL.getText();
        String strTargetType = choiceType.getSelectedItem();
        int numberSearched = 0;
        int numberFound = 0;
        if (strURL.length() == 0) {
           setStatus("ERROR: must enter a starting URL");
           return;
        }

        // initialize search data structures
```

```
vectorToSearch.removeAllElements();
vectorSearched.removeAllElements();
vectorMatches.removeAllElements();
listMatches.removeAll();

vectorToSearch.addElement(strURL);

while ((vectorToSearch.size() > 0)  && (Thread.currentThread() ==
searchThread))
{
    // get the first element from the to be searched list
    strURL = (String) vectorToSearch.elementAt(0);
    setStatus("searching " + strURL);

    URL url;
    try {
        url = new URL(strURL);
    } catch (MalformedURLException e) {
        setStatus("ERROR: invalid URL " + strURL);
        break;
    }
    // mark the URL as searched (we want this one way or the other)
    vectorToSearch.removeElementAt(0);
```

```
vectorSearched.addElement(strURL);
if (url.getProtocol().compareTo("http") != 0)
        break;
if (!robotSafe(url))
        break;

try {
        // try opening the URL
        URLConnection urlConnection = url.openConnection();

        urlConnection.setAllowUserInteraction(false);
        InputStream urlStream = url.openStream();
        String type
          = urlConnection.guessContentTypeFromStream(urlStream);
        if (type == null)
           break;
        if (type.compareTo("text/html") != 0)
           break;
        byte b[] = new byte[1000];
        int numRead = urlStream.read(b);
        String content = new String(b, 0, numRead);
        while (numRead != -1) {
           if (Thread.currentThread() != searchThread)
```

```
            break;
        numRead = urlStream.read(b);
        if (numRead != -1) {
            String newContent = new String(b, 0, numRead);
            content += newContent;
        }
    }
    urlStream.close();

    if (Thread.currentThread() != searchThread)
        break;
    String lowerCaseContent = content.toLowerCase();
    int index = 0;
    while ((index = lowerCaseContent.indexOf("<a", index)) != -1)
    {
        if ((index = lowerCaseContent.indexOf("href", index)) == -1)
            break;
        if ((index = lowerCaseContent.indexOf("=", index)) == -1)
            break;
        if (Thread.currentThread() != searchThread)
            break;
        index++;
        String remaining = content.substring(index);
```

```
StringTokenizer st
  = new StringTokenizer(remaining, "\t\n\r\">#");
String strLink = st.nextToken();


URL urlLink;
try {
        urlLink = new URL(url, strLink);
        strLink = urlLink.toString();
} catch (MalformedURLException e) {
        setStatus("ERROR: bad URL " + strLink);
        continue;
}
if (urlLink.getProtocol().compareTo("http") != 0)
        break;
if (Thread.currentThread() != searchThread)
        break;
try {
        // try opening the URL
        URLConnection urlLinkConnection
          = urlLink.openConnection();
        urlLinkConnection.setAllowUserInteraction(false);
        InputStream linkStream = urlLink.openStream();
```

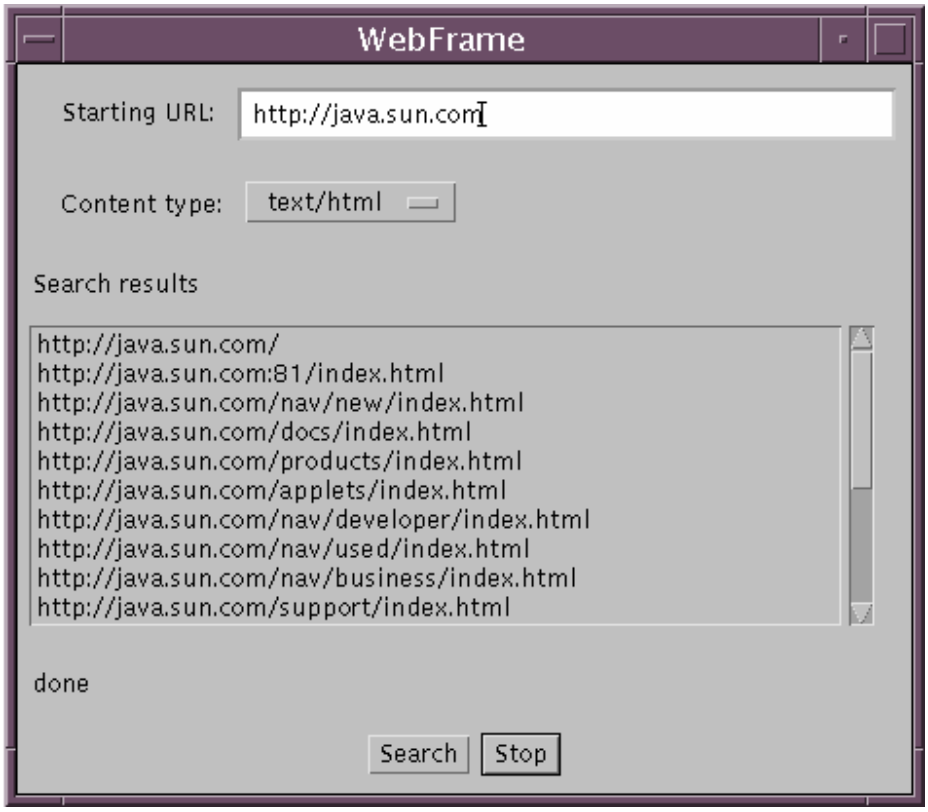
```
String strType
  = urlLinkConnection.guessContentTypeFromStream(linkStream);
linkStream.close();
if (strType == null)
   break;
if (strType.compareTo("text/html") == 0) {
   // check to see if this URL has already been
   // searched or is going to be searched
   if ((!vectorSearched.contains(strLink))
     && (!vectorToSearch.contains(strLink))) {
         if (robotSafe(urlLink))
            vectorToSearch.addElement(strLink);
   }
}
if (strType.compareTo(strTargetType) == 0) {
   if (vectorMatches.contains(strLink) == false) {
         listMatches.add(strLink);
         vectorMatches.addElement(strLink);
         numberFound++;
         if (numberFound >= SEARCH_LIMIT)
            break;
   }
}
```

