“Discovering & integrating Object Database schemas of B2C Web Sites”

Project Report

Submitted by

****

60-539-01

Winter 2012

School of Computer Science

University of Windsor

Professor: Dr. Christie Ezeife

Abstract:

Web mining is the integration of information gathered by traditional data mining methodologies and techniques with information gathered over the World Wide Web. Web mining allows you to look for patterns in data through content mining, structure mining, and usage mining. Content mining is used to examine data collected. Structure mining is used to examine data related to the structure of particular web sites. This project report focus on an application which uses both of these mining types and integrates the database schema derived from a web page related to one particular attribute that has the frequent patterns. This way the dynamic new data structure and attributes can be updated in the database. This mining tool developed does not depend on any static data so there is no need of constant update in the code. Database schema is generated with the web page given to the mining tool. Data types are chosen with the analysis of the values for a particular attribute.

1. Introduction

With the increase in the growth of information sources available on the World Wide Web, it has become increasingly necessary for users to utilize automated tools in finding the desired information resources, and to track and analyze their usage patterns. These factors give rise to the necessity of creating server-side and client-side intelligent systems that can effectively mine for knowledge. Web mining can be broadly defined as the discovery and analysis of useful information from the World Wide Web. This describes the automatic search of information resources available on-line, i.e. Web content mining, and the discovery of user access patterns from Web servers, i.e., Web usage mining.

Current Web sites present information on various topics in various formats. A great amount of effort is often required for a user to manually locate and extract useful data from the Web sites. Therefore, there is a great need for value-added service that integrates information from multiple sources. For example, customizable Web information gathering robots/crawlers, comparison-shopping agents, meta-search engines and news bots, etc. To facilitate the development of these information integration systems, we need good tools for information gathering and extraction. Suppose the data has been collected from different Web sites, a conventional approach for extracting data from various Web pages would have to write programs, called “wrappers” or “extractors”, to extract the contents of the Web pages based on a priori knowledge of their format. In other words, we have to observe the extraction rules in person and write programs for each Web site. However, programming wrappers require manual coding which generally entails extensive debugging and is, therefore, labor-intensive. In addition, since the format of Web pages is often subject to change, maintaining the wrapper can be expensive and impractical.

This report provides an overview of the tools and extends the structured automatic data extraction technique. It includes the studies about the idea of modeling web contents in objects and develops a mining process for object-oriented data model for web content integration or comparative mining.

I. Problem Statement:

In the development of data mining tools, the extracted data are analyzed and the extracted information is stored in the data ware house. The schema which stores the data are meant to be static which are created ones and the data are formatted before it is send to the data ware house. The data provided in the web are dynamic. The information provided about an attribute can be added or removed and the schema has to be updated according to that in the database which changes the database structure too. Without the knowledge of the schema of an attribute the database created with static fields are hard to maintain. Moreover the fields in every web site are populated in very different manner with different fields. So there is a need of development of a generic tool which extracts the schema of an attribute for a particular web site and then the data extracted are store the data with the database schema created.

II. Purpose of this application:

This application serves as an extractor of schema. It uses the concept of both structure mining and concept mining. Structure mining is used to find the frequent pattern in the whole web page and find the root of the xml which has the data of our focus. Once the parent of the xml which has the data related to the attribute of focus is found then concept mining is done to extract the fields and its data types.

This application creates a schema by analysing the structure and the content which is related to the products for now. This application is created after the analysis of different B2C web sites and their structures used to populate the information about the products in their web sites for customer's access. The study revealed the fact that almost all the B2C web sites provide the information about the products in the form of grid put into a block. The information is pulled from the database and the server-side script forms a block to envelope the product information and send to the client. This is the general idea web developers use to populate the product data in the web sites. I have used this piece of information to develop schema extractor of product attribute from any web sites with the link to web page which is populated with products.

2. Implemented modules

There are 3 Modules implemented in this project to accomplish the task of extracting the schema from a web page which displays the products and their related information. They are

1. HTML Cleaner Module
2. Cleaned HTML Parser
3. Frequent pattern structure finder

This section explains about my contribution towards this project of extracting the schema from a product web page. The project folder is provided along with this report namely *extractSchema* which has 2 folders namely *inputUncleanedHTML* and *outputCleanedHTML*.

