

University of Windsor

## Scholarship at UWindor

---

Electronic Theses and Dissertations

Theses, Dissertations, and Major Papers

---

2017

# Implementation and Web Mounting of the WebOMiner\_S Recommendation System

Amrutraj Alhad Chachad  
*University of Windsor*

Follow this and additional works at: <https://scholar.uwindsor.ca/etd>

---

### Recommended Citation

Chachad, Amrutraj Alhad, "Implementation and Web Mounting of the WebOMiner\_S Recommendation System" (2017). *Electronic Theses and Dissertations*. 5970.  
<https://scholar.uwindsor.ca/etd/5970>

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email ([scholarship@uwindsor.ca](mailto:scholarship@uwindsor.ca)) or by telephone at 519-253-3000ext. 3208.

# Implementation and Web Mounting of the WebOMiner\_S Recommendation System

By

**Amrutraj Chachad**

A Thesis  
Submitted to the Faculty of Graduate Studies  
through the School of **Computer Science**  
in Partial Fulfillment of the Requirements for  
the Degree of **Master of Science**  
at the University of Windsor

Windsor, Ontario, Canada

2017

© 2017 **Amrutraj Chachad**

# **Implementation and Web Mounting of the WebOMiner\_S Recommendation System**

By

**Amrutraj Chachad**

APPROVED BY:

---

**A. Sarker**

Department of Mathematics and Statistics

---

**Y. H. Tsin**

School of Computer Science

---

**C. I. Ezeife, Advisor**

School of Computer Science

May 9, 2017

## **DECLARATION OF ORIGINALITY**

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication. I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

## **ABSTRACT**

The ability to quickly extract information from a large amount of heterogeneous data available on the web from various Business to Consumer (B2C) or Ecommerce stores selling similar products (such as Laptops) for comparative querying and knowledge discovery remains a challenge because different web sites have different structures for their web data and web data are unstructured. For example: Find out the best and cheapest deal for Dell Laptop comparing BestBuy.ca and Amazon.com based on the following specification: Model: Inspiron 15 series, ram: 16gb, processor: i5, Hdd: 1 TB. The “WebOMiner” and “WebOMiner\_S” systems perform automatic extraction by first parsing web html source code into a document object model (DOM) tree before using some pattern mining techniques to discover heterogeneous data types (e.g. text, image, links, lists) so that product schemas are extracted and stored in a back-end data warehouse for querying and recommendation. Although a web interface application of this system needs to be developed to make it accessible for to all users on the web.

This thesis proposes a Web Recommendation System through Graphical User Interface, which is mounted readily on the web and is accessible to all users. It also performs integration of the web data consisting of all the product features such as Product model name, product description, market price subject to the retailer, etc. retained from the extraction process. Implementation is done using “Java server pages (JSP)” as the GUI designed in HTML, CSS, JavaScript and the framework used for this application is “Spring framework” which forms a bridge between the GUI and the data warehouse. SQL database is implemented to store the extracted product schemas for further integration, querying and knowledge discovery. All the technologies used are compatible with UNIX system for hosting the required application.

### **Keywords**

Web data mining, Data extraction and integration, Pattern Discovery, Wrapper Induction, Modeling Data Warehouse

## **DEDICATION**

This work is dedicated to my father Mr. Alhad Chachad and my mother Mrs. Dhanashri Chachad for their love, support and encouragement. Without their motivation and guidance, the completion of this work would not have been possible.

## **ACKNOWLEDGEMENT**

I would like to give my sincere appreciation to all of the people who have helped me throughout my education.

This thesis is dedicated to my supervisor Dr. C.I. Ezeife who has been a constant source of knowledge and information. I express my humble gratitude for her generosity to spend quality time to train her students to achieve their goals with sheer hard work.

I would also like to thank my external reader, Dr. Animesh Sarker, my internal reader, Dr. Yung H. Tsin, and my thesis committee chair, Dr. Ngon for making time to be in my thesis committee, reading the thesis and providing valuable input. I appreciate all your valuable suggestions and the time, which have helped improve the quality of this thesis.

Finally, I express my appreciations to all my friends and colleagues at University of Windsor, especially Bindu Peravali and Sravya Vangala for their support and encouragement. Thank you all!

## TABLE OF CONTENTS

DECLARATION OF ORIGINALITY .....	iii
ABSTRACT .....	iv
DEDICATION.....	v
ACKNOWLEDGEMENT .....	vi
TABLE OF FIGURES.....	ix
CHAPTER 1. INTRODUCTION:.....	1
1.1. <i>Web data Mining</i> .....	1
1.2. <i>Applications of Web data extraction systems:</i> .....	7
1.2.1. <i>Enterprise Application:</i> .....	8
1.2.2. <i>Social Applications:</i> .....	9
1.3. <i>Thesis Contributions</i> .....	10
1.3.1. <i>Feature Contribution</i> .....	10
1.3.2. <i>Method Contribution</i> .....	11
1.4. <i>Outline of Thesis:</i> .....	12
CHAPTER 2. RELATED WORK .....	13
2.1. <i>Existing Web Scrapers:</i> .....	13
2.2. <i>Related Systems:</i> .....	14
2.2.1. <i>IEPAD: Information Extraction Based On Pattern Discovery</i> .....	14
2.2.2. <i>STALKER: A Hierarchical Approach to Wrapper Induction</i> .....	17
2.2.3. <i>DEBYE: Data Extraction By Example</i> .....	19
2.2.4. <i>LIXTO: Visual Web Info Extraction</i> .....	21
2.2.5. <i>Simultaneous record detection and attribute labeling in web data extraction</i> ...	23
2.2.6. <i>WebOMiner: Towards Comparative Mining of Web documents using NFA</i> .....	25
2.2.6.1. <i>Major Limitations</i> .....	30



<i>2.2.7. WebOMiner_S:Comparative mining of B2C Websites.....</i>	<i>31</i>
<i>2.2.7.1. Major Limitations.....</i>	<i>39</i>
<b>CHAPTER 3. PROPOSED WebOMiner_SRec System .....</b>	<b>40</b>
<i>3.1. WebOMiner_SRec system: .....</i>	<i>40</i>
<i>3.1.1. Architecture of our WebOMiner_SRec System:.....</i>	<i>40</i>
<i>3.1.2. Algorithm for our WebOMiner_SRec System: .....</i>	<i>41</i>
<i>3.1.3. WebOMiner_SRec System modules and working: .....</i>	<i>43</i>
<i>3.1.3.1. Schema Mapper: .....</i>	<i>43</i>
<i>3.1.3.2. Data warehouse loader:.....</i>	<i>47</i>
<i>3.1.3.2.1. Data Access object class: .....</i>	<i>49</i>
<i>3.1.3.3. Query manager: .....</i>	<i>52</i>
<i>3.2. Possible contributions and developments: .....</i>	<i>58</i>
<b>CHAPTER 4. IMPLEMENTATION: .....</b>	<b>59</b>
<i>4.1. Where are the source code files:.....</i>	<i>59</i>
<i>4.1.1. Which OS is supported by the WebOMiner_SRec:.....</i>	<i>59</i>
<i>4.2. Installation and Working procedure:.....</i>	<i>60</i>
<i>4.2.1. Steps for Windows installation:.....</i>	<i>61</i>
<i>4.2.2. Steps for Unix installation:.....</i>	<i>62</i>
<i>4.3. How to use the WebOMiner_SRec system:.....</i>	<i>62</i>
<i>4.3. Comparing WebOMiner_SRec with other existing systems:.....</i>	<i>73</i>
<b>CHAPTER 5. CONCLUSION AND FUTURE WORK:.....</b>	<b>74</b>
<i>5.1. Future Work: .....</i>	<i>74</i>
<b>REFERENCES.....</b>	<b>76</b>
<b>APPENDIX - A .....</b>	<b>80</b>
<b>VITA AUCTORIS .....</b>	<b>90</b>

## TABLE OF FIGURES

<b>FIGURE 1: Example Input webpage.....</b>	<b>4</b>
<b>FIGURE 2: Expected output page.....</b>	<b>5</b>
<b>FIGURE 3: Output data extracted from Amazon.....</b>	<b>6</b>
<b>FIGURE 4: Googleshop interface .....</b>	<b>9</b>
<b>FIGURE 5: Web page with data regions and data blocks.....</b>	<b>26</b>
<b>FIGURE 6: WebOMiner Algorithm.....</b>	<b>27</b>
<b>FIGURE 7: Content object array for data block identification and classification.....</b>	<b>28</b>
<b>FIGURE 8: Architecture of the WebOMiner system.....</b>	<b>29</b>
<b>FIGURE 9: Experiments conducted on web data sources.....</b>	<b>30</b>
<b>FIGURE 10: Extracted schema class.....</b>	<b>34</b>
<b>FIGURE 11: Temp.xml file.....</b>	<b>35</b>
<b>FIGURE 12: Occurrence.data file.....</b>	<b>36</b>
<b>FIGURE 13: WebOMiner_SRec system architecture.....</b>	<b>40</b>
<b>FIGURE 14: Seller fact table for Datawarehouse.....</b>	<b>50</b>
<b>FIGURE 15: Product table for Datawarehouse.....</b>	<b>51</b>
<b>FIGURE 16: Create session.....</b>	<b>55</b>
<b>FIGURE 17: Hibernate configuration file.....</b>	<b>55</b>
<b>FIGURE 18: Hibernate mapping file.....</b>	<b>56</b>
<b>FIGURE 19: Input page to WebOMiner_SRec.....</b>	<b>63</b>
<b>FIGURE 20: Scraped data display result.....</b>	<b>64</b>
<b>FIGURE 21: Quick search results.....</b>	<b>65</b>
<b>FIGURE 22: Advanced search Interface.....</b>	<b>66</b>
<b>FIGURE 23: Advanced search for error free inputs.....</b>	<b>67</b>
<b>FIGURE 24: Advanced search results.....</b>	<b>68</b>
<b>FIGURE 25: Filter search results.....</b>	<b>69</b>
<b>FIGURE 26: Comparative search results.....</b>	<b>70</b>
<b>FIGURE 27: Historical search results.....</b>	<b>71</b>
<b>FIGURE 28: About page.....</b>	<b>72</b>

## CHAPTER 1: INTRODUCTION

Web data drives today's businesses and the Internet into a powerhouse of information. Most businesses rely on the web to gather data that is crucial to their decision-making processes. Thus, 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. Companies also, regularly assimilate and analyze product specifications, pricing information, market trends and regulatory information from various websites and when performed manually, this is often a time-consuming, error-prone process. World Wide Web (WWW) is growing exponentially over the years and the web documents constitute some of the largest repositories of information (Kosala and Blockeel 2000). The web poses great challenges for resource and knowledge discovery based on the following observations: (1) The web is too huge; (2) Complexity of Web pages; (3) Web is dynamic information source; (4) Diversity of user communities; (5) Relevancy of Information. Web content usually refers to the information that can be seen on a web document by an individual. It also includes some hidden information, which help users interact with web contents (Cooley et. al., 1997). Therefore, there is a huge need for a value-added service that can be used to integrate information from multiple sources.

### ***1.1. Web data Mining:***

Web mining aims to discover useful information or knowledge from the Web hyperlink structure, page content, and usage data (Zhou et. al., 2007). Although Web mining uses many data mining techniques, as mentioned above it is not purely an application of traditional data mining techniques due to the heterogeneity and semi-structured or unstructured nature of the Web data.

Many new mining tasks and algorithms were invented in the past decade. Based on the primary kind of data used in the mining process, Web mining tasks can be categorized into three types: Web structure mining, Web content mining and Web usage mining (Liu 2007).

Web structure mining: Web structure mining discovers useful knowledge from hyperlinks (or links for short), which represent the structure of the Web (Chakrabarti 2000). For example, from the links, we can discover important Web pages, which is a key technology used in search engines. We can also discover communities of users who share common interests. Traditional data mining does not perform such tasks because there is usually no link structure in a relational table.

Web content mining: Web content mining extracts or mines useful information or knowledge from Web page contents (Bharanipriya and Prasad 2011). For example, we can automatically classify and cluster Web pages according to their topics. These tasks are similar to those in traditional data mining. However, we can also discover patterns in Web pages to extract useful data such as descriptions of products, postings of forums, etc., for many purposes. Furthermore, we can mine customer reviews and forum postings to discover consumer opinions. These are not traditional data mining tasks.

Web usage mining (Srivastava et. al., 1997): Web usage mining refers to the discovery of user access patterns from Web usage logs, which record every click made by each user. Web usage mining applies many data mining algorithms. One of the key issues in Web usage mining is the pre-processing of clickstream data in usage logs in order to produce the right data for mining. However, due to the richness and diversity of information on the web, there are a large number of Web mining tasks.

The Web mining process is similar to the data mining process. The difference is usually in the data collection. In traditional data mining, the data is often already collected and stored in a data warehouse. For Web mining, data collection can be a substantial task, especially for Web structure and content mining, which involves crawling a large number of target Web pages.

***Example:***

Gathering customizable website information through robots/crawlers, comparison-shopping agents, meta-search engines and newsbots, etc. Web pages have hyperlinks and anchor texts, which do not exist in traditional documents. Hyperlinks are extremely important for search and play a central role in search ranking algorithms. Anchor texts associated with hyperlinks too are crucial because a piece of anchor text is often a more accurate description of the page than its hyperlink points. Second, Web pages are semi-structured (Hammer et.al., 1997). A Web page is not simply a few paragraphs of text like in a traditional document. A Web page has different fields, e.g., title, metadata, body, etc. The information contained in certain fields (e.g., the title field) is more important than in others. Furthermore, the content in a page is typically organized and presented in several structured blocks. Some blocks are important and some are not (e.g., advertisements, privacy policy, copyright notices, etc). Effectively detecting the main content block(s) of a Web page is useful to Web search because terms appearing in such blocks are more important.



Let us consider a live example of a web page taken from a product website having data blocks to display the existing product information in the web sellers inventory.

Figure 1 shows a Web page segment containing a list of Laptops on BestBuy (This is the source input HTML page). The description of each Laptop is a data record.

1 - 32 of 588

◀ Prev 1 2 3 4 5 ... 19 Next ▶

Sort: Best Match ▼



**\$899.99**



Asus ZenBook UX430  
14" Laptop - Grey (Intel  
Core i5-7200U/ 256GB  
SSD/ 8GB RAM /  
Windows 10)

★★★★★ (10)

☐ Sold out online

☐ Sold out in nearby  
stores

☐ Compare



**\$699.99**



ASUS X-Series 15.6"  
Laptop - Black (AMD  
Quad Core A10-9600P  
/ 1TB HDD / 12GB RAM  
/ Windows 10)

★★★★★ (53)

☒ Available online

☒ Available at nearby  
stores

☐ Compare



**\$749.99**


ASUS VivoBook X541  
15.6" Laptop - Black  
(Intel Core i5-7200U/  
1TB HDD/ 8GB RAM/  
Win 10) - Bilingual

★★★★★ (6)

☒ Available online

☒ Available at nearby  
stores

☐ Compare



**\$599.95**

Lenovo 15.6" Laptop -  
Black (AMD A12-9700P  
/ 1TB HDD / 12GB RAM  
/ Windows 10) - English

★★★★★ (8)

☒ Available online

☐ Sold out in nearby  
stores

☐ Compare

Name	Price	Seller	Time
HP 15.6" Touchscreen Laptop - Black (Intel Core i3-6100U/1 TB HDD/8 GB RAM/Windows 10 Home)	499.99	BestBuy	10-04-2017 15:20
ASUS VivoBook X541 15.6" Laptop - Black (Intel Core i5-7200U/ 1TB HDD/ 8GB RAM/ Win 10) - Bilingual	749.99	BestBuy	10-04-2017 15:20
Lenovo 15.6" Laptop - Black (AMD A12-9700P / 1TB HDD / 12GB RAM / Windows 10) - English	599.95	BestBuy	10-04-2017 15:20
ASUS X-Series 15.6" Laptop - Black (AMD Quad Core A10-9600P / 1TB HDD / 12GB RAM / Windows 10)	499.99	BestBuy	10-04-2017 15:20
HP 15.6" Touchscreen Laptop (AMD A9-9410 7th Generation/1 TB HDD/8 GB RAM/Windows 10 Home) - Black	699.99	BestBuy	10-04-2017 15:20
HP 15-AY138CA 15.6" Touchscreen Laptop - Silver (Intel Core i5-7200U/1TB HDD/8GB RAM/Windows 10)	799.99	BestBuy	10-04-2017 15:20
ASUS Zenbook UX330CA 13.3" Laptop (Intel Core m3-7Y30 / 256GB SSD / 8GB RAM / Windows 10) - Quartz Grey	899.99	BestBuy	10-04-2017 15:20

**Figure 2: Expected output**

Defined by the BestBuy input HTML page, the expected output gives a list of Laptops with the following description:

- Laptop Model: for eg. Asus Zenbook UX330CA
- Processor: Intel Core m3
- Memory: 256 GB
- RAM: 8GB
- Operating System: Windows 10
- Color: Quartz Grey
- Price: 899.00\$
- Deal: for eg. Black Friday deal, sale end date: 15<sup>th</sup> December 2016

Another example of a similar input page from Amazon:

Name	Price	Seller	Time
HP Stream 14" HD Laptop (Celeron N3060 Processor, 4GB RAM, 32GB Storage) with Windows 10 Home - Aqua Blue	299.99	amazon	03-04-2017 07:30
ASUS F555LA 15.6" Full-HD Laptop (Core i3, 4GB RAM, 500GB HDD) with Windows 10	529.99	amazon	03-04-2017 07:30
Acer Aspire One Cloudbook 11.6" Laptop (Intel N3050 2GB RAM, 32GB SSD) with Windows 10 and Office 365, Bilingual...	289.99	amazon	03-04-2017 07:30
Acer Aspire One Cloudbook 14" Laptop (Intel N3050, 2GB RAM, 64GB SSD) with Windows 10 and Office 365, Bilingual...	299.0	amazon	03-04-2017 07:30

**Figure 3: Output Data extracted from Amazon**

In this example, we observe that along with Best Buy, Amazon also has a list of laptops that have similar kind of product description. This is the main reason for developing a system that can produce comparative queries in order to obtain such kind of result: Find the cheapest Acer laptop which has a processor of Intel i5 generation, a HDD with at least 512 GB memory space and an operating system which supports at least Windows 10 application. Additional features can be graphic card description or the weight restrictions of the laptop.

***Types of pages used for extraction:***

These input pages are called as Structured web pages. These are the type of pages that follows a defined structure, which is attained due to the structured data source like a database or table. While data on the webpage is being published it will have a formatted style. A product rich page from an e-commerce website like bestbuy.ca is the example of structured web page since all the information such as product details are essentially stored



in a database which are published on to a webpage following a strict template. Such data can be extracted using wrappers by generating the extraction rules manually or automatically based on the target data. In a structured web page there are two classifications, the List Page and the detail page (Chang et. al., 2006).