```
                } catch (IOException e) {
                    setStatus("ERROR: couldn't open URL " + strLink);
                    continue;
                }
            }
        } catch (IOException e) {
            setStatus("ERROR: couldn't open URL " + strURL);
            break;
        }

        numberSearched++;
        if (numberSearched >= SEARCH_LIMIT)
            break;
    }
    if (numberSearched >= SEARCH_LIMIT || numberFound >= SEARCH_LIMIT)
        setStatus("reached search limit of " + SEARCH_LIMIT);
    else
        setStatus("done");
        searchThread = null;
}
void setStatus(String status) {
    labelStatus.setText(status);
}
```

```
public void actionPerformed(ActionEvent event) {
        String command = event.getActionCommand();
        if (command.compareTo(SEARCH) == 0) {
            setStatus("searching...");
            if (searchThread == null) {
                    searchThread = new Thread(this);
            }
            searchThread.start();
        }
        else if (command.compareTo(STOP) == 0) {
            stop();
        }
}
    public static void main (String argv[])
    {
            Frame f = new Frame("WebFrame");
            WebCrawler applet = new WebCrawler();
                    f.add("Center", applet);
            Properties props= new Properties(System.getProperties());
            props.put("http.proxySet", "true");
            props.put("http.proxyHost", "webcache-cup");
            props.put("http.proxyPort", "8080");
            Properties newprops = new Properties(props);
```

```
            System.setProperties(newprops);
            applet.init();
            applet.start();
            f.pack();
            f.show();
        }
}
```



Appendix B WEBL Code:

```
// A demonstration web crawler that restricts its
// crawl to a specific site.
import Str, WebCrawler;
var i ;  var j; var k;
var WebPageCount;
var n; var RealTables; var DecorationTables; var p; var string; var TotalTables;
i = 0; j = 0; k = 0; n = 0;
TotalTables = 0; WebPageCount = 0; RealTables = 0; DecorationTables = 0; p = 0;
//=================================================================
var F = fun(a)
        if a == 1 then return "a"
                elsif a == 2 then return "b"
                elsif a == 3 then return "c"
                elsif a == 4 then return "d"
                elsif a == 5 then return "e"
                elsif a == 6 then return "f"
                elsif a == 7 then return "g"
                elsif a == 8 then return "h"
                elsif a == 9 then return "i"
```