- *inputUncleanedHTML*, This folder has the product HTML web pages named with their corresponding business domain i.e. bestbuy, futureshop, dell.... etc. These unclean HTML web pages are downloaded using a web scraper tool which is not focused in this project report. The HTML files in this location are the input to the HTML Cleaner Module.

- *outputCleanedHTML,* This folder has the product HTML web pages named with their corresponding business domain ending with an underscore i.e. bestbuy_, futureshop_, dell_, ... etc. These Cleaned HTML web pages are generated by HTML Cleaner Module.

The Java *file extractSchema.java* implements the extraction of schema from the products HTML web page. It has 2 modules included in it they are Cleaned HTML Parser and Frequent pattern structure finder. The Java file *runBatchFile.java* implements creating or editing the batch file *cleanHTML.bat* with command to use the *htmlcleaner-2.2.jar* and then executing it. This creates the cleaned HTML product web page.

I. HTML Cleaner Module

I have used html cleaner to clean up the HTML. HtmlCleaner is open-source HTML parser written in Java. HTML found on Web is usually dirty, ill-formed and unsuitable for further processing. For any serious consumption of such documents, it is necessary to first clean up the mess and bring the order to tags, attributes and ordinary text. For the given HTML document, HtmlCleaner reorders individual elements and produces well-formed XML. By default, it follows similar rules that the most of web browsers use in order to create Document Object Model. However, user may provide custom tag and rule set for tag filtering and balancing.

Here is a typical example - improperly structured HTML containing unclosed tags and missing quotes:

```
<table id=table1 cellspacing=2px

   <h1>CONTENT</h1>

   <td><a href=index.html>1 -> Home Page</a>

   <td><a href=intro.html>2 -> Introduction</a>
```

Table 1 unclean HTML

After putting it through HtmlCleaner, XML similar to the following is coming out:

```
<?xml version="1.0" encoding="UTF-8"?>
  <html>
    <head />
    <body>
      <h1>CONTENT</h1>
      <table id="table1" cellspacing="2px">
        <tbody>
          <tr>
          <td>
 <a href="index.html">1 -&gt; Home Page</a>
          </td>
```

```
          <td>
            <a href="intro.html">2 -&gt; Introduction</a>
          </td>
        </tr>
      </tbody>
    </table>
  </body>
</html>
```

Table 2 Clean HTML

This module is implemented in a java file namely *runBatchFile.java*. It has a function *runFile()* which is called with an argument of the name of a file present in the folder *inputUncleanedHTML* in the project. The input file given to this function is an unclean HTML web page with products displayed in it. This function checks for existence of *cleanHTML.bat* in the project is it exists then this batch file will be edited to update the source file and destination file name of the unclean and cleaned HTML web page in a command. This file names are given as the input to the *htmlcleaner-2.2.jar*. The formed command is then executed to use the unclean HTML web page to generate the cleaned HTML web page and store it in the folder, *outputCleanedHTML*. If the batch file cleanHTML.bat doesn't exit this module creates one and then write the command and execute it.

Below is the format to use the htmlcleaner-2.2.jar with src and one argument and dest as the other argument to generate the cleaned HTML web page.