We have chosen to mine a list page instead of a detail page because of the following reasons:

1. List pages provide the most important attributes of the object in the customers perspective whereas the detail pages contains all the attributes of the object. As our main objective is to develop a system that can perform a comparative mining and historical querying as value added service for the customers.
2. Our observation is that the detail page contains more noise than the list page, on Considering the recommendations and advertisements. As we work towards eliminating noisy data in our data warehouse we prefer to mine less noisy web pages.

It is very resource consuming to extract data from the list page then move to another description page for data extraction as the list page has a list of products and each product link has a pointer to a description page. Detail web page displays only one object with all of its attributes (Annoni and Ezeife 2009). The object is a Laptop and the detail page displays all the additional attributes of this object such as name of the laptop, price, configuration, dimensions and weight, build quality, etc. All this can be extracted to store into a data warehouse for value added services.

### ***1.2. Applications of Web data extraction systems:***

For web data extraction systems, many works cover techniques to solve some particular problems related to a single or, sometimes, a couple of fields of application. The aim of this activity is to analyze the greatest possible number of applications that are strictly related to Web data extraction tasks. In the following, we describe a taxonomy in which key application fields are divided into two sections namely, enterprise applications and social applications.

### ***1.2.1. Enterprise Application:***

- **Context advertising:** The main underlying principle is to present, to the user, commercial advertisements together with the content of the Web page, ensuring a potential increase of the interest in the ad. This aim can be reached by analyzing the semantic content of the page, extracting relevant information, both in the structure and in the data, and then contextualizing the ads content and placement in the same page. Contextual advertising, compared to the old concept of Web advertising, represents an intelligent approach to provide useful information to the user, statistically more interested in the ads, and a better source of income for advertisers.
- **Customer Care:** Usually medium/big sized companies with customers support receives a lot of unstructured information like emails, support forum discussions, documentation, shipment address information, credit card transfer reports, phone conversation transcripts, etc. The capability of extracting this information simplifies the way of classifying them, analyzing relationships, populating own structured databases and ontologies, etc.
- **Database building:** In the Web marketing sector this can be defined as the activity of building a database about a particular domain. Fields of application are countless. Financial companies could be interested in extracting financial data from the Web, e.g. scheduling the activities to be executed automatically and periodically.

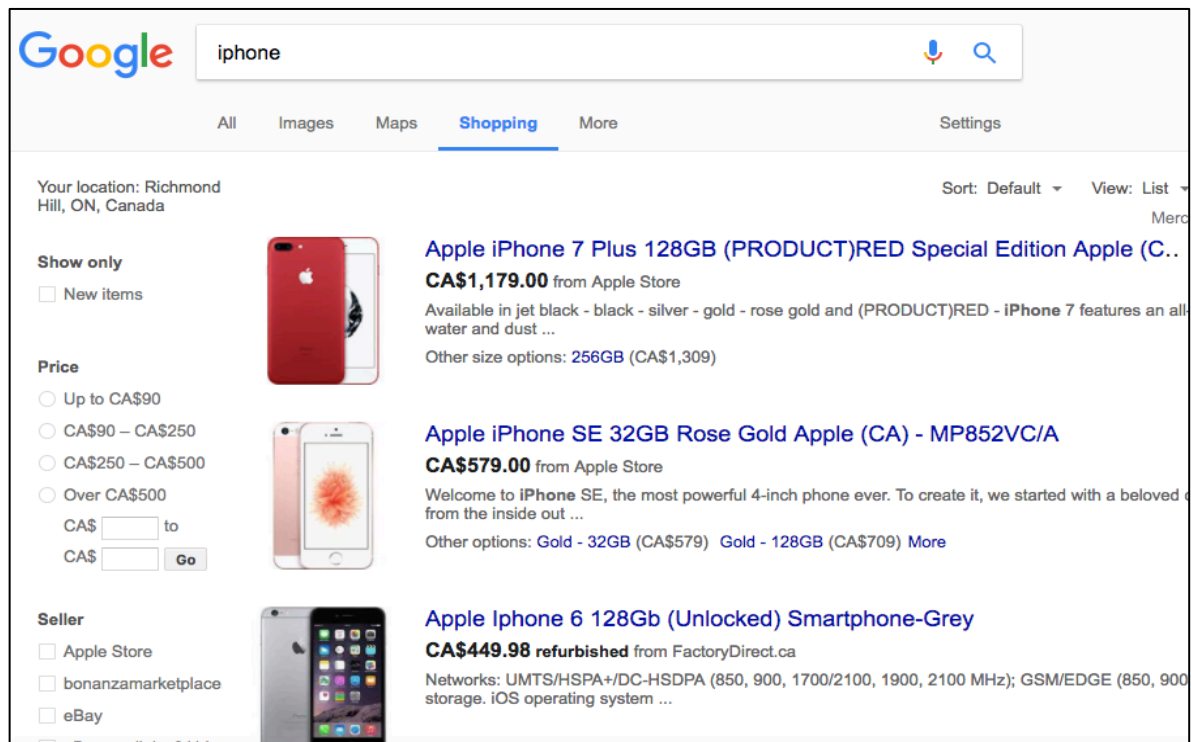
Companies selling products or services probably want to compare their pricing with other competitors. Other related tasks, obviously involved in the Web data extraction are: duplicating an on-line database, extracting dating sites information, acquiring job postings from job sites, etc.

### ***1.2.2. Social Applications:***

- **Social Networks:** Social networks attract attention of many industries for work done on extracting relevant data from social networks. This is a new interesting problem and field of application for Web data extraction systems.

- **Comparison Shopping:** One of the most appreciated Web social services is the comparison shopping, through platforms with the capability to compare products or services, going from simple price comparison to features comparison, technical sheets comparison, user experiences comparison, etc. These services heavily rely on Web data extraction, using websites as sources for data mining to make possible the comparison of similar items. There are a lot of comparison-shopping systems available on the Internet, which is also known as web data extraction systems. For example Googleshop, NextTag, ShopZilla, etc.

Figure 4 shows an example working of the Googleshop web data extraction application available for comparison-shopping:



**Figure 4: Example Googleshop application**

### ***1.3. Thesis Contributions:***

#### ***1.3.1. Feature Contribution:***

1. Extending the WebOMiner\_S data extraction system with a Web GUI for querying extracted E-Commerce data from multiple data sources through UNIX server for web access and availability.

2. Extending the WebOMiner\_S data extraction system such that the extracted product schemas previously extracted from the web, which were stored in the database as generic schema and attribute name like Product1 (column1, column2, Column3, column ...n) are now assigned more meaningful product table and attribute names such as laptop (price, model, cpu, etc.).

Secondly, the old system stored schemas from different websites eg. (Staples, BestBuy, etc) as separate tables like product1(column1, column2, column ...n) and product 2 (column 1, column 2, ... column n) making comparative querying difficult. This thesis renames extracted tables and attributes and also integrates them into one product data-warehouse table called in the database for comparative and historical querying.

3. Extending the WebOMiner\_S data extraction system such that it can directly answer comparative historical queries on the web through the GUI in much the same way as existing recommendation systems on the web such as Googleshop and ShopZilla do.

### ***1.3.2. Method Contribution:***

To achieve the goals of our system, thesis proposes the system called the WebOMiner\_SRec, which accepts user queries about comparative prices and features of retail E-Commerce store products as well as previously extracted product schema and their data values from E-Commerce websites. It outputs through the GUI, the results on product and pricing data drawn from stored accumulated multiple data source (e.g. Staples, BestBuy) product information into data warehouse. Input is Product1 (Column 1, Column 2, ... Column n) and the Output is Laptop (product name, price, etc.). The WebOMiner\_SRec system does this through the following sequence of steps:

1. Query manager algorithm is devised to process the flow of the entire application. It works as follows:
  - (i) It creates an environment to run the WebOminer\_SRec on the web using miner server (Debian server in Unix environment) using Maven plugin (Maven scripts help to run the java application through Unix terminal).
  - (ii) JDBC connection class ensures a secure connection to the data-warehouse by calling Servlet() methods in Java for data-warehouse connection parameters.
  - (iii) After the connection is done successfully queries are made to fetch desired results that are encapsulated in the form of java objects and rendered back to the view interface as “ResultList” data object.
2. Schema Mapper function creates a feature and column element such as (Dell Laptop, product name). Then it maps the feature to its respective columns using:
  - (i) Schema Mapper generates mapping rules based on the data type, for example: an explicit rule is defined for each of the column attributes specified in the data-warehouse product table. These rules check for data type: as <Integer> with symbols like “\$” to be placed in the “price” column element. Accordingly, all columns have their respective rules scripted for appropriate data value matching.
  - (ii) Once all the feature elements are mapped to their respective column elements, the load file method defined by the data-warehouse loader is used to transport data into the data-warehouse.

3. Data-warehouse loader accepts the renamed product schema e.g. Laptop (product name, price, ...) from step 1 to store it in an already existing data warehouse product table. It also includes one dimension table for the seller called "Seller table" that stores information about the sellers. The third file is the Load file element to transport data from the source database to the data warehouse. This is done as follows:
  - (i) Seller\_fact table contains foreign key attribute about the seller information such as (seller name, ID, time/date, seller url, ...) from the product table that helps to categorize and maintain information based on the sellers/vendors.
  - (ii) A <load file> method is called that contains the data and feature to column mapping file that loads entire data or the amount of data specified by the buffer size of the Load file directly into the already existing product table in the data-warehouse.

#### ***1.4. Outline of Thesis:***

Chapter 2: A survey of related articles and existing similar systems specifying the architecture, algorithms and major limitations associated with these related systems.

Chapter 3: Discusses the proposed WebOMiner\_SRec system and the advantages it has over the other related systems. It defines the architecture, algorithm and the system requirements to execute this model.

Chapter 4: Discusses the major contributions, developments and implementation for our proposed WebOMiner\_SRec system.

Chapter 5: Provides conclusion and future work for our approach.

Chapter 6: Provides a list of References.

## **APPENDIX – A**

## **VITA AUCTORIS**

## CHAPTER 2: RELATED WORK

Web data extraction system is defined as a software automatically and repeatedly extracting data from Web pages with changing contents and delivers extracted data to a database or some other application (Baumgartner et al., 2009). It introduces three important aspects:

- Automation and scheduling: The automation of page access where creating many instances of the same task to handle asynchronous update changes is important. Scheduling is also important, e.g. if a user needs to grab data from a news website, the scheduler is necessary to execute scripts automatically and periodically.
- Data transformation: The steps between data extraction and data delivery such as data cleaning are captured in the process of data transformation. Here, the main objective is to fetch a unique homogenous structure for the data (Rahm & Do, 2000).
- Use of the extracted data: After the data is extracted and parceled in the required format, It is now represented as structured data and is now ready to be stored in any form of DBMS, warehouse or CMS management. This storage is necessary to either structure the data or for the purpose of statistical analysis (Berthold & Hand, 1999).

### ***2.1. Existing web data extraction systems available on the web:***

There are several systems and applications available on the web that perform web data extraction and compare product results. Examples of such systems are:

1. Google Shop (Best Application available on the web)
2. NextTag (One of the most popular depending on viewers choice)
3. Price-Grabber
4. Shopping.com

## **2.2. Related System:**

### **2.2.1 IEPAD: Information Extraction Based On Pattern Discovery**

(Chang & Lui, IEPAD: information extraction based on pattern discovery, 2001)

IEPAD is the semi-supervised learning system, which extracts data from web without using some manual methods such as labeling the Sample pages. It highlights the usage of repetitive patterns occurred in web pages in order to present the target data as a structure called PAT trees. And, multiple sequence alignment is used to comprehend all record instances.

The Major contributions of the paper include 1) Identifies the target data record without any human intervention by examining the fact that data records occur as repetitive patterns in a web document and data is pushed into web pages from underlying databases using a common template. 2) Uses a data structure called PAT trees a Binary suffix tree, to discover repetitive patterns as they only record exact match for suffixes. 3) Uses multiple sequence alignment by the center star method, which begins from each occurrence of a repeat and ends before the beginning of next occurrence.

The Architecture of the system has 3 modules first is an Extraction rule generator, the pattern viewer and the Extractor module. Where the first module analyses the html tag structure and generates rules, the second module displays the rules to the uses to select one rule among them to perform extraction and the third module extracts all the data records that match with the rule selected. Their algorithm is explained with a running example as follows

Example 1: For Instance consider this 2 lines of html code as input for the system in their first step.

```
<B>Congo</B>< I>242</I><BR>  
<B>Egypt</B><I>342</I><BR>$
```

The HTML page is given as an input to this component it then translates into a string of abstract representations called tokens, which is identified by a binary code. Here tokens are divided into two types, tag tokens and text tokens where tag tokens are the html tags and text tokens are the original text in between the tags denoted by Text().

Discovering patterns based on tag tokens majorly block level tags excluding text tags will result in more abstract patterns that represent the whole web page.



Example 2: Token string generated by the translator for the above example is as follows

Html(<B>)Text(\_)Html(</B>)Html(<I>)Text(\_)Html(<BR>)

Html(<B>)Text(\_)Html(</B>)Html(<I>)Text(\_)Html(<BR>)\$

These as encoded into binary strings as shown below

Html (<B>) 000, Html (</B>) 001, Html (<I>) 011 then the pattern would be 00000101.

The binary pattern, which is the output of the previous step, is given as input to this step. A

Patrica tree is a specification of binary suffix tree where 0 goes to left and 1 goes to right.

Here, each internal node is an indication of which bit is to be used for branching. Every

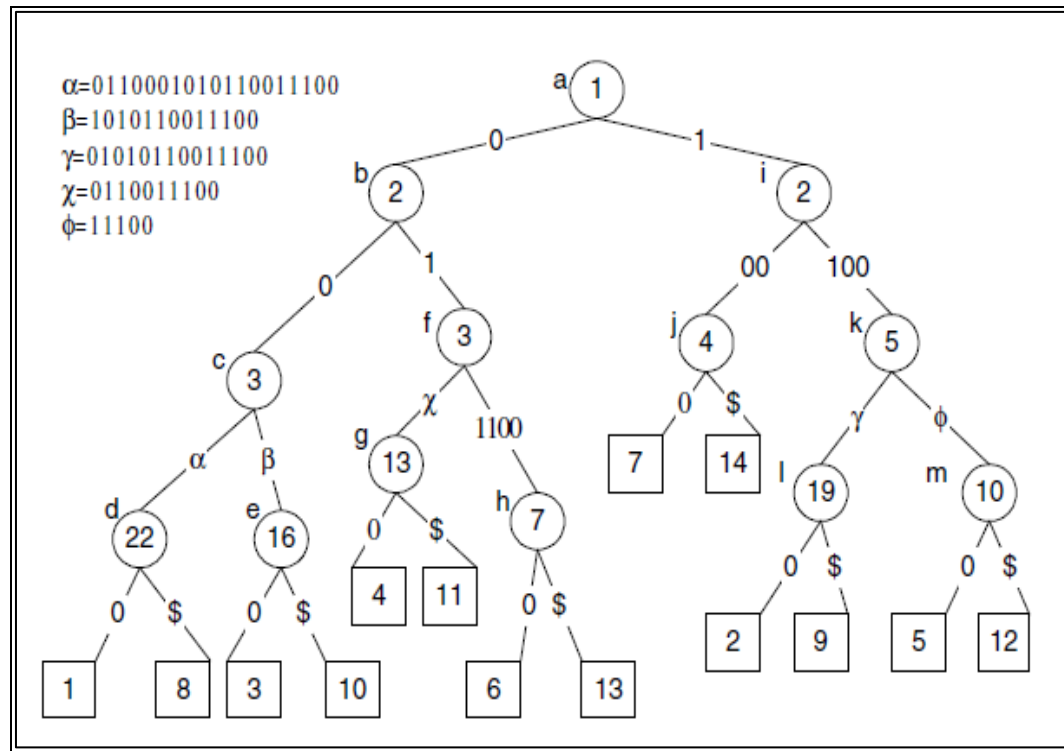
sequence of bits starting from each of the encoded token extending it to the end of the token

string. Each string is represented by a bit position. An important observation in pat tree is

that, all suffix strings with the same prefix will be under the same sub tree. Also, each edge

is associated with a virtual edge label, which is the string between those two nodes. We can

uniquely identify a string by its prefix.



Every Path label of the internal node represents a repeated sequence of inputs. Hence to discover the repeated patterns we now only have to observe the path labels. Thus the problem of identifying data records is narrowed to fetching repeated patterns in IEPAD.

For a path label in the internal node  $v$  if its sub tree is having different left characters then it is called Left\_Diverse. Based on this a lemma is proposed that maximum repeat should be a left diverse. The number of times any pattern has occurred can be easily known by the occurrence counts and the reference positions. Hence the user has to define a threshold value or it can even be a default value. Regularity, compactness and coverage are defined as the standard deviation of the interval between two adjacent occurrences is calculated as regularity. Whereas compactness is a measure of the density of a maximal repeat, this is used to eliminate maximal repeats that are scattered apart beyond a given bond. Along with these, Coverage measures the volume of content in the maximal repeats. IEPAD uses multiple string alignment for pattern matching; it does not allow any partial or inappropriate matching only strings exactly matched are considered. For Example, Suppose “kjl” is the pattern discovered for token string, if we have multiple alignments kjlmn, kjlm, kjlmo the extraction pattern can be generalized as “kjl[m\X]o[p\\_]” where “ - “ represents missing characters. Thus the problem is transformed to find the multiple alignments of the  $k$  strings. The approximation algorithm is used to find the center string  $S_c$  in  $K$  strings. Once the center string is found, each string is iteratively aligned to the center string to construct multiple alignments, which is in turn used to construct the extraction pattern.

K	J	L	M	O	P
K	J	L	X	O	P
K	J	L	X	O	-

The extraction process starts when the user selects one or more patterns from the pattern viewer. The extractor searches the PAT tree for maximum occurrences to find the target data but if the web page cannot be expressed as PAT tree then a normal pattern-matching algorithm can be used for extraction. Their results show 97% accuracy in successfully identifying the target data. However the major Short comings of the paper includes the following

- 1) The assumption that multiple data records will be homogenous fails as data records in web pages can have different attributes because of the missing values.
- 2) Requires a trained user that can infer rules to extract actual data records.
- 3) Their assumption that target data will be under the same parent node fails for large number of web pages, as there is no necessity in HTML5 after using the block tags to section web pages.
- 4) Does not propose any strategy to overcome noise or unnecessary data.

- 5) Their use of binary suffix trees only produces exact matches is a huge drawback because of the missing attribute values in data records.
- 6) Did not discuss about the heterogeneity of the web data such as the Images and their extraction process.