```
                elsif a == 10 then return "j"
                elsif a == 11 then return "k"
                elsif a == 12 then return "l"
                elsif a == 13 then return "m"
                elsif a == 14 then return "n"
                elsif a == 15 then return "o"
                elsif a == 16 then return "p"
                elsif a == 17 then return "q"
                elsif a == 18 then return "r"
                elsif a == 19 then return "s"
                elsif a == 20 then return "t"
                elsif a == 21 then return "u"
                elsif a == 22 then return "v"
                elsif a == 23 then return "w"
                elsif a == 24 then return "x"
                elsif a == 25 then return "y"
                elsif a == 26 then return "z"

        end;
end;

while i < 26 do
//while i < 2 do
```

```
i = i + 1;
            while j < 26 do
            //while j < 26 do

                        j = j + 1;
                        while k < 26 do
                        //while k < 3 do
                        k = k + 1;
                        string =  F(i) + F(j) + F(k);
                        PrintLn(string);
//=================================================================
                    var MyCrawler = Clone(WebCrawler_Crawler,
                    [.
                        Visit = meth(s, page)
                        var title = Text(Elem(page, "title")[0]) ? "notitle";
                        PrintLn(page.URL, " title=", title);
                        WebPageCount = WebPageCount + 1;

                        n = 0;
                        var q = Elem(page, "table");
                        every table in q do
                        p = 0;
```

```
                    n = n + 1;
                    TotalTables = TotalTables + 1;
                    var qtr = Elem(table, "tr");
                    every tr in qtr do
                            p = p + 1
                    end;
                    if p > 10 then
                            RealTables = RealTables + 1;
                    end;
            end;
end,
ShouldVisit = meth(s, url)
            Str_StartsWith(url, `http://www[.]`+string+`[.]com/`)
                            and
            Str_EndsWith(url, "(/)|(index.html?)")
            end,
 .]);
     MyCrawler.Start(3);
      // use 3 worker threads to crawl site
      MyCrawler.Enqueue("http://www."+string+".com/");
      while !MyCrawler.farm.Idle() do Sleep(5) end;
      end;  //k end
            k = 1;
```

```
        end;
        j = 1;// j end

        PrintLn (" Final TotalTables");
        PrintLn (TotalTables);
        PrintLn (" Final Total WebPage Count");
        PrintLn (WebPageCount );
       PrintLn ("The total number of real tables is");
       PrintLn (RealTables );
       PrintLn ("The total number of decoration tables is");
       PrintLn (TotalTables - RealTables);
       PrintLn ("Percentage of real tables");
       PrintLn ((RealTables / TotalTables) * 100);
   end; // i end
```

Appendix C-User Manual

The WEBL program that we were running was saved on your local AFS account. We used to access the AI machine from our local systems using Telnet.

To access the AI machine from your Windows system, go to start -> Run, then type "telnet ai1.njit.edu". This will open the particular AI systems login page where we enter the AFS user id and password. On entering the correct user id and password you will get to the AI machine prompt.

Example: ai1-41 ahd2>:

At the prompt you can enter "webl <filename>.webl". This shall compile and start running the WEBL code.

The college AFS account already has WEBL loaded there by making our task of compiling and running the program easier.

Since our programs used to die out we had split our programs into files. For 26 alphabets there are 26 files, each file labeled as, Example:

aaa-azz.webl, baa-bzz.webl, caa-czz.webl ….. zaa-zzz.webl.

So to run the first file for website www.aaa.com to www.aaz.com you will have to run the file aaa-azz.webl by entering "webl aaa-azz.webl" at the prompt.

The programs are in "afs/cad/research/p/2/TableSearch/Azar" directory.

Appendix D – Allocation of Work

Sections	What?	Who did?
I	Title Page	Saurabh Joshi
Ii	Approval Page	Azar Daruwalla
Iii	Abstract	Saurabh Joshi
Iv	Table of Contents	Azar Daruwalla
1.1	Problem Statement	Saurabh Joshi
1.2	Web Crawler	Saurabh Joshi
1.3	WebL Language Selection	Azar Daruwalla
2.1	Hello World	Saurabh Joshi
2.2.1	Web Crawler – Base Class	Saurabh Joshi
2.2.2	Web Crawler - Implementation	Saurabh Joshi
2.2.3	Web Crawler – Problems Faced	Saurabh Joshi
2.2.4	Web Crawler – Controlling the Crawler	Saurabh Joshi
2.3	Information Extraction from WebPages	Saurabh Joshi
2.4	Shift to Java Coding	Azar Daruwalla
2.5	Analysis	Azar Daruwalla
3	Observation and Recommendations	Azar Daruwalla
4	Conclusion	Saurabh Joshi
5	Future Work	Azar Daruwalla
	References	Azar Daruwalla
Appendix A – Output		Saurabh Joshi
Appendix B – Program Code		Azar Daruwalla
Appendix C – User Manual		Azar Daruwalla
Appendix D – Allocation of Work		Saurabh Joshi