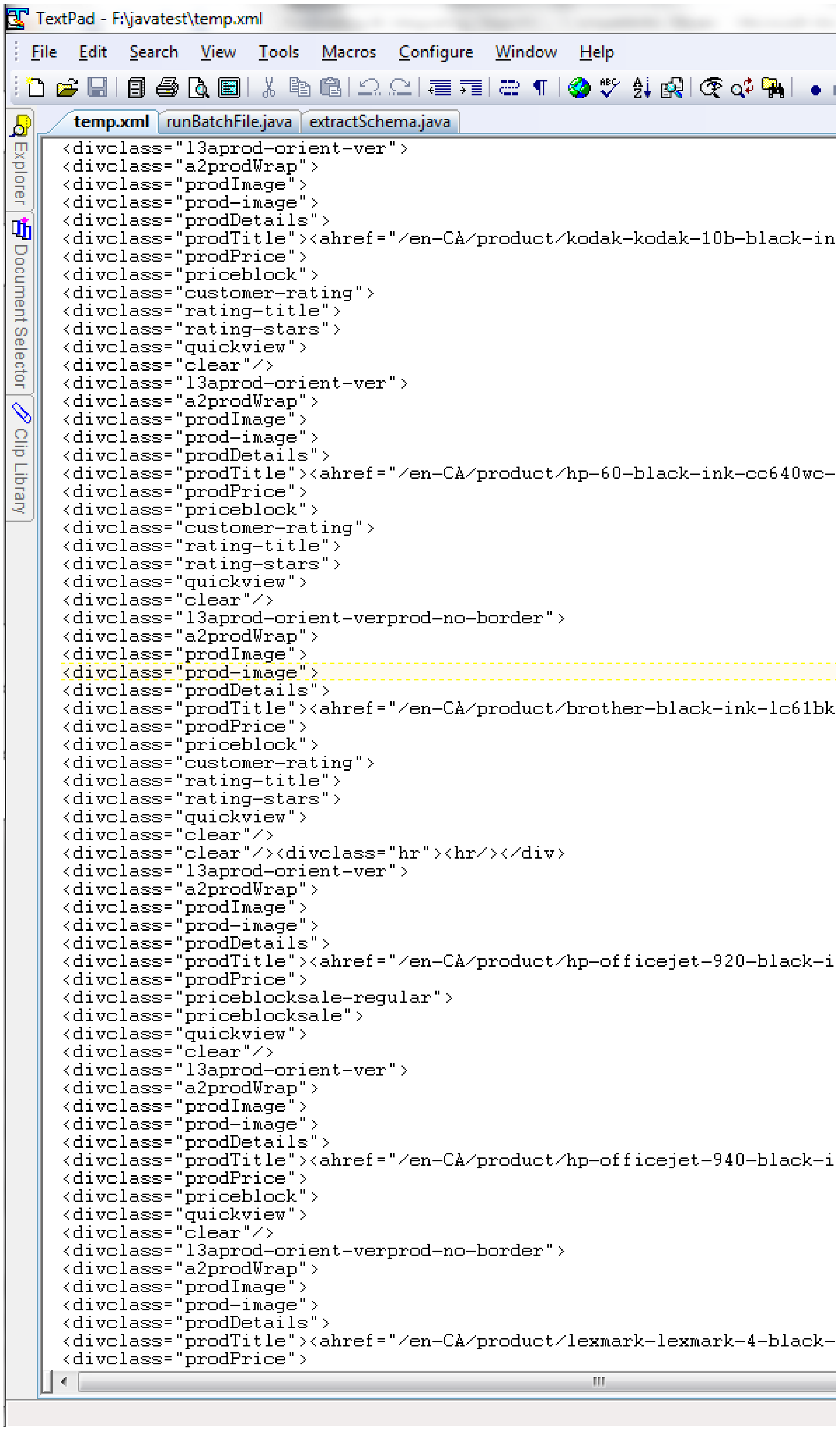
```
java -jar *\WebOMiner\htmlcleaner-2.2.jar src=business.html dest=business_.html
```

The results of the cleaned HTML are stored in a defined location which is later used by the Cleaned HTML parser module.

II. Cleaned HTML Parser

This module implements the parsing of the cleaned HTML which is obtained as the output of the previous module HTML Cleaner. This module is implemented in a java file namely *extractSchema.java* by a function called *parseTheHTML().* Object to the class *extractSchema* is created with the argument passed to the constructor of this class which is the name of the business. i.e. bestbuy, futureshop, etc. The parser uses the name of the business and finds the corresponding cleaned HTML in the folder *outputCleanedHTML* appending underscore to the business. The cleaned HTML is sent to the parser in turn. The parser does the following process