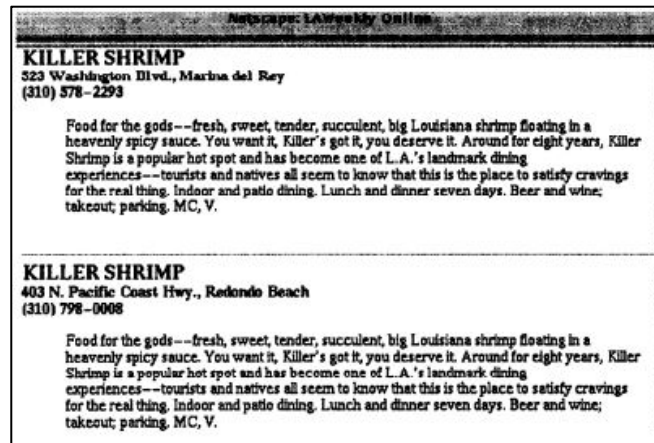
### ***2.2.2 STALKER: A Hierarchical Approach to Wrapper Induction***

( Muslea, Minton, & Knoblock, A hierarchical approach to wrapper induction, 1999)

This is a supervised approach for data extraction, where the user labels some sample pages from which the system generates the extraction rules. There by data in web pages having similar structure can be extracted by matching those rules. It performs a hierarchical data extraction by using embedded catalog formalism. The Major Contributions of the paper includes 1) Introducing the concept of hierarchical data extraction using embedded catalogue tree in which leaves are the attributes and internal node are the tuples. 2) web content can be extracted by some consecutive tokens called landmarks that enable the wrapper to locate data x with in a data region p. 3) Multiple scans are performed to handle missing values and multiple permutations. The Stalker proposed to represent a web page as typical embedded catalogue tree where each leaf node contains the target data and the internal nodes form the list of K tuples. Again the items in k tuples can be a leaf node l or another list L. Each edge of the tree is associated with an extraction rule in order to extract x from p.

For example consider the Restaurant description as given below

- 1:<p> Name: <b> Yala </b><p> Cuisine: Thai <p><i>
- 2:4000 Colfax, Phoenix, AZ 85258 (602) 508-1570
- 3:</i> <br> <i>523 Vernon, Las Vegas, NV 89104 (702) 578-2293
- 4:</i> <br> <i>403 Pica, LA, CA 90007 (213) 798-0008</i>



The stalker inductive algorithm generates extraction rules to capture each data item by taking as input the set of sequence of prefix tokens. User identifies the start of  $x$  from the sample page using the GUI. For instance consider that the user has marked the area codes then the sequential covering algorithm starts by generating linear  $l$ , as it then covers all the examples and generates disjuncts. In the fore mentioned example algorithm initially generates SkipTo() it matches  $E_2$  and  $E_4$  but does not match  $E_1$ ,  $E_3$ . While in the second iteration algorithm considers only uncovered examples and generates SkipTo (phone) and SkipTo(<b>). The figure represents the slg that accepts both the rules. Thus Stalker tries to generate slg that accepts all the positive examples and reject all the negative ones. The algorithm contains a function called learnDisjuncts () generates all the candidates repeatedly selects and refines the best candidate until it finds a perfect disjunct which only accepts the positive pages. In order to find best disjunct stalker searches for a disjunct that accepts the largest number of positive examples. Another function Refine () obtains netter disjuncts() by making its landmarks more specific or by adding new states to the automata. Followed by the GetTokens() which gives out the tokens that have appeared at least once. Their experimentation proves that their methodology achieves 97% accuracy. But Shortcomings of the paper are 1) their algorithm requires Multiple passes over the input data to extract the target data. 2) The performance of the algorithm entirely depends on the sample training pages given, making the system dependent on a trained user. 3) Their Technique is scientifically complex and time consuming that it cannot be used in real time applications. 4) Maintaining wrapper is another huge dis-advantage of this system; every time new presentation structure arrives the wrapper has to be updated.

### 2.2.3 DEBYE: Data Extraction By Example

(Laender, Ribeiro-Neto, & da Silva, DEByE—data extraction by example, 2002)

A supervised learning system where user marks example pages there by the structure is learnt from the pages through patterns and from it structures can be extracted by comparing the new data records with previous one. The Major contributions of the paper involve 1) this paper proposes Object oriented content extraction without loss of relationship between the data items. 2) It extracts atomic components first and then it assembles them into objects. 3) Uses pattern matching to identify the similar tokens for extraction.

Debye has 2 modules The Graphical interphase and The Extractor. An object is denoted as a pair  $O = \langle T, V \rangle$  such that  $V \in \text{domain}(T)$  where  $V$  is a type of  $O$ . thus  $O$  is an instance of  $T$ .  $\text{domain}(T) = \{a_1, a_2, a_3, \dots, a_n\} (n \geq 1)$ . Instances of A-types are termed as atoms. The pair  $\langle T, a_i \rangle$  is termed as attribute value pair  $_{User}$  has to highlight the target data from the web page using the GUI. This target data is treated as objects. Such assembled objects are used to generate *oe\_patterns* in xml. These patterns are used to extract data from the new web pages. The initial and final positions of occurrence of data are indicated by *ipos*, *fpos* respectively. Attribute value pair patterns are described as  $\langle T, a_i \rangle$ , a local syntactic context is associated to each avp it is the tokens surrounded by avp. Using this context information avp patterns are built. In the given example in order to identify avp pattern for type price would be as expressed in (a) that the value should be prefixed by  $S_{pre} = \text{"A deadly lie--\$"} in particular and suffixed by$

$S_{suf} = \text{"; dead online"}$ . So it strictly extracts only specific values. A more generalized pattern is indicated by (b) it denotes that price value is prefixed by "\$" and followed by ";" and "\*" indicates any value in that place.

```

<TUPLE type="Author">
  <ATOM type="Name">
    <VALUE ipos="2058" fpos="2073">Agatha Christie</VALUE>
  </ATOM>
  <LIST type="Book">
    <TUPLE type="Book">
      <ATOM type="Title">
        <VALUE ipos="2084" fpos="2122">The Adventure of the Christmas Pudding</VALUE>
      </ATOM>
      <ATOM type="Price">
        <VALUE ipos="2131" fpos="2135">5.95</VALUE>
      </ATOM>
    </TUPLE>
    <TUPLE type="Book">
      <ATOM type="Title">
        <VALUE ipos="2257" fpos="2280">The Hound of Death</VALUE>
      </ATOM>
      <ATOM type="Price">
        <VALUE ipos="2289" fpos="2293">8.95</VALUE>
      </ATOM>
    </TUPLE>
  </LIST>
</TUPLE>

```

Malcolm Hamer--GOLF MYSTERIES!--  
A Deadly Lie--\$10.95; Dead on Line--\$13.95;  
Shadows on the Green--\$10.95; Sudden Death--\$9.95;

Fig. 6. A sample text.

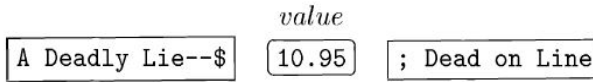


Fig. 7. Example of an AVP context.

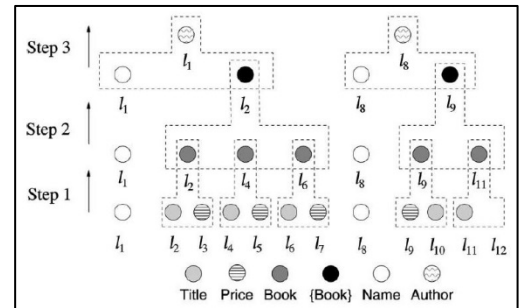
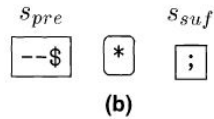
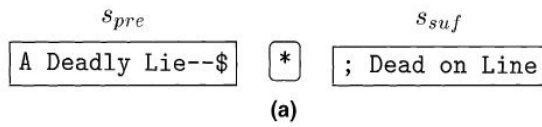


Figure 4: The execution of  
bottom up

OE patterns represent the structure of the data that the user needs to extract, its internal nodes form another OE tree or type nodes (like price, book) and leaves are avp patterns. There are two extraction strategies specified, the bottom up and the top down strategy. The Top down strategy is a straight forward, the oe patterns are assembled for each example object and using these patterns to discover new objects. This procedure is repeated based on the number of example pages provided. Here the whole object is demolished into components thus called a top down approach. This approach does not work well if the page is variable structure.

In the Bottom up strategy the avp's are priory recognized and extracted. For this approach objects are represented as triple  $O = \langle T, V, L \rangle$  where  $t$  is the type of the object,  $v$  is the value and  $l$  is the index. The last row of circles indicates the avps. Each is avp is associated with a label  $l_i$ . Contiguous pairs of title, price are formed as book instances. The values in  $L_9$  are in reverse order and  $L_{11}$  does not have the price component, the bottom up strategy can handle such cases. The list of objects assembled after the first step is indicated in 2<sup>nd</sup> row. Further the book instances are grouped up as list of book instances. From the example  $l_1$  is combined with  $l_2$  and  $l_8$  is combined with  $l_9$ . Thus the data can be extracted from similar pages. Major Short Comings are 1) Extraction precision falls low in case of missing attributes or multi-ordered attributes. 2) Multiple scan of input page is needed to extend each atomic attribute.

#### ***2.2.4 LIXTO: Visual Web Info Extraction:***

(Baumgartner, Flesca, & Gottlob, 2001)

Lixto is an interactive tool for data extraction built with logic based language called Elog. Elog uses a data log like logical syntax and semantics. Which translates relevant html strings into XML. The LIXTO system provides a user interface to select the data of user's interest. Thus selected data is translated into xml. There is no necessity for the user to have an idea over ELOG the underlying language. Elog is wrapper-programming language; it generates wrappers for data extraction by taking input as the user selected data from the web page. It is a collection of data-log like rules containing special extraction conditions in their bodies. Once a wrapper is generated it can be automatically extract relevant information from a permanently changing page. The lixto system associates the user-

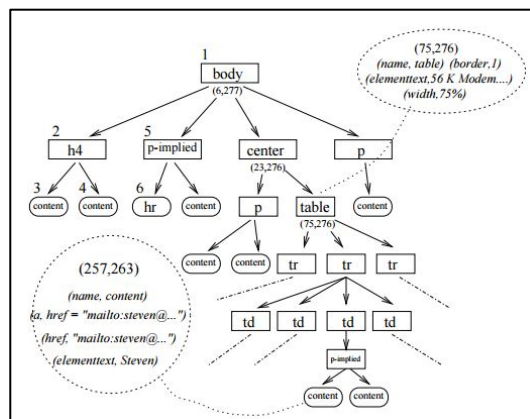
selected instance with a generalized tree path in the HTML parse tree, there by identifies similar rules. For example, user selects pattern <ITEM> followed by sub pattern <PRICE>, this sub pattern relationship expresses that an instance of <PRICE> should occur within a pattern <ITEM>. Then the system searches for such occurred patterns. User can also add conditions like after/before/ not after/range etc. Setting such filters can fetch perfectly desired information. Pattern creation is done by User explicitly providing the patterns their parents, can specify any attributes required then highlights all the objects in the provided example already. For example if the user selects a <TABLE> as a pattern, LIXTO searches for the occurrence this tag in the entire documents also highlights the <TR> tags. A rule is constructed for generating a rule to extract all the target data items. The patterns generated along with the hierarchical relationships are translated to XML by using their names as default <XML> tags. Extraction language Mechanisms is as follows, Head predicates are defined in Elog as record (S, X) where S indicates the parent pattern for instance <TABLE> in our running example. X looks for the sub elements that qualify as S. Thus defined head predicates are the patterns that Elog defines and extracts data according to the web page. Lixto offers two mechanisms of data extraction- string extraction and tree extraction. In the tree their tree paths identify extraction mechanism elements. An example of tree path can be \*.table.\*.tr here \* indicates can be called a wildcard, it indicates anything can occur. For correctly extracting only target data, attribute conditions can be applied. An attribute condition is a triple specifying a required name, value and a corresponding regular expression. The second extraction method is based on strings. In the parse tree leaf nodes indicate the strings. Regular expressions are used to capture them. Elog atoms correspond to special predicates with well-defined semantics as follows, In Elog the function mapping a given source S to a set of elements matching an *epd* is treated as relation subelem (S, epd, X). Subelem (s, epd, x) evaluates to true iff s is a tree region, *epd* is an element path definition and x is a tree region contained in s where the root of x matches *epd*. The extraction definition predicates specify a set of Extraction instances, the context condition predicates specify that some other sub tree or text must (not) appear after / before a desired extraction pattern. For example, on a page with several tables, the final table could be identified by an external condition stating that no table appears after the desired table. The internal condition predicates are semantic conditions like isCountry(x) or Date(x). Another



class of predicates are the pattern predicates that indicates to patterns and their relationship their parents. A standard rule in Elog is as quoted here  $\text{New}(S, X) \leftarrow \text{Par}(-, S), \text{Ex}(S, X), \text{Co}(S, X, \dots)[a, b]$ , where  $S$  is the parent instance variable,  $X$  is the pattern instance variable,  $\text{Ex}(S, X)$  is an extraction definition atom, and the optional  $\text{Co}(S, X)$  are further imposed conditions. A set of such rules forms a pattern. All the rules in a pattern uses the same information like the name of the pattern, its parent etc. Such patterns form an extraction program, which can be applied to similar pages. For Example, A wrapper for the above quoted example every item is stored in its own table extracted by `<record>`, all such patterns are defined in the same record. The price attribute uses concept attribute, namely `isCurrency` which matches with the string like `$` and the bids pattern uses a reference to price. The final two patterns are strings.

### 2.2.5 Simultaneous record detection and attribute labeling in web data extraction

This paper proposes a method called hierarchical conditional labeling, A novel graphical approach to mutually optimize record recognition and attribute naming. Major contributions of the paper involve 1) This approach extracts data from list web pages as well as detail web pages. 2) Uses a vision based tree construction mechanism. 3) The problem data record detection is viewed as assigning data record labels to the block in the vision tree. Attributes labeling is considered as assigning labels to leaf blocks, and both the tasks can be performed simultaneously.



4) Efficiently incorporates all useful features and their importance weights. Web pages are represented as vision trees by using page layout features such as font, color, size separators are the horizontal and vertical lines that do not cross any node visually.



A block in a web page is represented as a node in the vision tree. Conditional Random Fields are Markov random fields. If  $x, y$  are 2 random variables then  $G = (V, E)$  where  $x$  = label over observation to be labelled and  $y$  is the variables over corresponding labels. Here,  $y$  could be a linear chain.

Now the problem of web data extraction can be narrowed as finding the maximum conditional probability of  $y$  over  $x$ , where  $x$  are the features of all blocks,  $y$  are the possible label assignments.

Based on the hierarchical representation of data a model can be constructed by denoting inner nodes as rectangles and leaf nodes as ellipses. A random variable  $Y$  is associated with each label. Based on the model graph a junction tree is constructed by mapping ellipses clique nodes and rectangles as separators. Then the algorithm selects the arbitrary clique node as a root node. The figure 2 represents the junction tree of the aforementioned example. The value in each internal node are the values of that particular node along with its child nodes. The value in the root is thus  $\{0,1,2\}$ . This is followed by two phases. Using HCRF model for web extraction it is necessary to define the label spaces followed by separation between the variables at leaf nodes and those at inner nodes, because we need to extract the attribute value of the leaf node where a for the inner node has to be identified

whether it is a data record or not. There is a necessity to explicitly define label space and inner label space each for its own purpose. For example the leaf label space for the extraction of products data can be {Product name, product id, price, description etc}.

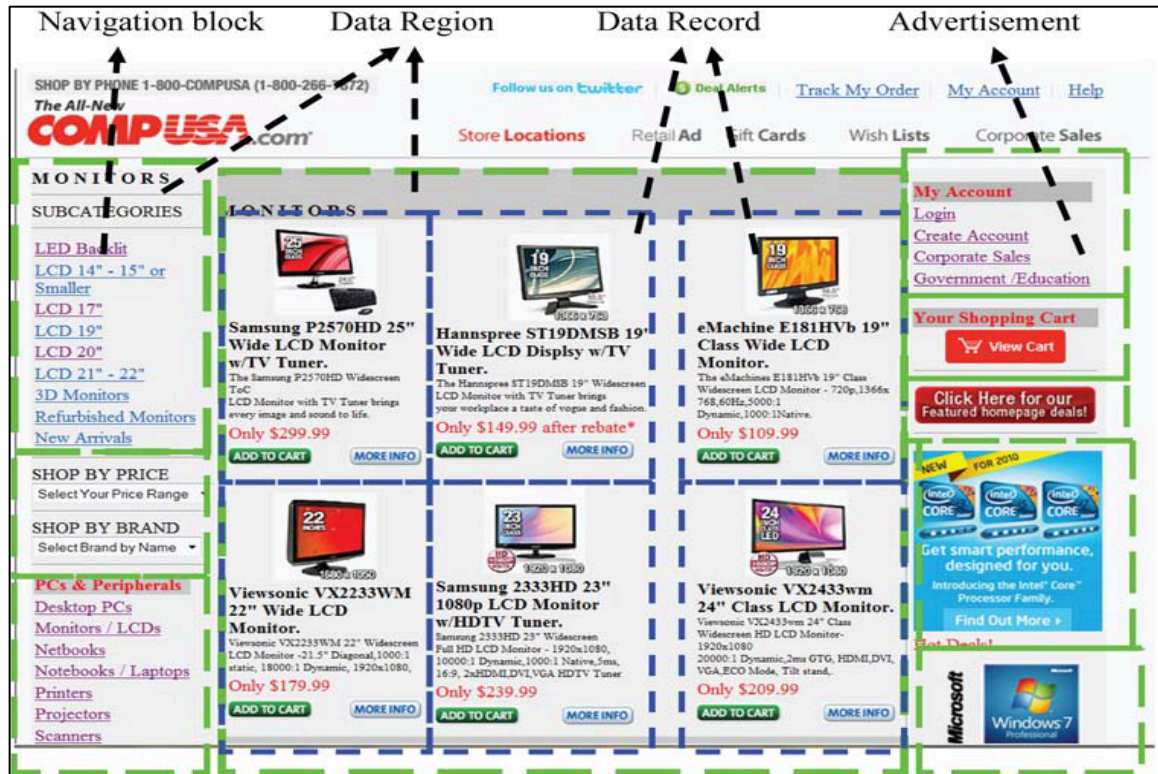
However inner label space has 2 partitions object\_type independent and object\_type dependent. There are certain labels like page head, nav bar, page tail etc, that links to different parts of web page. Such objects are independent of Object\_type. The intermediate labels such as data records are dependent on Object\_type. The features of Elements and Blocks in HCRFS are for each element, the algorithm extracts both vision and content features, all the information can be obtained from the vision tree features. The tree distance of two blocks is defined as the edit distance of their corresponding sub-trees. To exploit the similarity between blocks the shape and dtype distance can also be computed. They also consider that repeated information in web pages does not give useful information, as an example they quote that the “add to cart” button and such types occur repeatedly in each page which are not useful.

#### ***2.2.6. WebOMiner: Towards Comparative mining of Web document objects with NFA*** (Ezeife & Mutsuddy, 2012)

The WebOMiner system develops an automatic object oriented web content extraction and mining technique for mining heterogeneous contents that are complex for in-depth knowledge discovery. Complex querying consists of multiple combined queries along with information kept up to date.