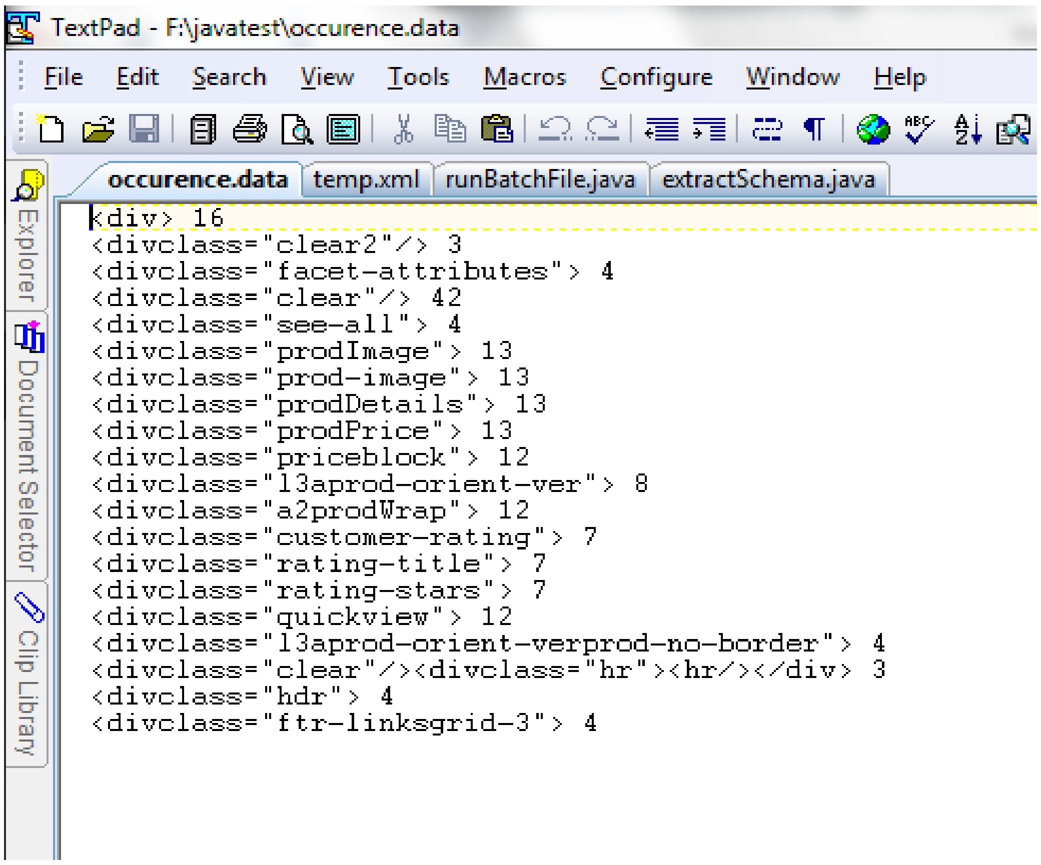
1. Receives the Cleaned HTML file
2. Creates a *temp.xml* file locally to store all the blocks in the web page given. i.e.<div>,<table>
3. Xml is stored removing all the attributes except class in every block as shown in the figure below.


TextPad - F:\javatest\temp.xml
File Edit Search View Tools Macros Configure Window Help
temp.xml runBatchFile.java extractSchema.java
Explorer Document Selector Clip Library

```
<divclass="l3aprod-orient-ver">
<divclass="a2prodWrap">
<divclass="prodImage">
<divclass="prod-image">
<divclass="prodDetails">
<divclass="prodTitle"><ahref="/en-CA/product/kodak-kodak-10b-black-in
<divclass="prodPrice">
<divclass="priceblock">
<divclass="customer-rating">
<divclass="rating-title">
<divclass="rating-stars">
<divclass="quickview">
<divclass="clear"/>
<divclass="l3aprod-orient-ver">
<divclass="a2prodWrap">
<divclass="prodImage">
<divclass="prod-image">
<divclass="prodDetails">
<divclass="prodTitle"><ahref="/en-CA/product/hp-60-black-ink-cc640wc-
<divclass="prodPrice">
<divclass="priceblock">
<divclass="customer-rating">
<divclass="rating-title">
<divclass="rating-stars">
<divclass="quickview">
<divclass="clear"/>
<divclass="l3aprod-orient-verprod-no-border">
<divclass="a2prodWrap">
<divclass="prodImage">
<divclass="prod-image">
<divclass="prodDetails">
<divclass="prodTitle"><ahref="/en-CA/product/brother-black-ink-lc61bk
<divclass="prodPrice">
<divclass="priceblock">
<divclass="customer-rating">
<divclass="rating-title">
<divclass="rating-stars">
<divclass="quickview">
<divclass="clear"/>
<divclass="clear"/><divclass="hr"><hr/></div>
<divclass="l3aprod-orient-ver">
<divclass="a2prodWrap">
<divclass="prodImage">
<divclass="prod-image">
<divclass="prodDetails">
<divclass="prodTitle"><ahref="/en-CA/product/hp-officejet-920-black-i
<divclass="prodPrice">
<divclass="priceblocksale-regular">
<divclass="priceblocksale">
<divclass="quickview">
<divclass="clear"/>
<divclass="l3aprod-orient-ver">
<divclass="a2prodWrap">
<divclass="prodImage">
<divclass="prod-image">
<divclass="prodDetails">
<divclass="prodTitle"><ahref="/en-CA/product/hp-officejet-940-black-i
<divclass="prodPrice">
<divclass="priceblock">
<divclass="quickview">
<divclass="clear"/>
<divclass="l3aprod-orient-verprod-no-border">
<divclass="a2prodWrap">
<divclass="prodImage">
<divclass="prod-image">
<divclass="prodDetails">
<divclass="prodTitle"><ahref="/en-CA/product/lexmark-lexmark-4-black-
<divclass="prodPrice">
```

Figure 1 sample temp.XML file

4. The file *temp.xml* is read and the frequency of the blocks and one with child nodes are noted in different file called *occurrence.data.*



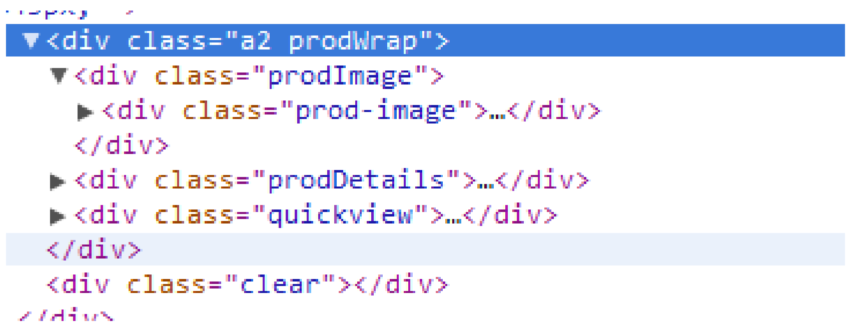
```
<div> 16
<divclass="clear2"/> 3
<divclass="facet-attributes"> 4
<divclass="clear"/> 42
<divclass="see-all"> 4
<divclass="prodImage"> 13
<divclass="prod-image"> 13
<divclass="prodDetails"> 13
<divclass="prodPrice"> 13
<divclass="priceblock"> 12
<divclass="l3aprod-orient-ver"> 8
<divclass="a2prodWrap"> 12
<divclass="customer-rating"> 7
<divclass="rating-title"> 7
<divclass="rating-stars"> 7
<divclass="quickview"> 12
<divclass="l3aprod-orient-verprod-no-border"> 4
<divclass="clear"/><divclass="hr"><hr/></div> 3
<divclass="hdr"> 4
<divclass="ftr-linksgrid-3"> 4
```

5. This *occurrence.data* file is analysed to get the parent node of the product block whose children will also be presented in the *occurrence.data* which is the block of focus.

III. Frequent pattern structure finder

This module analyzes the file *occurrence.data* created by the previous module. This file consist of </div>s and <table>s with the corresponding class name and the frequency of occurrence.

1. *DocumentBuilderFactory, DocumentBuilder* are used to create a DOM tree and stored in memory.
2. *XPath* is used to retrieve the complete node using the parent tag name and the class name.
3. Every *XML* retrieved out of the *XPath* with the tag name and class name are analyzed recursively for the structure and stored temporary.
4. The frequency of this structure all over the DOM tree is found and stored using the function *printFinalTree()*.
5. The maximum frequencies of the structure which has maximum number of children that are also with present in the *occurrence.data* are considered as the product block.
6. This block will have the information about the product which will be later used to derive the schema dynamically.



```
▼<div class="a2 prodWrap">
  ▼<div class="prodImage">
    ▶<div class="prod-image">…</div>
    </div>
  ▶<div class="prodDetails">…</div>
  ▶<div class="quickview">…</div>
 </div>
 <div class="clear"></div>
```

Figure 2 Frequent structure found in best buy web page

IV. Schema Extractor

This module is implemented in the constructor of *extractSchema* class. The product block which is found using the frequent pattern structure finder is the input to this module. The steps involved in extracting the schema from the product block are as follows

1. The product block *XML* is read using *XPath* and *NodeList* datatype.

2. Every tag in the product block is read to classify the data and their corresponding data type.

3. <img> tag will have the data of the type blob or image, <label> will have data type String etc.