**Example:**

Give comparative analysis of products during the past 2 years including number of sales, views and comments about the product from all the possible websites that deliver the expected product. Consider the following figure taken from (Ezeife & Mutsuddy, 2012) that provides a representation of the regular web content available in any B2C website:



**Figure 5: Web page with data regions and data blocks**

The above product web page consists of Data regions, Data blocks, Data records, Navigation block and the Advertisement block. Data records are usually found in the data regions, which are represented as sub-trees in a DOM tree representation.

**Example:** A Product data record may contain: Product (title: string, image: image-file, difSize (product number: integer, brand: string, price: real)).

The WebOMiner system has 4 modules:

- Crawler Module: Crawler is a spider that crawls through all the desired web pages to

mirror the web contents such as text and image files into the local computer system. In this case, the crawler stores a local copy of the HTML source page in the computer system.

- **Cleaner Module:** The cleaner module is an important part of this technique that helps in cleaning the input HTML document by revising the structure, eliminating noise information, using transformation rules and so on. The output of the cleaner module is given to the extractor module for further processing.
- **Extractor Module:** This module implements the main algorithm of the system.

**Algorithm (The WebOMiner Main Algorithm)**

**Algorithm WebOMiner()**

**Input:** Set of HTML files (WDHTML File) of Web documents

**Output:** Set of patterns of objects (P) for mining results

**Variables:** ContentObjectArray[]

**Begin**

**For** each WDHTML File, **do**

1. Call SiteMapGenerator() to crawl and extract web pages into the local directory.
2. Call HTMLCLEANER-2.2 to cleanup HTML code
3. Call WebOminer.BuildDOMTree() to create a DOM Tree of the refined HTML file and extract web content object sequences from DOM Tree. Store objects in ContentObjectArray[].
4. Call MinerContentObject.IdentifyTuple() to identify data and classify records according to their pattern.
5. Call CreateDBTable() to store data records into a database.

**End For**

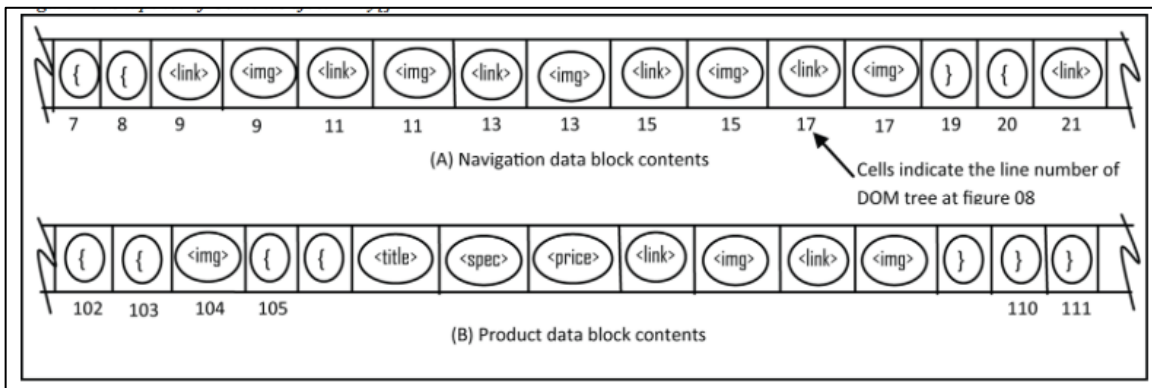
**End**

**Figure 6: WebOMiner algorithm**

The extractor module takes as input the cleaned HTML file and constructs a DOM Tree to classify the data by assigning them to object classes and further appends these objects to the content array list.

Further, MinecontentObject.IdentifyTuple() function is called which generates the seed NFA pattern for the data objects. Classification is done by matching and storing the appropriate tuples in their respective TupleList category.

**Example:** product tuple, image tuple, list tuple, text tuple, form tuple, etc.



**Figure 7: Content Object array for data block identification and classification**

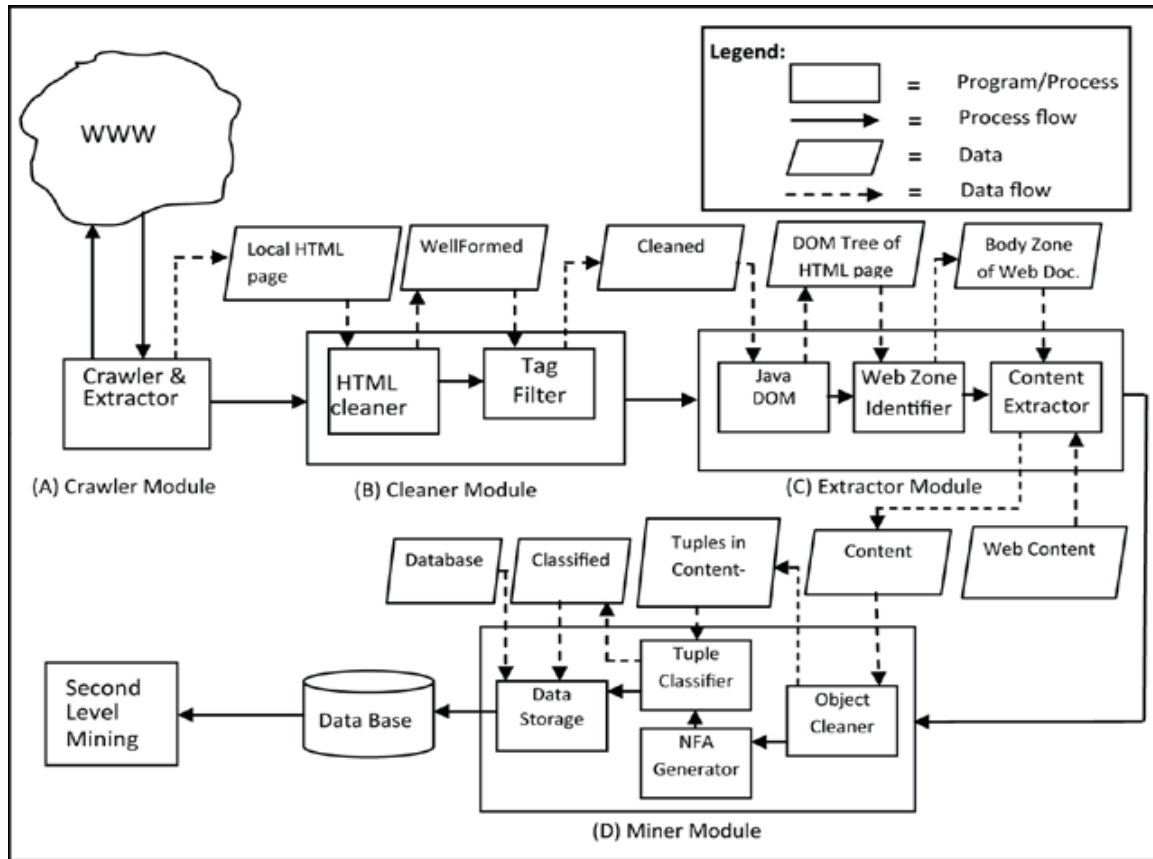
#### **NFA (Non-deterministic finite state automata):**

After the data recorded by the DOM Tree construction is analyzed, NFA is used to identify individual tuples or individual records of data found in this DOM tree structure. The NFA algorithm helps in assigning and classifying the data and their respective data types into well-defined categories.

In NFA each pair of states has a next state. NFA functions by being able to identify any kind of schema.

- Miner module: (CreateDBTable ()): The miner module does the work of storing these data types and the data records in a database. A data warehouse can then be implemented to store many such database records implementing a star schema to form the fact and dimension tables for executing complex queries.

- Architecture of the WebOMiner system is as follows (Ezeife & Mutsuddy, 2012):



**Figure 8: Architecture of the WebOMiner system**

Experiments: Experiments are conducted for a total of 7 different websites for analyzing product related web pages. The resulting precision and recall values are as follows:



Website	Data Records					
	Product	List	Noise	Text	Total	Correct
www.futureshop.ca	10	13	4	0	27	27
www.compUSA.ca	18	21	8	2	49	47
www.bestbuy.ca	7	10	4	1	22	21
www.walmart.ca	2	4	2	-	8	8
www.walmart.ca	2	4	2	-	8	8
www.shopping.ca	40	4	4	-	48	47
www.dell.ca	14	13	4	-	31	28

**Figure 9: Experiments conducted on web data sources**

Precision is defined to be the ratio of total correct records retrieved = 176 by the total number of records retrieved by the system which happen to be 176.

Recall is defined to be the ratio of total correct records retrieved = 176 by the total number of records available in the web page which happen to be 185.

$\text{Precision} = 176/176 = 100 \%$	$\text{Recall} = 176/185 = 96 \%$
---------------------------------------	-----------------------------------

#### ***2.2.6.1. Major Limitations of WebOMiner system:***

1. The WebOMiner system works on only limited types of data tuples such as product name, price, etc. to extract information. Any other kind of information necessary or unnecessary stays unnoticed.
2. System is not automated. Data extraction is performed manually for each web page.
3. Crawler scans through the whole web site. Instead, it needs to automatically focus on only targeted web pages.
4. Object database schemas need to be designed automatically with data extraction for each web site.



5. Object warehouse schema needs to be designed for effective integration of these database schemas generated for each websites.
6. Cleaner module is incapable of handling long tag attributes. Noise data handling is uncertain after the web page is cleaned.
7. Data warehouse schema design and implementation is unknown. Therefore, it is unclear to understand if the system would perform well with different and complex data sets.
8. NFA is a slow and difficult matching technique. Therefore, performance can be disappointing in terms of large datasets or huge websites.

***2.2.7. WebOminer\_S: Comparative Mining of B2C Web Sites by discovering web database schemas: (Ezeife & Peravali, 2016):***

The purpose of this application is to extract schema from the entire webpage in order to retrieve only the required information from any B2C website and store it in the data warehouse for comparative mining. It contains five modules, which are described as follows:

***Crawler module:***

A web crawler also known as a web spider is a meta search engine that integrates the most popular search results from various search engines like Google, yahoo, etc. This application contains a mini web crawler taken from the WebOMiner system (Ezeife & Mutsuddy, 2012) that crawls through the web to extract data from targeted web pages that include texts, tags and image contents. A mirror image of this document is stored in the local hard drive.

### ***Cleaner module:***

HTML cleaner is open source software used to clean web pages that are usually un-clean in the form of missing tags and ill formed structure; therefore they are unsuitable for further processing. HTML cleaner works by redefining all elements individually and maintaining a well formed structure. An example is shown below to represent Uclean and Clean HTML file:

#### 1. Unclean HTML:

```
<h1>CONTENT</h1>
<table id=table1 cellspacing=2px
  <td><a href=index.html>1 -> Home Page</a>
  <td><a href=intro.html>2 -> Introduction</a>
```

#### 2. Cleaned HTML:

```
<h1>CONTENT</h1>
  <table id="table1" cellspacing="2px">
    <tbody>
      <tr>
        <td>
          <a href="index.html">Home Page</a>
        </td>
        <td>
          <a href="intro.html"> Introduction</a>
        </td>
      </tr>
    </tbody>
  </table>
```

The HTML cleaner is used to transform unclean html page to a clean html page. It helps in quickly skipping certain unwanted tags or attributes and restructuring the page contents. Therefore, it easily avoids the expensive document model manipulation process required after cleaning.

This module is implemented in java file called runBatchFile.java. The function runFile() calls the input argument file called inputUncleaned HTML. As the name says, this file is unclean and needs structuring.

This file is then provided to the htmlcleaner-2.2.jar in the form of a batch file called cleanHTML.bat, which produces the cleaned HTML file. This cleaned file is stored in the project folder with the name outputCleanedHTML. The module is designed to create a batch file if it does not exist while cleaning process begins.

The syntax to use the html cleaner-2.2.jar with source and destination arguments is as follows:

```
Java -jar*rootfolder\htmlcleaner-2.2.jar src=business.html dest=business_.html
```

Here, the src=business.html is the input unclean HTML file and

Dest=business\_.html is the cleaned HTML file which is then stored at a specific location which is later used by the HTML parser.

An HTML parser or syntactical analyzer is a parser used to analyze a string of symbols either in natural or computer language using the rules of grammar. But in this case, the HTML parser takes in HTML code and extracting relevant information like the title of the page, paragraphs in the page, headings in the page, links, bold text etc.

The parser called extractSchema.java uses the cleaned HTML page. The function used to trigger this is parseTheHTML (). ExtractSchema is the main class; therefore an object of this class is created which calls an argument having the name of the business being amazon, ebay, bestbuy, etc.

**Example:** Object generated of the extractSchema class is as follows,  
(Ezeife & Peravali, 2016):

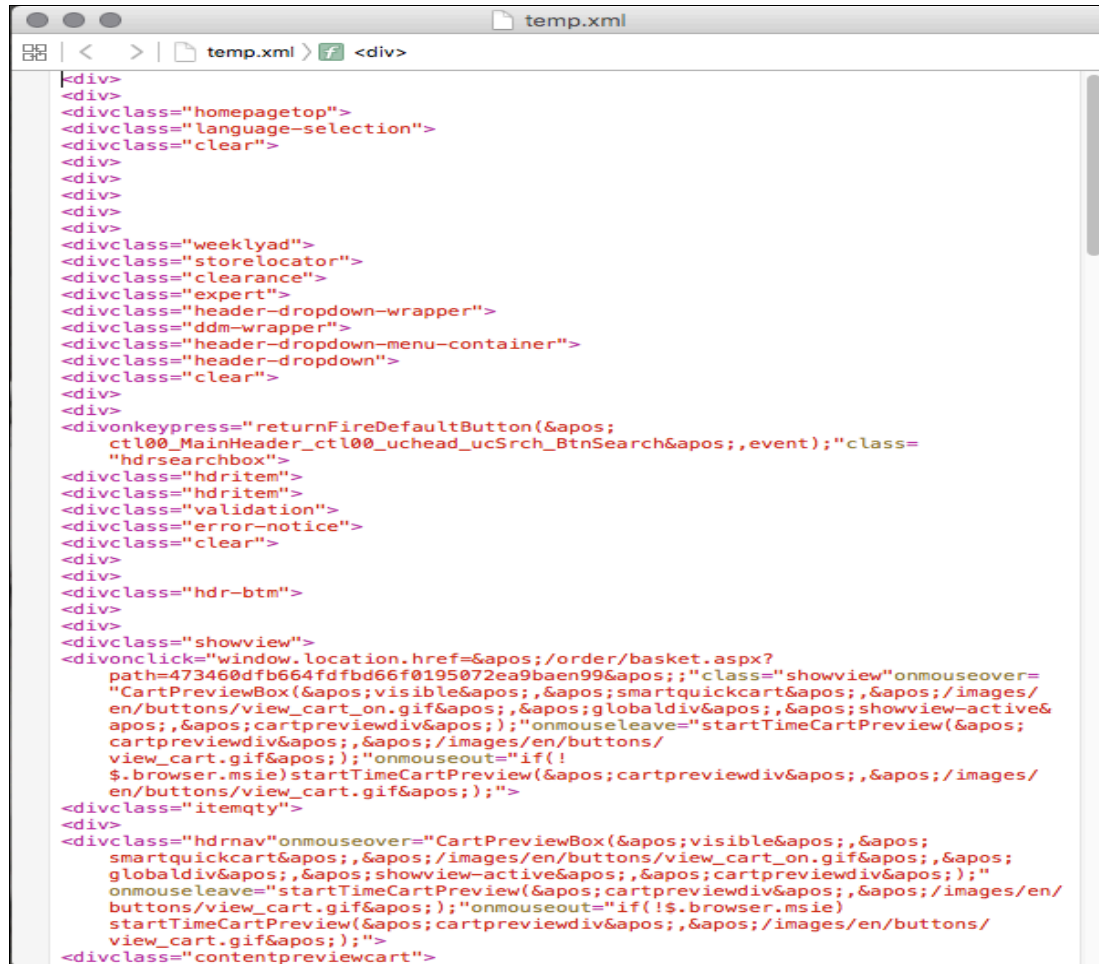
```
public static void main(String[] args) throws Exception{
    runBatchFile obj2 = new runBatchFile();
    obj2.runFile(args[0]);
    Thread.sleep(5000L);
    /*
     * Extracts the database schema from the cleaned html page
     */
    new extractSchema(args[0]);
}
```

**Figure 10: Extract Schema class main() method**

The method called from the Extract Schema class does the following work on the cleaned HTML file:

1. Extracts and stores all block of data such as <div>, <span>, <table> into a file called temp.xml.
2. All the attributes present in the page are eliminated and only the class attribute is saved into the xml file.

A view of the temp.xml file contents (Ezeife & Peravali, 2016):



```

<div>
<div>
<divclass="homepage-top">
<divclass="language-selection">
<divclass="clear">
<div>
<div>
<div>
<div>
<divclass="weeklyad">
<divclass="storelocator">
<divclass="clearance">
<divclass="expert">
<divclass="header-dropdown-wrapper">
<divclass="ddm-wrapper">
<divclass="header-dropdown-menu-container">
<divclass="header-dropdown">
<divclass="clear">
<div>
<div>
<divonkeypress="returnFireDefaultButton(&apos;
    ctl00_MainHeader_ctl00_uchead_ucSrch_BtnSearch&apos;;event);"class=
    "hdrsearchbox">
<divclass="hdritem">
<divclass="hdritem">
<divclass="validation">
<divclass="error-notice">
<divclass="clear">
<div>
<div>
<divclass="hdr-btm">
<div>
<div>
<divclass="showview">
<divonclick="window.location.href=&apos;/order/basket.aspx?
    path=473460dfb664fd666f0195072ea9baen99&apos;;"class="showview"onmouseover=
    "CartPreviewBox(&apos;visible&apos;;&apos;;smartquickcart&apos;;&apos;;/images/
    en/buttons/view_cart_on.gif&apos;;&apos;;globaldiv&apos;;&apos;;showview-active&
    apos;;&apos;;cartpreviewdiv&apos;;);"onmouseleave="startTimeCartPreview(&apos;
    cartpreviewdiv&apos;;&apos;;/images/en/buttons/
    view_cart.gif&apos;;);"onmouseout="if(!$.browser.msie)
    startTimeCartPreview(&apos;cartpreviewdiv&apos;;&apos;;/images/
    en/buttons/view_cart.gif&apos;;);">
<divclass="itemqty">
<div>
<divclass="hdrnav"onmouseover="CartPreviewBox(&apos;visible&apos;;&apos;;
    smartquickcart&apos;;&apos;;/images/en/buttons/view_cart_on.gif&apos;;&apos;;
    globaldiv&apos;;&apos;;showview-active&apos;;&apos;;cartpreviewdiv&apos;;);"
    onmouseleave="startTimeCartPreview(&apos;cartpreviewdiv&apos;;&apos;;/images/en/
    buttons/view_cart.gif&apos;;);"onmouseout="if(!$.browser.msie)
    startTimeCartPreview(&apos;cartpreviewdiv&apos;;&apos;;/images/en/
    buttons/view_cart.gif&apos;;);">
<divclass="contentpreviewcart">

```

Figure 11: Temp.xml file

Once the program reads the file temp.xml it further creates the occurrence.data file, which holds the account of the child and parent frequency. This file is important because it is observed that the data records are huge in number in any product page. Therefore the format for these records will be naturally more than any other HTML structure written for any other purpose.

A view of the occurrence.data file:



```
<div> 42
<divclass="clear"> 17
<divclass="contentno-bottom-pad"> 12
<divrenderemptytext="false"> 3
<divclass="productlast"> 5
<divclass="theme-product-headlineclear-fix"> 7
<divclass="inner-product-wrapper"> 9
<divclass="product-content-wrapperclear-fix"> 9
<divclass="pricepill"> 6
<divclass="rebate-top"> 6
<divclass="pricepillred"> 6
<divclass="more-like-this"align="center"> 8
<divclass="fieldbox-ftr-br-col"> 3
<divclass="headline-background"> 3
<divclass="type-headline"> 3
<divclass="ts-image"> 7
<divclass="TechSpotLight-content"> 5
<divclass="article-headline"> 7
<divclass="article-date"> 7
<divclass="article-body"> 7
<divclass="art-divider"> 7
<divclass="supporting-info"> 6
<divclass="product"> 5
<divclass="theme-a"> 6
<divclass="rating-holder"> 4
<divclass="rating"> 4
```

Figure 12: Occurrence.data file

The occurrence.data file gives the information about the parent node of the block usually present with maximum frequency including children blocks as well.

### ***Frequent pattern structure finder:***

Frequent pattern structure finder analyzes the occurrence.data file to read the tag information and the frequency of these tags to identify the data of focus. Further, it functions as follows:

1. It creates a DOM Tree of this node information using the DocumentBuilderFactory. The documentBuilder functions to retrieve the DOM document from the XML document.

2. Xpath is then used for defining various parts in the XML document and navigating between them. It retrieves the complete node using parent tag name.
3. The frequency of this node obtained is stored using the function printFinalTree().
4. The frequency children block is considered as the product data block.

**Example:**

Figure 21 is the occurrence.data file. It shows maximum number of child nodes present within each parent node. Structure Finder algorithm marks these occurrences in order to find the most frequent pattern. Tags such as <divclass="clear">, <divclass="contentno-bottom-pad">, <divclass="product-contenttn-wrapperclear-fix"> provide the structure to extract the database schema.

The frequent structure finder algorithm is shown below:

**Algorithm FSFinder ():**

**Input:** occurrence.data file, #WDHTML file

**Output:**

Nodelist frequent structures patterns (FRS), data tuples

Xmlnodetag: string (concatenating all the data tuples)

xpathExpression expr: string, path: string

Nodelist tag structure: struct list, data tuple: product block

Packages: Java DOM, Java Path, Java transformer

**Begin**

1. Call DocumentBuilderFactory ()
2. Call xpathFactory () to create xpath object to obtain the node list
3. **For** each tag in occurrence.data file,  
     Retrieved xml is stored in xmlnodetag[1]  
     Path = "//div[@class="+xmlnodetag[1]]";  
     xpathExpression expr = xpath.compile(path)  
     struct list = xpath.evaluate // retrieves all the matching nodes

```

nodelength = struct list.getlength()
#childs = struct list (0).getchildnodes().getlength()
If nodelength >= final nodelength, then final nodelength = nodelength &
If #childs > final childs, then final child = #child

```

```

Product block = xmlnodetag[1]
Data tuples (product blocks) = struct list

```

**End For**

4. **For** structlist child nodes

Begin

Append child nodes to the Frequent structure list

**End for**

5. Return string of integrated Data tuples (product blocks)

**End**

***Schema extractor module:***

The extractSchema.java file performs the task of this module. On identification of the product block it efficiently retrieves the schema in the following manner:

1. All tags in the block are read to recognize the data type.
2. Each type of data is then classified into their respective types.

***Example:***

Tags such as <img> are categorized into image data type and so on.

Tags such as <p>, <h>, <li>, <a>, <span> are considered optional and stored as a “string” type of data.



3. Finally the schema consists of all the tag fields and their data types integrated into a single String variable (WebSchema), which is the final output of this project that can be further stored in a database.

A view of the string variable WebSchema is as follows (Ezeife & Peravali, 2016):

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

The extracted data above is from the best buy website. Similarly, many more website data's can be extracted and stored in a data warehouse with the entire source website attributes and properties participating as the dimension tables values.

#### ***2.2.7.1. Limitation for the WebOMiner\_S:***

1. Limitation of the WebOMiner\_S is that it does not have a web-based application. WebOminer\_S does not have a provision for querying data records available in the data warehouse.
2. Also there is no provision for integrating the data records as for each extraction, the system creates a new database table. Therefore, new integration rules need to be specified after every extraction. The product tables generated dynamically do not have appropriate column labels for the discovered schema. It is difficult to integrate product data from multiple data records from different websites such as bestbuy, Costco, etc. In order to overcome this, columns have to be named appropriately.

## CHAPTER 3: PROPOSED WebOMiner\_SRec SYSTEM

### 3.1. System Design, architecture and algorithm:

The purpose of the proposed WebOMiner\_SRec system is to take input: the discovered Product\_Schema from the (WebOMiner\_S system containing: crawler module, cleaner module, HTML parser module, frequent pattern finder module schema extractor module) and developing the (WebOMiner\_SRec: Schema\_Mapper module and the Query Management Interface). The output of the WebOMiner\_SRec system is the answer to all the queries made to the datawarehouse for analysis. Let us understand all the modules and their working in detail.

#### 3.1.1. Architecture for our WebOMiner\_SRec system:

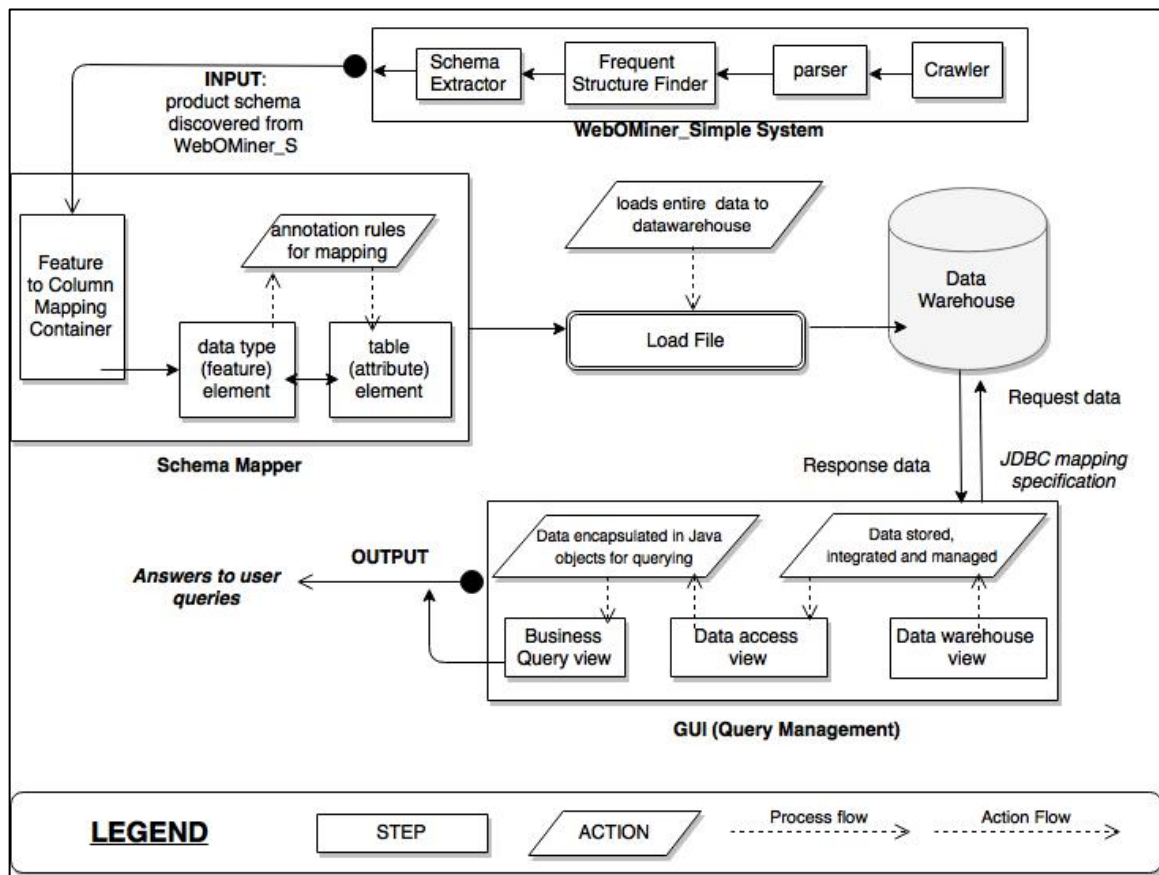


Figure 5: WebOMiner\_SRec system architecture

Our systems aims at developing a Simplistic user interface by mounting the WebOMiner\_SRec system on the web domain using Java Enterprise Web functions. The WebOMiner\_S is a web data extraction system for yielding all essential information about products. The WebOMiner\_S on its final stage outputs a product schema such as Product 1(column 1, column 2, ... column n), which is then taken as an input for our WebOMiner\_SRec system. The WebOMiner\_SRec has an algorithm called “Schema\_mapper” for giving meaningful names to the column attributes of the already existing product table in the data warehouse. It then creates a mapping container that provides an environment based on annotators to map all the column elements to the appropriate feature element such as (Dell Laptop Inspiron series to product name). Annotators are the rules or conditions written in a <scriptlet> format that contains an annotation tree to map the column attributes to their exact data types. A Load file element is introduced to perform the transport of all the mapped data from the database to the data warehouse.

Once the transformed data is loaded to the datawarehouse product table, another table called the seller table is created which stores all the information about the seller (for e.g. Seller name, Id, seller url, date/time). This information is referenced directly from the product table. Seller ID is the foreign key attribute used for integrating the Seller information. Therefore, the final step of the WebOMiner\_SRec is to create a secure non-terminating connection between the data warehouse and the GUI. A connection is created calling the JDBC.session.connection () methods to request data from the warehouse. Java object method() handles the transaction processing for providing the perfect response to the requested interface query. The final output of the WebOMiner\_SRec is the ResultList object that gives query results to the web users.

### ***3.1.2. Algorithm for our Web Recommendation system:***

Algorithm WebOMiner\_SRec

**Input:** Product\_schema discovered which is the output obtained from WebOMiner\_S

**Output:** Data from warehouse encapsulated in java objects (answer to user queries)

Other: Annotators: .xml

**Begin**

1. Schema\_Mapper: Define a feature to column (attribute) mapping container:

**For** each (input product\_data),

- 1.1. Define a feature <element> (n) and a column <element> (m)
- 1.2. Check the annotation rules for mapping feature and column <element>
- 1.3. If “n” satisfies the rules for “m” condition

Then, map “n to m”

Note: the rules are defined based on RegEx and equivalent data types

- 1.4. If  $n \neq m$ , set m value = Null//

2. Loadfile<element>:

- 2.1. If mapping is performed completely for all “m to n” values

Then, load all mapped data to loadfile.xml

- 2.2. Loadfile.xml specifies the buffer size (0 – 1024 MB)

- 2.3. For every loadfile.xml,

Check: if loadfile.xml is ready “OR” in uploading stage from Mapping\_Container.

- 2.4. If ready: upload the mapped data to the datawarehouse.

**End For**

3. Query\_Manager: Establish connection between data warehouse and java object class.

4. Object data access: If connection to the database is made:

- 4.1. Query request consists: Access permission to the data values.

- 4.2. If required data exists,

Encapsulate the data into Java object for access by appropriate display fields.

If not, gives an IO Exception

- 4.3. Map HTML/CSS interface fields to exact display data objects

- 4.4. Render query data on the Interface to show results.

**End**

### ***3.1.3. System modules and working:***

#### ***3.1.3.1. Schema\_Mapper module:***

##### **Algorithm for SchemaMapper:**

**Input:** Product schema and data tuples obtained from WebOMiner\_S

**Output:** Product schema with meaning column values assigned to the product\_domain table in the datawarehouse and the datatuples to be stored in the datawarehouse product\_domain table.

Other: MappingAnnotations.xml file

##### **Begin**

1. Schema\_Mapper: Define a feature to column (attribute) mapping container:  
for each (input product\_data),
2. Define a feature <element> (n) and a column <element> (m)
3. Connect to the database using
4. Check if “m” equals to every column attribute of the product\_domain table
5. Check the annotation rules for mapping feature and column <element>  
Where rules define conditions for column to data type matching:  
If value=integer [0-9 && (!/) ends with \$] map the integer value to price column,  
etc.
6. If “n” satisfies the rules for “m” condition  
Then, map “n to m”
7. if n != m, set m value = Null//  
end for;

##### **End**

##### ***How Schema\_Mapper works:***

It makes sure that the value of the feature element is stored in a column of the appropriate type. A feature structure is the underlying data structure that represents an analysis result. A feature structure is an attribute-value structure. Each feature structure is of a *type* and every type has a specified set of valid features or attributes (properties), much like a Java class.

Features have a range type that indicates the type of value that the feature must have, such as String.

For example, the text span "HP laptop 1TB HDD i3 processor" might be spanned by an annotation of type Product with the features (product name, product description).

The type system defines the types of objects (feature structures) that may be discovered. The type system defines all possible feature structures in terms of types and features (attributes). It is possible to define any number of different types in a type system. A type system is domain and application specific.

The analysis annotators produce their analysis results in the form of *annotations*. Annotations are a special kind of feature structure that is designated for analysis processing. An annotation spans or covers a piece of input data and is defined in terms of its beginning and end positions.

***Working example:***

An annotator that recognizes expressions created for the data "100.55 US Dollars" an annotation of type Expression that covers the text with the feature currencySymbol set to "\$".

All Schema\_mapper structures are stored in the MappingAnnotations.xml file.

The Schema\_Mapper structure contains the following objects:

1. The type system description that indicates the types, subtypes, and their features for (e.g. Dell laptop i5 processor).
2. The column and feature attributes (e.g. product description).

To transport the Schema\_mapper structure to the data warehouse we call the mapping () method that contains the database connection configuration information and a description of the appropriate connection path to the data warehouse table. Using SchemaMapper(), each feature is checked and mapped correctly to the already existing product table columns. The following example shows the mapping rules generated by the Schema Mapper structure:

Format structure for the @annotator is as follows:

```
<SchemaMappingSpec>    <name>com.tt.DocumentAnnotation</name>
<filter syntax="FeatureValue">toBeProcessed=0</filter>
<JdbcMappings>    <explicitMappings>
<explicitMappingRuleapplyToSubtypes="false">
<type>com.ibm.omnifind.types.Product</type>
<table>sample.Product</table>
<featureMappings>
<feature>uniqueId()</feature>
<column>ProductId</column></featureMapping>
<featureMapping><feature>uniqueId()</feature><column>ProductId()</column>
</featureMapping>
```

Annotator mapping rules are provided for the <Feature> and the <Column> element. These mapping rules perform accurate validation methods.

***For example:***

```
$validator.mappingRule("currency", function(feature{product_price}, column{PRICE})
{ return this.(featureMapping) ||
    /\$(\d{1,3}(\,\d{3})*(\d+))(\.\d{2})?$/ .test(productdb.table{price, product_price});)
};
```

The above rule explains:

1. Validation annotation for price. It calls the MappingRule method to validate <feature> product\_price, and <column> PRICE.
2. Mapping rule method checks for data integrity constraints for e.g. If it is a price attribute, it checks for integer value, decimal value, "." Product before the decimal value and also symbols like "\$".
3. Validator annotation further returns a <featureMapping> element that is required to start the next process of data loading.

The Schema\_mapper structure, is created as follows:

1. Create the Schema structure mapping file as the XSD schema: JDBCMapping.xsd in ES\_INSTALL\_ROOT/configurations/parserservice/data directory.
2. Include the mapping in a <JdbcConfiguration xmlns="jdbc/100/xml"> element. The namespace (specified in the xmlns attribute) must be exactly as shown.
3. Add a <databaseConnection> element that contains all database configuration information and a <JdbcMappingSpec> element that contains the mapping structure for the product table columns such as (product name, product id, prod description, etc.) that are stored in the data warehouse product table.
4. Add the following component elements to the <databaseConnection> element:
  - (i) Mandatory: A <connectionUrl> element. This element contains the database connection URL. Depending on the JDBC driver implementation, we can use local or remote access to the database.
  - (ii) Mandatory: A <driver> element. This element contains the name of the JDBC driver class, for example com.db2.jcc.DB2Driver.
  - (iii) Mandatory: A <driverLibraries> element. This element lists the driver libraries.
  - (iv) Mandatory: An <authentication> element. This element contains the user name and password for the database (e.g. username: root, password: tiger).

The following component elements are added to the <jdbcMappingSpec> element:

A <JdbcMapping> element shows the types and features that are mapped to the data warehouse product table and its respective columns.

A <ExplicitMapping> element. This element is mandatory. It must have one or more <ExplicitMappingRule> elements that define the annotation types and their subtypes. If a mapping is defined in the explicit mappings rule, all of the annotations that match the mapping definition will be stored in the data warehouse. If there is an explicit mapping rule for the Seller annotation, all products related to the seller are added to the seller table in the data warehouse.



5. The <ExplicitMappingRule> must contain the attribute applyToSubtypes, which, if set to true, stores not only the feature structure that is listed in the <type> element, but also all of the feature structures derived from it. Add the following component elements to <explicitMappingRule>:
  7. A <type> element that contains the feature structure type.
  8. A <table> element that contains the product schema and table name. The syntax follows the rule schema.table\_name.
  9. A <filter> element that contains an explicit condition that is evaluated each time the mapping rule matches. If the condition evaluates to true, the annotation stores the feature structure in the mappings file.
  10. The <featureMapping> element component structure works as follows:

If we are mapping a feature or feature path, the component elements include:

  - 11. A <feature> element with the name of the feature. The feature must be defined for the feature structure. We use a feature path construct to list all the features. (for e.g. feature/prod\_ID/schema.product\_table).
  - 12. A <column> element with the name of the column in which the feature value is to be stored (for e.g. schema.product\_table/product\_name: Apple iphone 6). Database columns that are not used in any feature mappings use a default value (usually null).

### ***3.1.3.2. Datawarehouse Loader:***

#### **Algorithm for Datawarehouse loader:**

**Input:** product\_schema obtained from schema mapper module and data tuples to be stored in the data warehouse product table.