4. The schema extracted for every single block is concatenated in the public string variable *webSchema* which is finally printed as output of the entire module integrated.

```
Discovered schema for a single page of bestbuyWebsite is below:
-----------------------------------------------------------------
Product(prod-image:  String    img300x300:  image  prodDetails:  String
prodTitle: String  prodPrice: String  priceblock: String  pricetitle:
String  shop-now: String  customer-rating: String  rating-title: String
rating-stars: String )
```

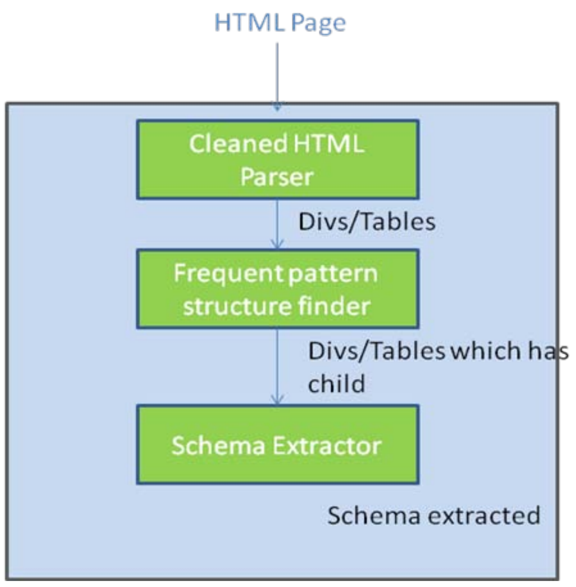
Figure 3 Extracted schema from best buy website

The extracted schema is displayed in the debug console for now which can be used to create a data table with this schema and finally a data ware house to store the data extracted from the B2C web sites. All the above mentioned modules are integrated in one class file with different functions. The extracted schema from the best buy product web page is given above.

```
1.  extractSchema(business)
2.  FOR ALL divTag in business_.html DO
3.      occurrence.data<-Write(divTag,divTag(Class),frequencyOfdivTag(Class))
4.  END FOR
5.  occurrence.data = parseTheHTML(business_.html)
6.  domTree = documentBuilder(business_.html)
7.  FOR ALL tag IN occurrence.data DO
8.      tagToxml = XPath(tag)
9.      IF tagToxml  IS CHILD structList_Node
10.         READ NEXT Node
11.     END IF
12.     IF structList CONTAINS tagToxml
13.         tagToxml IDCount  = tagToxml IDCount + 1
14.     ELSE
15.         structList = structList + tagToxml
16.     END IF
17. END FOR
18. FOR ALL product_block  IN structList DO
19.     IF product_block  < previous_ product_block
20.         product_block   = previous_ product_block
```

```
21.         END IF
22. END FOR
23. FOR ALL *tag* IN *product_block*
24.         IF tag = img
25.                 *Web_Schema* = *Web_Schema* +class(*img*) + blob
26.         ELSE IF tag = text
27.                 *Web_Schema* = *Web_Schema* +class(*text*) + varchar
28.         END IF
29. END FOR
30. RETURN *Web_Schema*
```

Block diagrammatic representation of the implemented modules are given below.

31. 

32. **Figure 4Architecture of schema extractor**

3. Empirical Evaluations

The integration of all the modules are built in together to form a software bundle which accepts the URL of the web site. The software processes the web content and generates the schema of the products. The experiment is done with 4 different web sites (bestbuy.com, futureshop.ca, compUSA.com, walmart.ca, shopping.com, dell.com) for empirical evaluation of our system using different page structures. Our system is implemented in Java programming language. We then run our system in 64-bit Windows 7 home operating system at Intel Due Core 2.26 GHz, 4.00 GB RAM hp machine for each of these mirror web sites for empirical evaluation of our Schema Extractor system. We use the standard precision and recall measures to evaluate the results of our system. Precision is measured as average in percentage for the number of correct data retrieved divided by the total number of data retrieved by the system. Recall is measured as average in percentage for the total number of correct data retrieved divided by the total number of existing data in the web document. The results of the retrieval by our Schema Extractor system is tabulated in table 02 below:

Website	Actual Schema			Extracted Schema			
	Field Count	String field count	image field count	Correct	Wrong	Missing	Irrelevant
www.bestbuy.ca	11	9	2	10	1	0	1
www.futureshop.ca	12	10	2	12	0	0	6
www.canadiantire.com	17	14	3	15	2	0	5
www.walmart.ca	15	12	3	12	1	0	2

Precision	81%
Recall	89%

Table 3 Experimental results showing extraction of schema from web pages.