**Output:** data records stored in the existing product table and seller data stored in the seller table.

**Begin**

1. Create database connection along with product\_domain table with column attributes specific to the product schema obtained from Schema Mapper from Mapping Annotations.xml file.
2. A loadfile<> element is defined to transport all the extracted data to the product\_domain table.
3. For each attribute;  
    Store all the data tuples with their unique ID  
    End for;
4. Create another seller\_fact table having attributes for seller information.  
    For each seller in the product\_domain table;  
    Map the unique ID of the seller to the seller\_fact table as the Foreign key along with the date/time attribute.  
    End for;

**End*****How datawarehouse loader works:***

1. A <loadFile> element contains the following components:
  - The load file directory in a <loadFileDirectory> element.
  - The load file size in a <loadFileSize> element. The load file size limits are 10 <= loadFileSize <= 10240 (10MB <= loadFileSize <= 10GB). If no value is defined, the default is 1024 MB (1GB).
  - The load script name in a <loadScript> element.If we do not specify a <loadFile> element, all data is stored directly in the database by using JDBC connection.

## 2. Data Access Object:

DAO stands for Data Access Object, which is commonly used for database interaction. DAOs exist to provide a means to read and write data to the database and they should expose this functionality through an interface by which the rest of the application will access them. The DAO support makes it easy to work with data access technologies like JDBC, Hibernate, JPA, or JDO in a consistent way. Therefore, our system implements Hibernate connectivity to transport the LoadFile.xml to the database.

3. A data warehouse is generally understood as an integrated and time-varying collection of data primarily used in strategic decision-making (Husemann et. al., 2000). Data Warehouse is used to design a relation for a Product dataset for the following reasons:

1. Data Warehouse is subject oriented: Finance, Business, etc.
2. It stores Meta Data information
3. It is useful for accurate querying and analysis
4. It is a non-volatile repository

The WebOMiner Simple system implementation defines the use of a Data Warehouse as a tool to design relations between database records, as the main purpose of this application is to query historical information for analysis.

Therefore, we create a Seller\_fact table to store all the seller information. Such as Seller\_id, Seller\_name and Date/time. We also create a product\_domain table, which stores all the extracted data records. All the seller data tuples along with their unique id are stored in the seller\_fact table. These id's are the foreign key attributes. This helps to integrate all the data records from the product\_domain table into the seller\_fact table to categorize and save information on the basis of sellers or vendors. This also helps in accurate querying and helps to categorize records in the future when more sellers are added to the system database.

**Working Example:**

1. Figure 14 represents the seller\_fact table created to store and categorize all the seller information from integrated from the product\_domain table in the data warehouse. This table integrates and saves the ID, name, url link and the website name.

Example syntax: Create table seller\_fact

(ID int not null;

Name string, url string, website string)

Add Foreign Key constraints:

```
Alter Table Seller_Fact ADD CONSTRAINTS _FK_ID FOREIGN KEY (ID)
REFERENCES product_domain (ID);
```

For example: **Seller\_table**

ID	NAME	URL	WEBSITE
3	BESTBUY	http://www.bestbuy.ca/en-CA/category/iphone/36586.aspx?searchRedirect=iphone	BestBuy.ca

TABLES		Search: <input type="text" value="seller_id"/> <input type="button" value="="/> <input type="text" value=""/> <input type="button" value="Q"/>		<input type="text" value="Search"/>	
<input checked="" type="checkbox"/>	fact_domain				
<input checked="" type="checkbox"/>	seller				
		seller_id	name	scrape_url	website
		1	amazon	https://www.amazon.ca/s/ref=nb_sb_noss/153-7022464-7249238?url=search-alias%3Daps&field...	www.amazon.com
		2	CraigList	https://newyork.craigslist.org/search/sss?sort=rel&query=	CraigList
		3	bestbuy	http://www.bestbuy.ca/en-CA/Search/SearchResults.aspx?query=	BestBuy
		4	costco	https://www.costco.ca/CatalogSearch?keyword=	CostCo
		5	walmart	https://www.walmart.ca/search/	walmart

**Figure 14: Seller\_fact table from the datawarehouse**

- Figure 15 represents the product\_domain table created to store all the data records along with their unique id's and the date/time attribute.

Example syntax: Create table SellerFact

(id int PK not null, comments null, currency null, name string not null, link PK, price int not null, seller string, date/time sysdate;  
)

For example: **product\_domain table**

Id	Name	Currency	Price	Seller	Date/time
1	Iphone 6s 32gb, space grey	Null	49.99	bestbuy	03-04-2017 11:59:00

Search: id = Search										
fact_domain	id	comments	currency	name	original_link	price	rating	seller	ship_info	time
	1	NULL	NULL	Bell Apple iPhone 6s 32GB Smartphone - Space Grey - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	2	NULL	NULL	Rogers Apple iPhone 6s 32GB Smartphone - Space Grey - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	3	NULL	NULL	Fido Apple iPhone 6s 32GB Smartphone - Space Grey - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	4	NULL	NULL	Rogers Apple iPhone 6s 128GB Smartphone - Space Grey - 2 Year Agreement	NULL	49.99	NULL	BestBuy	NULL	2017-03-31 11:59:10
	5	NULL	NULL	Fido Apple iPhone 6s 32GB Smartphone - Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	6	NULL	NULL	Bell Apple iPhone 6s 32GB Smartphone - Rose Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	7	NULL	NULL	TELUS Apple iPhone 6s 32GB Smartphone - Rose Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	8	NULL	NULL	Rogers Apple iPhone 6s 128GB Smartphone - Rose Gold - 2 Year Agreement	NULL	49.99	NULL	BestBuy	NULL	2017-03-31 11:59:10
	9	NULL	NULL	Koodo Apple iPhone 6s 32GB Smartphone - Space Grey - With A Tab Large	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	10	NULL	NULL	Rogers Apple iPhone 6s 32GB Smartphone - Rose Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	11	NULL	NULL	Rogers Apple iPhone 6s 128GB Smartphone - Gold - 2 Year Agreement	NULL	49.99	NULL	BestBuy	NULL	2017-03-31 11:59:10
	12	NULL	NULL	TELUS Apple iPhone 6s 32GB Smartphone - Space Grey - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	13	NULL	NULL	Virgin Mobile Apple iPhone 6s 32GB Smartphone - Space Grey - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	14	NULL	NULL	Fido Apple iPhone 6s 32GB Smartphone - Rose Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	15	NULL	NULL	Rogers Apple iPhone 6s 128GB Smartphone - Silver - 2 Year Agreement	NULL	49.99	NULL	BestBuy	NULL	2017-03-31 11:59:10
	16	NULL	NULL	Rogers Apple iPhone 6s 32GB Smartphone - Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	17	NULL	NULL	Bell Apple iPhone 6s 32GB Smartphone - Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	18	NULL	NULL	Rogers Apple iPhone 6s 32GB Smartphone - Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	19	NULL	NULL	Rogers Apple iPhone 6s 32GB Smartphone - Silver - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	20	NULL	NULL	Virgin Mobile Apple iPhone 6s 32GB Smartphone - Rose Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	21	NULL	NULL	Koodo Apple iPhone 6s 32GB Smartphone - Rose Gold - With A Tab Large	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	22	NULL	NULL	Fido Apple iPhone 6s 32GB Smartphone - Silver - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	23	NULL	NULL	Bell Apple iPhone 6s 128GB - Space Grey - 2 Year Agreement	NULL	49.99	NULL	BestBuy	NULL	2017-03-31 11:59:10
	24	NULL	NULL	TELUS Apple iPhone 6s 32GB Smartphone - Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	25	NULL	NULL	TELUS Apple iPhone 6s 32GB Smartphone - Silver - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	26	NULL	NULL	Apple iPhone 6s 32GB Smartphone - Space Grey - Open Box	NULL	649.99	NULL	BestBuy	NULL	2017-03-31 11:59:10
	27	NULL	NULL	Koodo Apple iPhone 6s 32GB Smartphone - Gold - With A Tab Large	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	28	NULL	NULL	Apple iPhone 6s 32GB Smartphone - Rose Gold - Open Box	NULL	649.99	NULL	BestBuy	NULL	2017-03-31 11:59:10
	29	NULL	NULL	Virgin Mobile Apple iPhone 6s 32GB Smartphone - Silver - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10
	30	NULL	NULL	Virgin Mobile Apple iPhone 6s 32GB Smartphone - Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10

**Figure 15: product\_domain table from the data warehouse**

### 3.2.3.3. *Query Manager:*

Algorithm for Query Manager:

**Input:** Database connection parameters from the Data warehouse loader module

**Output:** Answers to user queries displayed on the GUI

**Begin:**

1. Define a Web MVC framework to handle the process flow from making a request to the data warehouse and getting a response on the GUI.
2. For every request to the data warehouse;  
Establish a secure session,  
Establish a secure connection,  
Have error free user queries to get optimal results,  
End the connection until a new request is generated.  
End for;
3. Display the appropriate results to the user in the form of Java objects (as java is Object oriented language. Objects are real time instances)

**End**

#### *How the Query manager Works:*

1. The Web-MVC module Model-View-Controller (MVC) is implementation for web applications. The Web-Socket module provides support for WebSocket-based, two-way communication between the client and the server.

- The **Model** encapsulates the application data and in general they will consist of POJO.
- The **View** is responsible for rendering the model data and in general it generates the HTML output that the client's browser could interpret.

- The **Controller** is responsible for processing the flow from the interface to the business logic and the database connection.

Following is the sequence of events corresponding to an incoming HTTP request to DispatcherServlet: After receiving an HTTP request, DispatcherServlet consults the HandlerMapping to call the appropriate Controller. The Controller takes the request and calls the appropriate service methods based on GET or POST method. The service method will set model data based on the java business logic and returns view name to the DispatcherServlet. The DispatcherServlet will take help from ViewResolver to pickup the defined view for the request. Once view is finalized, The DispatcherServlet passes the model data to the view, which is finally rendered, on the browser.

The DispatcherServlet: It delegates the request to the controllers to execute the functionality specific to it. The `@Controller` annotation indicates that a particular class serves the role of a controller. The `@RequestMapping` annotation is used to map a URL to either an entire class or a particular handler method.

```
@Controller
@RequestMapping("/data")
public class MinerController {
    @RequestMapping(method = RequestMethod.GET)
    public String printRecord(ModelMap model) {
        model.addAttribute("message", "Display result!");
        return "Record";
    }
}
```

The @Controller annotation:

It defines the controller class. Here, the first usage of @RequestMapping indicates that all handling methods on this controller are relative to the /minerWeb path. Next annotation @RequestMapping (method = RequestMethod.GET) is used to declare the Recordprint() method as the controller's default service method to handle HTTP GET request.

2. After a successful connection, HTTP Response provides the Result List object to the view interface. The format of the ResultList data object is as follows:

// ResultList object [0] [Product _1] {"Java data object"}		
ID	Int	33
Product_name	String	Dell Inspiron i5 series
Price	Float	499.99
Date	date	03-04-2017

***Working Example:***

Hibernate application is provided using “org.hibernate” package in Java.

1. Session creation:

In order to connect to the database from the Java enterprise logic, a session is created using the following packages:

- org.hibernate.sessionfactory
- prg.hibernate.session

For example let us consider the code developed in our approach for creating a session between the Java class and database:



```

package main;

import java.io.Serializable;
import java.sql.Connection;
import model.Product;
import org.hibernate.LockMode;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

/**
 *
 * @author root
 */
public class Main {
    public static void main(String[] args) {
        Configuration configuration=new Configuration();
        configuration.configure("conf/hibernate-product.cfg.xml");
        SessionFactory factory=configuration.buildSessionFactory();
        //System.out.println(factory);
        Session session=factory.openSession();
    }
}

```

**Figure 16: Create Session**

## 2. Configuration file:

Once the session is created, a call to the configuration file is made to establish a connection between the Java class and the database. It is done using the “hibernate.connection.driver\_class”.

**Example: contains the username: “Scott” and password: “tiger” authentication**

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" |
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property>
    <property name="hibernate.connection.driver_class">oracle.jdbc.OracleDriver</property>
    <property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:orcl</property>
    <property name="hibernate.connection.username">scott</property>
    <property name="hibernate.connection.password">tiger</property>
    <property name="hibernate.show_sql">true</property>
    <mapping resource="mapping/hibernate-product.hbm.xml"/>
  </session-factory>
</hibernate-configuration>

```

**Figure 17: Hibernate configuration file**

### 3. Mapping file:

It is necessary that the names defined for Java class and the database are same. Let us consider an example for mapping constraints.

**Example:** *Product is the name of the table in the data warehouse whereas model.Product is the name of the Java class too.*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="model.Product" table="PRODUCT">
    <id name="id" column="PID"/>
    <property name="name" column="PNAME"/>
    <property name="price" column="PRICE"/>
  </class>
</hibernate-mapping>
|
```

**Figure 18: Hibernate Mapping file**

Once the connection is completed, user specific queries can be made to the system. Let us see how our system can perform query operations on data-warehouse tables using SQL and JDBC Template object.

#### ***Working Example for database querying:***

(i) Query: Find all records from the product data table from seller name (BestBuy.ca). “%” Symbol is used to classify all the seller information to satisfy the input user criteria.

```
if (StringUtils.isNotBlank(searchParams.getSellerName())) {
    if (StringUtils.isEmpty(dynamicParams)) {
        dynamicParams += " where";
    } else {
        dynamicParams += " " + searchParams.getOperation() + " ";
    }
    dynamicParams += " (seller like '%" + searchParams.getSellerName() + "%')";
}
```

(ii) Query: Find all records from the product data table for Product name (e.g. Apple Iphone).

```
public List<FactDomain> fetchSearchResults(SearchParams searchParams) {
    LOG.info("Fetching Advanced Search Results ");
    String baseQuery = "select * from fact_domain";
    String dynamicParams = "";

    if (StringUtils.isNotBlank(searchParams.getProductName())) {
        if (StringUtils.isEmpty(dynamicParams)) {
            dynamicParams += " where";
        } else {
            dynamicParams += " " + searchParams.getOperation() + " ";
        }
        dynamicParams += " (name like '%" + searchParams.getProductName() + "%' )";
    }
}
```

(iii) Query: Find all records from the product data table satisfying the price range (for e.g. 100\$ - 900\$) provided by the user at runtime.

```
        if (StringUtils.isNotBlank(searchParams.getMinPrice()) &&
StringUtils.isNotBlank(searchParams.getMaxPrice())) {
            if (StringUtils.isEmpty(dynamicParams)) {
                dynamicParams += " where";
            } else {
                dynamicParams += " " + searchParams.getOperation() + " ";
            }
            dynamicParams += " (price between " + searchParams.getMinPrice() + " and " +
searchParams.getMaxPrice()
                + ")";
        }
```

(iv) Query: Find all records from the product data table satisfying the date range (for e.g. “15-03-2017” to “07-04-2017”) stored by the system date:

```
        if (StringUtils.isNotBlank(searchParams.getStartDate()) &&
StringUtils.isNotBlank(searchParams.getEndDate())) {
            if (StringUtils.isEmpty(dynamicParams)) {
                dynamicParams += " where";
            } else {
                dynamicParams += " " + searchParams.getOperation() + " ";
            }
            dynamicParams += " (epoch_time between " + searchParams.getStartDateEpoch() + "
and "
                + searchParams.getEndDateEpoch() + ")";
        }

String dynamicQuery = baseQuery + dynamicParams;
dynamicQuery += " order by time desc, price asc";
```

(v) Query: Display all the query results to the GUI view (returns a ResultList object):

```
Query query = entityManager.createNativeQuery(dynamicQuery, FactDomain.class);  
List<FactDomain> resultList = query.getResultList();  
return resultList;
```

After the queries are made to get the results, transaction management is an important part of RDBMS-oriented enterprise application that is used to ensure data integrity and consistency. Our system provides transaction capabilities to POJOs. Pojo's are the java class objects that encapsulate the warehouse data in order to transport them to the presentation layer for display.

In the above example: **resultList** is the java object, which contains all the encapsulated information that is rendered for display.

### ***3.2. Possible contributions and developments:***

The implementation of WebOMiner\_SRec delivers good results with a simplistic user interface that is delivered to the end user who wants to query the extracted data stored in the data warehouse, as the WebOMiner\_S system does not provide any graphical user interface for querying and knowledge discovery. Therefore the WebOMiner\_S is observed to be a naive system with large number of possibilities for improvements. The possible contributions are:

1. Our WebOMiner\_SRec system develops annotation rules to map the column and the data values appropriately.
2. Integrating the data warehouse records by mapping the required identifiers in the fact-domain tables to provide consistency and integrity of data.
3. WebOMiner\_SRec makes the use of Java object class methods to provide correct data encapsulated in java objects to the front user (proves to be the answers to the user specific queries).

## CHAPTER 4. IMPLEMENTATION AND RESULTS

### *4.1. Where are the source code files for the system:*

(i) The entire WebOMiner\_SRec application is available at “miner.newes.uwindsor.ca”. This is a “Debian server” in Linux supported by University of Windsor to deploy Java web applications. To access it as user just type the URL “[54.69.20.208:8080/about](http://54.69.20.208:8080/about)” in your web browser.

(ii) To access it as administrator in order to modify/ improves the existing work, access the Woddlab directory file system (request access permission from Dr. Christie Ezeife) → Look for AmrutWebOMiner\_SRec (which contains the source code and a readme.txt file as a guide for getting the code running) → Download the code on your local Windows or Linux machine and follow the readme.txt document to run the WebOMiner\_SRec application.

#### *4.1.1. Which operating system supports this application:*

WebOMiner\_SRec System is developed to work with Windows and is portable in University of Windsor CDF Solaris Operating System on Unix environment and most importantly on the Debian server for deploying the system on the web. Minimum requirements to have the system running are: (i) atleast 4gb RAM, (ii) 256 gb HDD.

To make it simple, all setup tools used are compatible with both Windows and Unix operating system. In that case, only minor changes are required to run the system on Windows and on Unix. The steps to install and run the source-code for the system are explained below in section 4.2.

## **4.2. INSTALLATION AND WORKING PROCEDURE:**

1. Conform operating system requirement described in section 4.1.1. Download the source code on your computer from the Woddlab directory → AmrutWebOMiner\_SRec.
2. Install java JDK 1.6 or later version and set the class path / path in environment variable.

Setting an environment variable is discussed in the flowing link:

<http://forums.sun.com/thread.jspa?threadID=5450340>

3. For debugging and running the system we need to download and install Spring STS. Step 1. Install Spring Source Tool Suite (STS) IDE; download the latest binaries from <http://www.springsource.org/springsource-tool-suite-download>. Once you downloaded the installation, unpack the binary distribution into a convenient location.

Setup Spring Framework 4.0 Libraries

Step 2. Please download JAR files from here:

<http://repo.spring.io/milestone/org/springframework/spring/4.0.0.RC1/spring-framework-4.0.0.RC1-dist.zip>

Set your CLASSPATH variable on this directory properly otherwise you will face problem while running your application.

4. Install Maven (for building the application on Unix system):

Maven is downloadable as a zip file at “<http://maven.apache.org/download.cgi>”. Only the binaries are required, so look for the link to apache-maven-{version}-bin.zip or apache-maven-{version}-bin.tar.gz.

Once you have downloaded the zip file, unzip it to your computer. Then add the bin folder to your path.

To test the Maven installation, run mvn from the command-line:

“mvn -v”

## 5. Install My Sql latest version:

MySQL Workbench can be installed using the (.msi) installation package. The MSI package bears the name mysql-workbench-version-win32.msi, where version indicates the MySQL Workbench version number.

Step 1. Right-click the MSI file → select the Install option from the pop-up menu, or simply double-click the file.

In the Setup Type window you may choose Complete or Custom installation. To use all features of MySQL Workbench choose the complete option.

## 6. Install Tomcat version 8.0:

Step 1. Goto <http://tomcat.apache.org> ⇒ Under "Tomcat 8.5. {xx} Released"

(Where {xx} is the latest upgrade number) ⇒ Downloads ⇒ under "8.5. {xx}" ⇒ Binary Distributions ⇒ Core ⇒ "ZIP" package (e.g., "apache-tomcat-8.5.{xx}.zip", about 9 MB).

Step 2. Create your project directory; say "d:\myProject" or "c:\myProject". UNZIP the downloaded file into your project directory. Tomcat will be unzipped into directory "d:\myProject\apache-tomcat-8.0.{xx}".

Step 3. For ease of use, we shall shorten and rename this directory to:

"d:\myProject\tomcat".

Take note of Your Tomcat Installed Directory. Hereafter, refer to the Tomcat installed directory as <TOMCAT\_HOME>.

### 4.2.1. Steps for windows installation:

1. Import “WebOMiner\_SRec” in spring by using following steps:

File → Import Project → General → File System → Select “WebOMiner\_SRec” from your computer → ok. The “webominers” project will be imported to Spring STS.

2. Go to Build path button in Menu bar of the Spring STS, select and click Debug (WebOMiner\_SRec) button. If the program is imported properly a new window will be opened in the web browser with the running application.

#### **4.2.2. Steps for Unix installation:**

1. Use the command “mvn spring-boot:run” (this is a maven plugin for the Unix command line). This command includes a run goal, which can be used to launch the application directly from the command line. Running the application on the Unix environment is simpler than running it on the Windows operating system. The java main class has already been added to the source code. Therefore, the need to call the main class from the command line is not required.

#### ***4.3.HOW TO USE THE WebOMiner\_SRec System:***

Once the entire application is installed on the computer and ready to run, follow the steps below to use the WebOMiner\_SRec on the local machine or follow the link 54.69.20.208:8080/about to access it directly on the web browser:

Step 1: GUI for the home page to input product and vendor information:

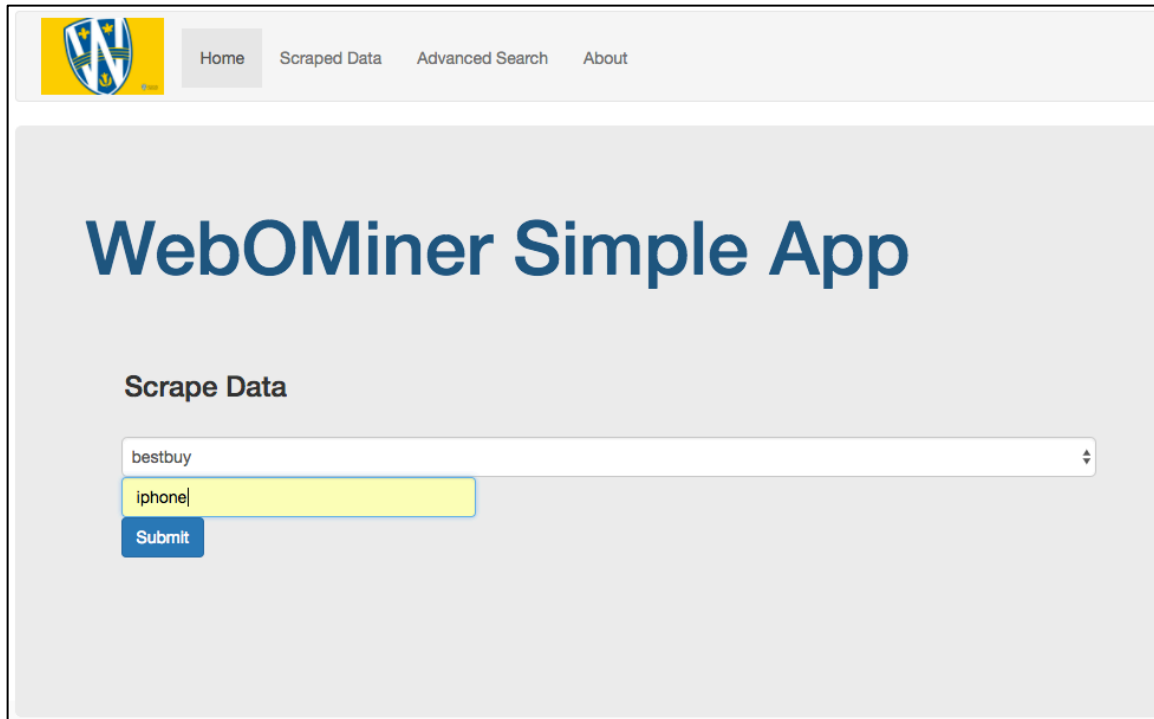
WebOMiner\_SRec system implementation includes a front-end interface that further connects to a Java servlet like Tomcat to access it on the browser.

The index page of the implementation has two action fields, one for the input seller source such as “BestBuy” and the other is product input such as “iphone” that will be used for extracting the data and for collecting records of data blocks into a data warehouse table having the following attributes:

*<ID>,<PRODUCT NAME & DESC>,<PRICE>,<SELLERINFO>,<DATE/TIME>*



Figure 19 represents a view of the home page developed for the WebOMiner\_SRec system:



The screenshot shows the home page of the WebOMiner Simple App. At the top, there is a navigation bar with a logo on the left and four links: Home, Scraped Data, Advanced Search, and About. The main content area has a large heading 'WebOMiner Simple App'. Below this heading is a section titled 'Scrape Data'. In this section, there is a dropdown menu with 'bestbuy' selected, a text input field containing 'iphone', and a blue 'Submit' button.

**Figure19: Input page to our WebOMiner\_SRec System**

On submitting the required information the records are stored in the data warehouse table that are represented on the “Scraped Data” interface which is shown in Step 2.

Step 2: Scraped data interface for displaying the data records stored in the data warehouse table.

The retrieved data records from the input fields specified in step 1 are displayed in the Scraped data interface in the form of the following schema:

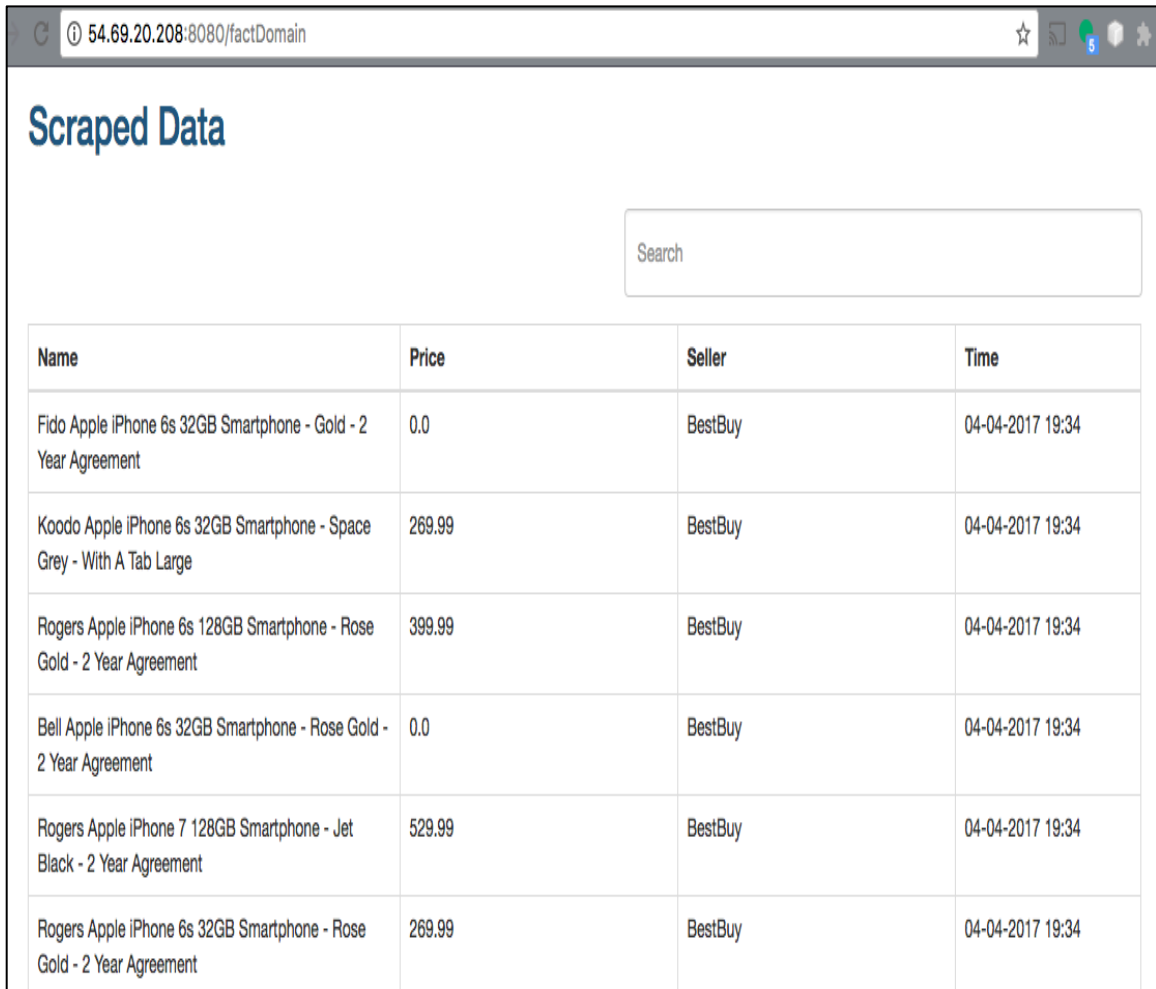
*<Name><price><seller><time>*

All the data records are perfectly placed in their respective columns to query these records is future application.

For example:

NAME	PRICE	SELLER	TIME/DATE
Apple iphone6 32gb	269.99	BestBuy	04-04-2017 (19:34)

Figure 20 represents the results on the display interface of “Scraped data”



The screenshot shows a web browser window with the address bar displaying '54.69.20.208:8080/factDomain'. The page title is 'Scraped Data'. Below the title is a search bar with the placeholder text 'Search'. Below the search bar is a table with the following data:

Name	Price	Seller	Time
Fido Apple iPhone 6s 32GB Smartphone - Gold - 2 Year Agreement	0.0	BestBuy	04-04-2017 19:34
Koodo Apple iPhone 6s 32GB Smartphone - Space Grey - With A Tab Large	269.99	BestBuy	04-04-2017 19:34
Rogers Apple iPhone 6s 128GB Smartphone - Rose Gold - 2 Year Agreement	399.99	BestBuy	04-04-2017 19:34
Bell Apple iPhone 6s 32GB Smartphone - Rose Gold - 2 Year Agreement	0.0	BestBuy	04-04-2017 19:34
Rogers Apple iPhone 7 128GB Smartphone - Jet Black - 2 Year Agreement	529.99	BestBuy	04-04-2017 19:34
Rogers Apple iPhone 6s 32GB Smartphone - Rose Gold - 2 Year Agreement	269.99	BestBuy	04-04-2017 19:34

**Figure20: Scraped data display result**

The Scraped data interface also contains a field to perform a quick search and it also displays the number of records retrieved.

For example: A quick search for “iphone 6” displays 121 records that are available in the database from all the various vendors/sellers like Amazon, BestBuy, etc.

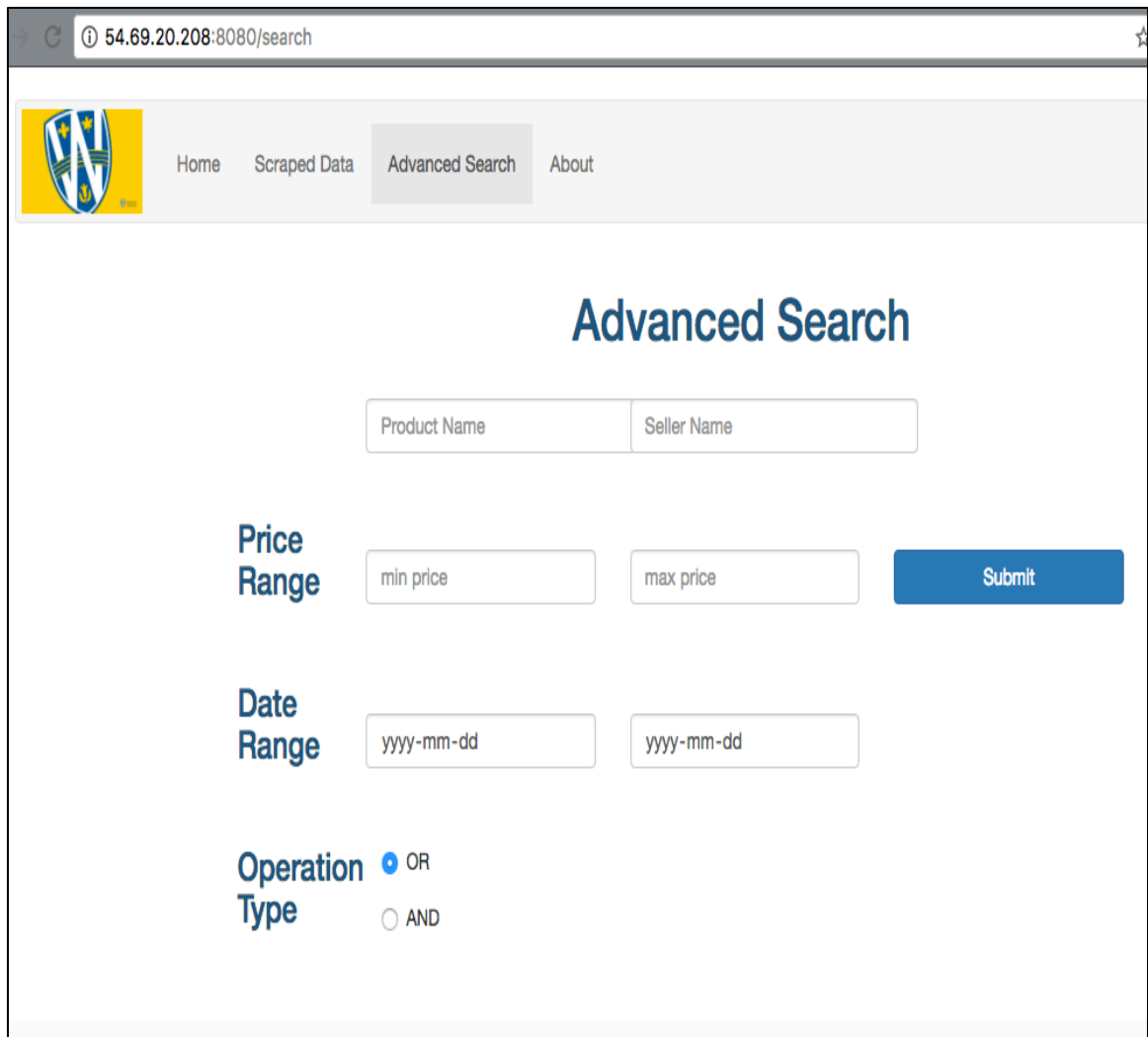
Figure 21 represents the quick search field to filter records

Scraped Data			
121 item			
<input type="text" value="iphone 6"/>			
Name	Price	Seller	Time
Apple iPhone 6 16GB Factory Unlocked GSM 4G LTE Smartphone, Space Gray (Certified Refurbished)	389.99	amazon	11-05-2017 21:00
Apple iPhone SE 16GB Factory Unlocked LTE Smartphone - Space Gray (Certified Refurbished)	399.99	amazon	11-05-2017 21:00
Apple iPhone 6 16GB 4G LTE Unlocked GSM Cell Phone - Space Gray	603.04	amazon	11-05-2017 21:00
Rogers Apple iPhone 7 Plus 128GB Smartphone - Jet Black - 2 Year Agreement	679.99	BestBuy	09-05-2017 19:34
Fido Apple iPhone 6s 32GB Smartphone - Space Grey - 2 Year Agreement	269.99	BestBuy	09-05-2017 19:34
TELUS Apple iPhone 6s 32GB Smartphone - Rose Gold - 2 Year Agreement	99.99	BestBuy	09-05-2017 19:34
Fido Apple iPhone 6s 32GB Smartphone - Gold - 2 Year Agreement	269.99	BestBuy	09-05-2017 19:34

**Figure21: Result from quick search on “iphone 6”**

Step 3: Advanced Search interface for querying to obtain required results:

Figure 22 represents the Advanced Search feature that allows the user to query any kind of historical information from the database records. This information can be queried based on the following search criteria's: Product name, Seller name, Price, Date, Time, additional information displayed such as “product details”.