4. Experimental Results

The purpose of our experiment is to measure the performance of schema extractor system for schema extraction. Table 04 shows small scale experimental results as performance measure for our schema extraction system. We have taken one page per web site for experiment and the numbers n column shows different types of data in those pages. The Total column shown total number of data records for each pages. For those pages schema extractor system is able to identify schema correctly. Very less wrong data record identification is observed and it makes sense because our system is not based on the prediction. It missed no information because it extracts the data from web pages from different websites. There are 14 irrelevant attributes which are generated in the extraction of schema.

We observed the reason for irrelevant attributes. All of those irrelevant are in List type data records and because of mixing object type in data tuple. Our definition of List data tuple is a set of <text> and there should be at least 3-pairs in the tuple to be qualified as List tuple. But those <image> and <text> and therefore did not satisfy any of the criteria.

5. Conclusion and Future work

This project report has the work to prepare the data for mining. I have developed 3 modules which do the process of finding the frequent tree structure pattern and the schema is extracted from the xml created from the frequent pattern. The experiment is done with 5 different web sites to show that this work is feasible and yields precision and recall are above 85%. Since this is a very first effort to mine web content data using object-oriented approach, we feel there is plenty of room for improvement and to open new thread. In this method the data type are not well defined and the analysis of the child tag would give the data type close to the original data.

USER MANUAL

Installing schema extractor software

This article explains the how to install the schema extractor in any machine that has Java Runtime Environment (JRE).

1. Locate the folder *extractSchema* in the project CD in the drive.

2. Copy the folder into any of the location in the local hard drive.

3. Use the following format to form the command to run extractSchema.class

```
F:\javatest>"C:\Program Files\Java\jdk1.7.0_03\bin\java.exe"  extractSchema bestbuy
```

   The above format has an argument to run the program. The argument is the name of the business which indicates the corresponding cleaned html file in the folder *outputCleanedHTML.*

4. If any new product HTML web page's schema has to be extracted, copy the unclean HTML web page into the *inputUncleanedHTML*

5. Repeat the step 3 and step 4 to get the extracted schema in the console.

Technical User Manual

This article explains the how to install the schema extractor project in any machine that has Java Development Kit (JDK). I have used eclipse to implement the schema extraction. Main function is located in the extractSchema.java. Run extractSchema.java as Java application as follows.

1. Locate the folder *extractSchema* in the project CD in the drive.

2. Copy the folder into any of the location in the local hard drive.

3. Import the project into the IDE

4. Make sure the location of *temp.xml* and *Occurence.data* mentioned in the file *extractSchema.java* are pointing to corresponding location in the hard drive.

5. Make sure the location of the *inputUncleanedHTML* and *outputcleanedHTML* mentioned in the extractSchema.java and runBatchFile.java are pointing to corresponding location in the hard drive.

6. Turn on debug mode by setting the variable *debugFlag* to true in *extractSchema.java*

7. Use the following format to form the command to compile *extractSchema.java*

```
F:\javatest>"C:\Program Files\Java\jdk1.7.0_03\bin\javac.exe"  extractSchema.java
```

8. Use the following format to form the command to run *extractSchema.class*

```
F:\javatest>"C:\Program Files\Java\jdk1.7.0_03\bin\java.exe"  extractSchema bestbuy
```

The above format has an argument to run the program. The argument is the name of the business which indicates the corresponding cleaned html file in the folder *outputCleanedHTML.*

9. If any new product HTML web page's schema has to be extracted, copy the unclean HTML web page into the *inputUncleanedHTML*

10. Repeat the step 3 and step 4 to get the extracted schema in the console.

11. This module can be added to Titas implementation of WebOMiner by copying *extractSchema.java* in the source and creating the object of *extractSchema* class in Main.java of WebOMiner project.

Reference:

[1] Web Mining: Information and Pattern Discovery on the World Wide Web R. Cooley, B. Mobasher, and J. Srivastava Department of Computer Science and Engineering.

[2] Chia-Hui Chang and Shao-Chen Lui. 2001. IEPAD: information extraction based on pattern discovery. In *Proceedings of the 10th international conference on World Wide Web* (WWW '01). ACM, New York, NY, USA, 681-688. OI=10.1145/371920.372182 http://doi.acm.org/10.1145/371920.372182

[3] Towards Comparative Web Content Mining using Object Oriented Model Titas Mutsuddy A Thesis Submitted to the Faculty of Graduate Studies through the School of Computer science.

[4] HtmlCleaner http://htmlcleaner.sourceforge.net/index.php

[5] Java programming http://www.java2s.com/