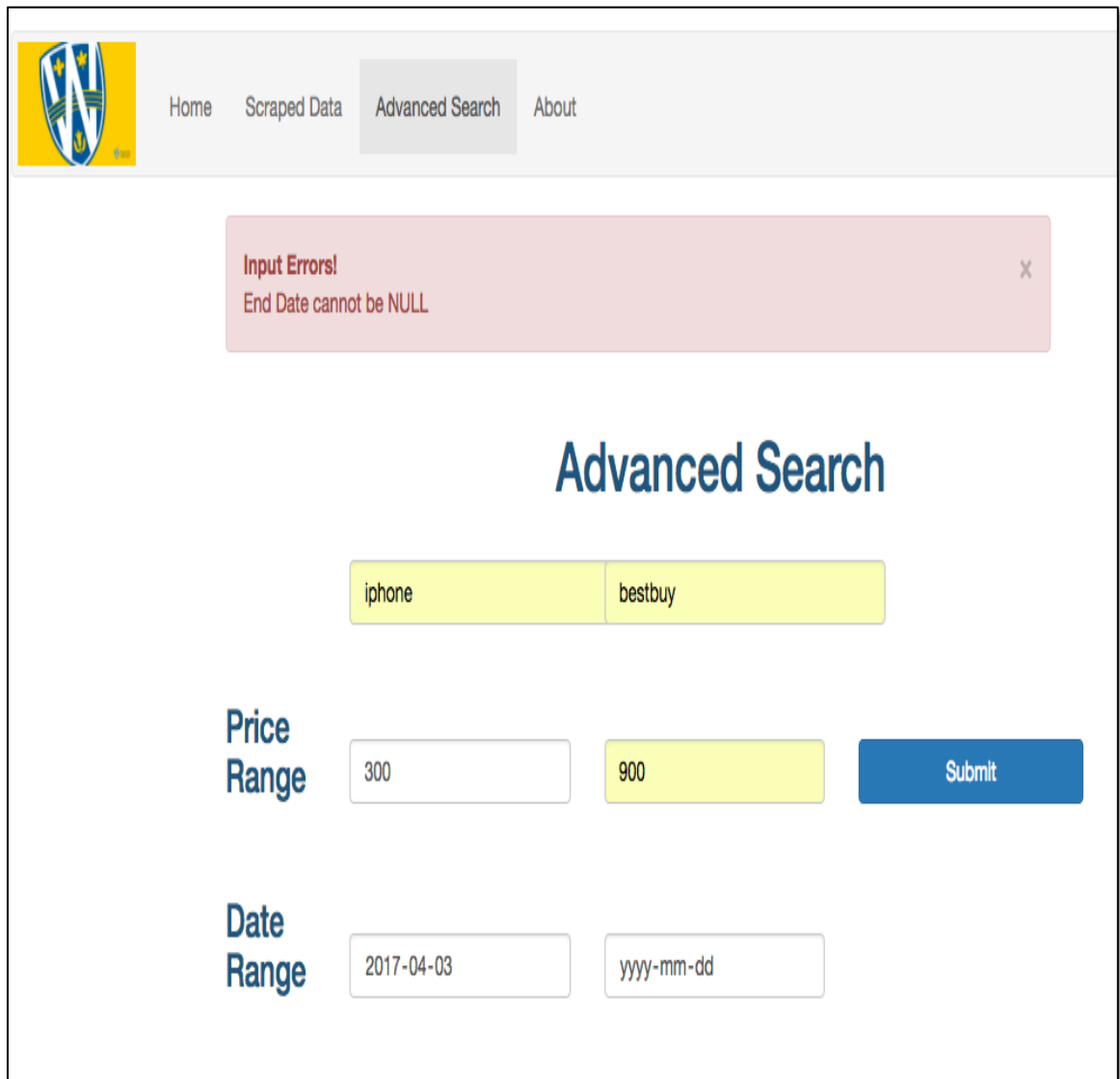


The screenshot shows a web browser window with the address bar displaying "54.69.20.208:8080/search". The page features a navigation bar with a logo and links for "Home", "Scraped Data", "Advanced Search" (which is highlighted), and "About". The main heading is "Advanced Search". Below this, there are input fields for "Product Name" and "Seller Name". The "Price Range" section includes labels for "min price" and "max price" next to their respective input fields, and a blue "Submit" button. The "Date Range" section has two input fields with the placeholder "yyyy-mm-dd". The "Operation Type" section has two radio buttons: "OR" (which is selected) and "AND".

**Figure 22: Advanced Search**

The Advanced search has the feature of error correction, which helps to correct wrong inputs provided by the user. An example would be “date range cannot have a low value date in “Todate” field than the “Fromdate” field OR “it cannot have null values”

Figure 23 represents the use of error correction as follows:



The screenshot displays a web application's 'Advanced Search' page. At the top, there is a navigation bar with a logo on the left and links for 'Home', 'Scraped Data', 'Advanced Search' (which is highlighted), and 'About'. Below the navigation bar, a red error message box is visible, stating 'Input Errors!' and 'End Date cannot be NULL'. The main heading 'Advanced Search' is centered. Below the heading, there are two input fields for search terms: 'iphone' and 'bestbuy'. Further down, there are sections for 'Price Range' and 'Date Range'. The 'Price Range' section has two input fields: '300' and '900', followed by a blue 'Submit' button. The 'Date Range' section has two input fields: '2017-04-03' and 'yyyy-mm-dd'.

**Figure 23: Advanced search for error free inputs**

Advanced search example:

The following figure displays results on the basis of input filters provided as:

Product: iphone

Price range: 0\$ to 1350\$

Date range: 03-04-2017 to 04-04-2017

**Result: Apple iphone 6s gold, 32gb, Vendor: Fido-BestBuy, on date: 04-04-2017**

Figure 24 represents the results obtained from advanced search tab:

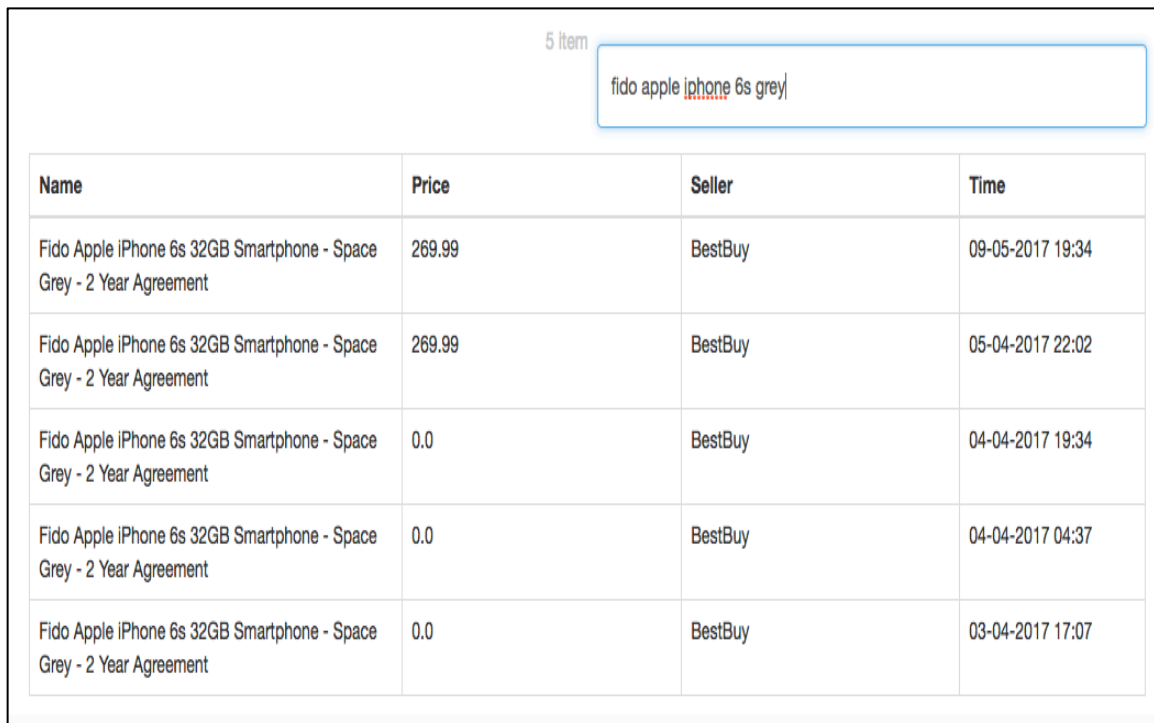
The screenshot displays an advanced search interface. At the top, there is a search bar with 'iphone' entered. Below this, there are sections for 'Price Range' (0 to 1350), 'Date Range' (2017-04-03 to 2017-04-04), and 'Operation Type' (OR selected). A 'Submit' button is present. Below the filters, there is a text input field with the placeholder 'What're you looking for?'. At the bottom, a table displays the search results.

Name	Price	Seller	Time
Fido Apple iPhone 6s 32GB Smartphone - Gold - 2 Year Agreement	0.0	BestBuy	04-04-2017 19:34
Bell Apple iPhone 6s 32GB Smartphone - Rose Gold - 2 Year Agreement	0.0	BestBuy	04-04-2017 19:34

**Figure24: Advanced search results**

From the results we observe that the following seller “Best buy” can have iPhones from different providers as seen in the display tab. Therefore, we can further query these results as follows:

Example query: Figure 25 filters all records from Bestbuy with Fido service for iphone 6s which is grey in color.



The screenshot shows a search interface with a search bar containing the text "fido apple iphone 6s grey". Below the search bar, there is a table with 5 items. The table has four columns: Name, Price, Seller, and Time. The data rows show five records for "Fido Apple iPhone 6s 32GB Smartphone - Space Grey - 2 Year Agreement" from "BestBuy" with prices of 269.99, 269.99, 0.0, 0.0, and 0.0, and timestamps from 03-04-2017 to 09-05-2017.

Name	Price	Seller	Time
Fido Apple iPhone 6s 32GB Smartphone - Space Grey - 2 Year Agreement	269.99	BestBuy	09-05-2017 19:34
Fido Apple iPhone 6s 32GB Smartphone - Space Grey - 2 Year Agreement	269.99	BestBuy	05-04-2017 22:02
Fido Apple iPhone 6s 32GB Smartphone - Space Grey - 2 Year Agreement	0.0	BestBuy	04-04-2017 19:34
Fido Apple iPhone 6s 32GB Smartphone - Space Grey - 2 Year Agreement	0.0	BestBuy	04-04-2017 04:37
Fido Apple iPhone 6s 32GB Smartphone - Space Grey - 2 Year Agreement	0.0	BestBuy	03-04-2017 17:07

**Figure 25: filter search result**

In the above figure we can observe that the user can input any query such as “fido apple smartphone grey”. It need not be the exact same word picked from the data record. Therefore, it is possible to compare a product specification or the product price between two sources like “BestBuy and Costco”.

For example:

If a customer is searching for a laptop specifically having an “i5 processor” then he needs to compare the following criteria:

Brand information: dell, Lenovo, Acer, apple, etc.

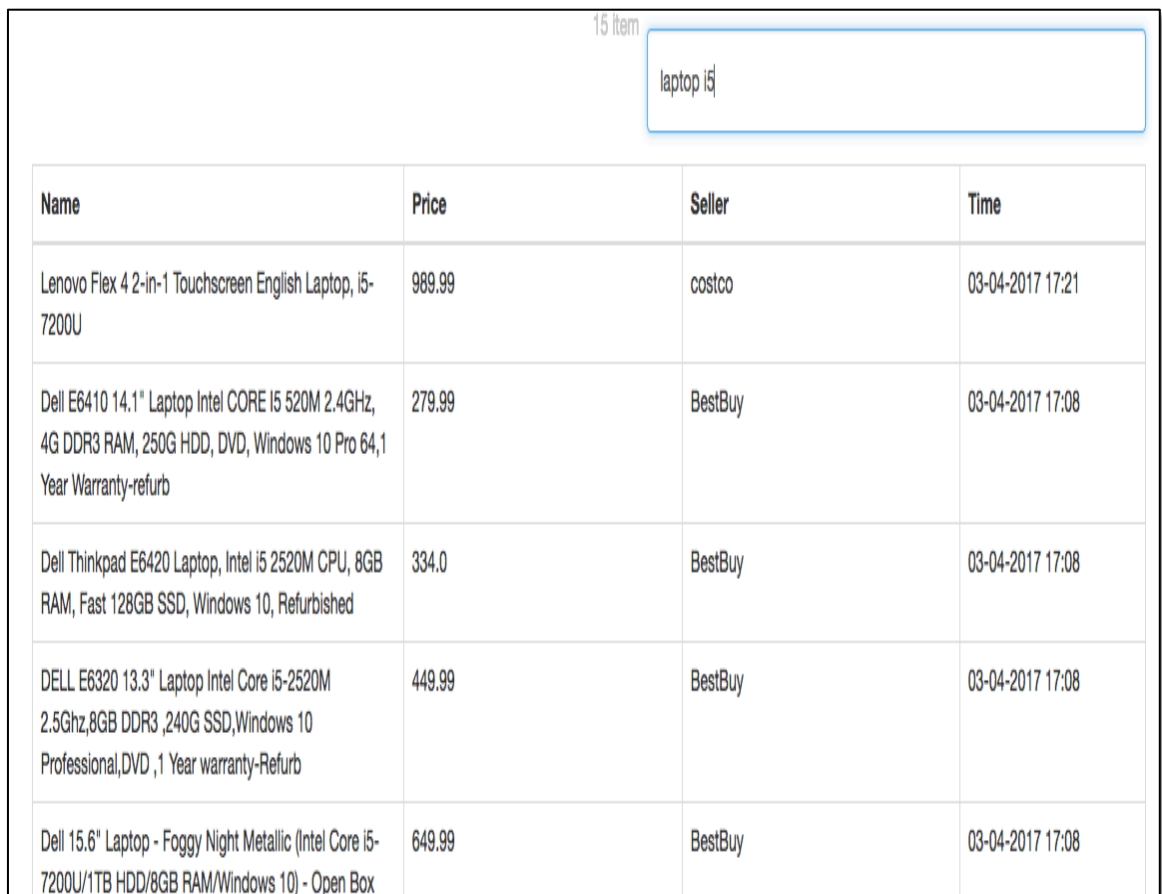
Other specifications: Memory HDD, Ram, etc.

Price: reasonable price and quality requirements.

Seller info: BestBuy, costco, etc.

**Result: Dell E6410 is the cheapest laptop for 279.99\$ from BestBuy**

Figure 26 represents the results obtained by comparing two or more different vendors to find results for laptops with i5 processor for the cheapest price.



15 item

laptop i5

Name	Price	Seller	Time
Lenovo Flex 4 2-in-1 Touchscreen English Laptop, i5-7200U	989.99	costco	03-04-2017 17:21
Dell E6410 14.1" Laptop Intel CORE I5 520M 2.4GHz, 4G DDR3 RAM, 250G HDD, DVD, Windows 10 Pro 64,1 Year Warranty-refurb	279.99	BestBuy	03-04-2017 17:08
Dell Thinkpad E6420 Laptop, Intel i5 2520M CPU, 8GB RAM, Fast 128GB SSD, Windows 10, Refurbished	334.0	BestBuy	03-04-2017 17:08
DELL E6320 13.3" Laptop Intel Core i5-2520M 2.5Ghz,8GB DDR3 ,240G SSD,Windows 10 Professional,DVD ,1 Year warranty-Refurb	449.99	BestBuy	03-04-2017 17:08
Dell 15.6" Laptop - Foggy Night Metallic (Intel Core i5-7200U/1TB HDD/8GB RAM/Windows 10) - Open Box	649.99	BestBuy	03-04-2017 17:08

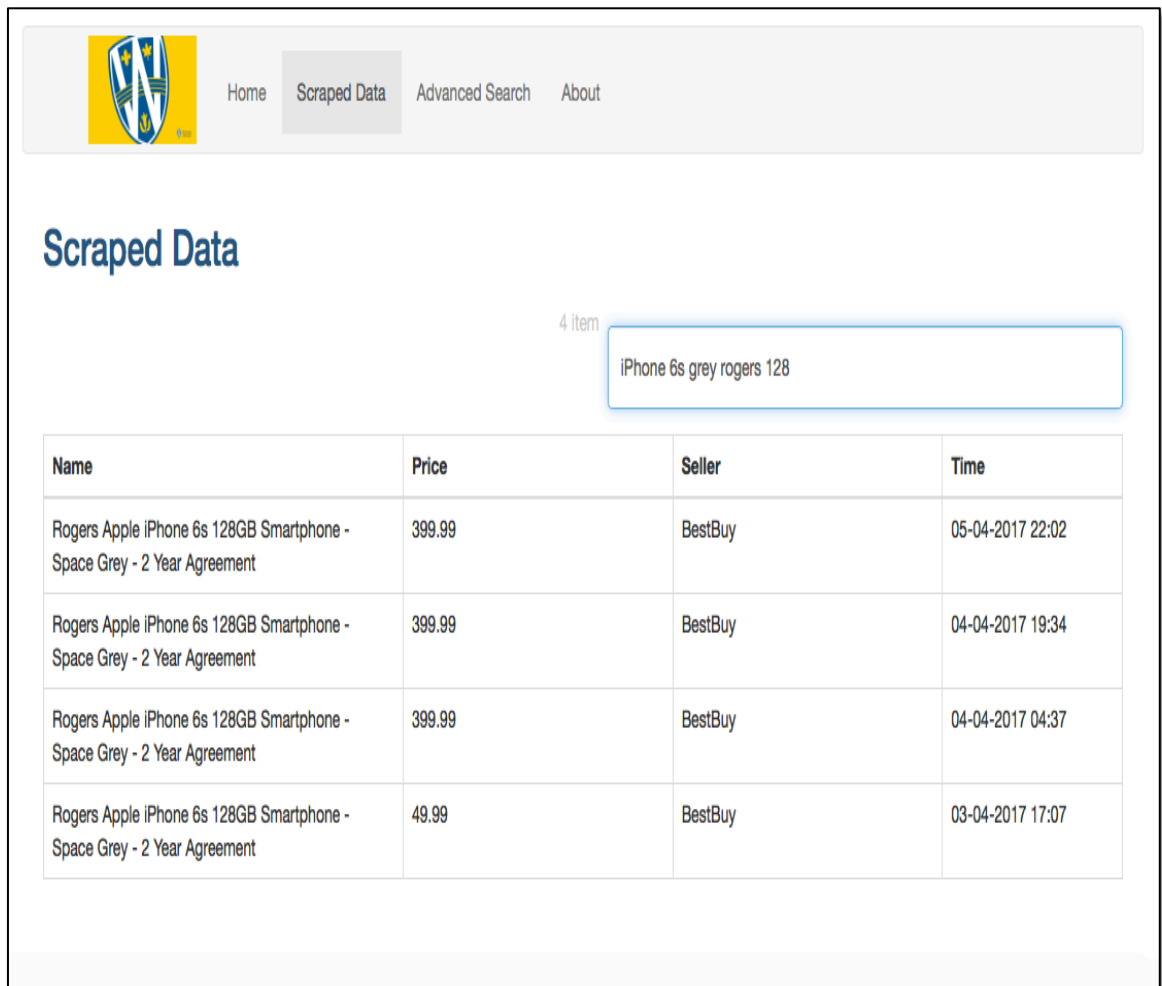
**Figure 26: Comparative Search results**



### Example Historical Querying:

The WebOMiner\_SRec system uses datawarehouse tables to store records and query historical data. This application has the date/time attribute in order to retrieve historical information from the present database records.

Figure 27 represents the result when the same web-source, in our case “Best Buy” changes its product information over a time period.



The screenshot displays the WebOMiner\_SRec application interface. At the top, there is a navigation bar with a logo on the left and links for Home, Scraped Data (which is highlighted), Advanced Search, and About. Below the navigation bar, the title "Scraped Data" is prominently displayed. To the right of the title, it indicates "4 item" and shows a search input field containing the text "iPhone 6s grey rogers 128". Below this, a table presents the historical search results. The table has four columns: Name, Price, Seller, and Time. It contains four rows of data, all for the same product (Rogers Apple iPhone 6s 128GB Smartphone - Space Grey - 2 Year Agreement) sold by BestBuy at a price of 399.99, but at different times and prices over a period of several days in April 2017.

Name	Price	Seller	Time
Rogers Apple iPhone 6s 128GB Smartphone - Space Grey - 2 Year Agreement	399.99	BestBuy	05-04-2017 22:02
Rogers Apple iPhone 6s 128GB Smartphone - Space Grey - 2 Year Agreement	399.99	BestBuy	04-04-2017 19:34
Rogers Apple iPhone 6s 128GB Smartphone - Space Grey - 2 Year Agreement	399.99	BestBuy	04-04-2017 04:37
Rogers Apple iPhone 6s 128GB Smartphone - Space Grey - 2 Year Agreement	49.99	BestBuy	03-04-2017 17:07

**Figure 27: Historical search results**

For example:

In the above figure,

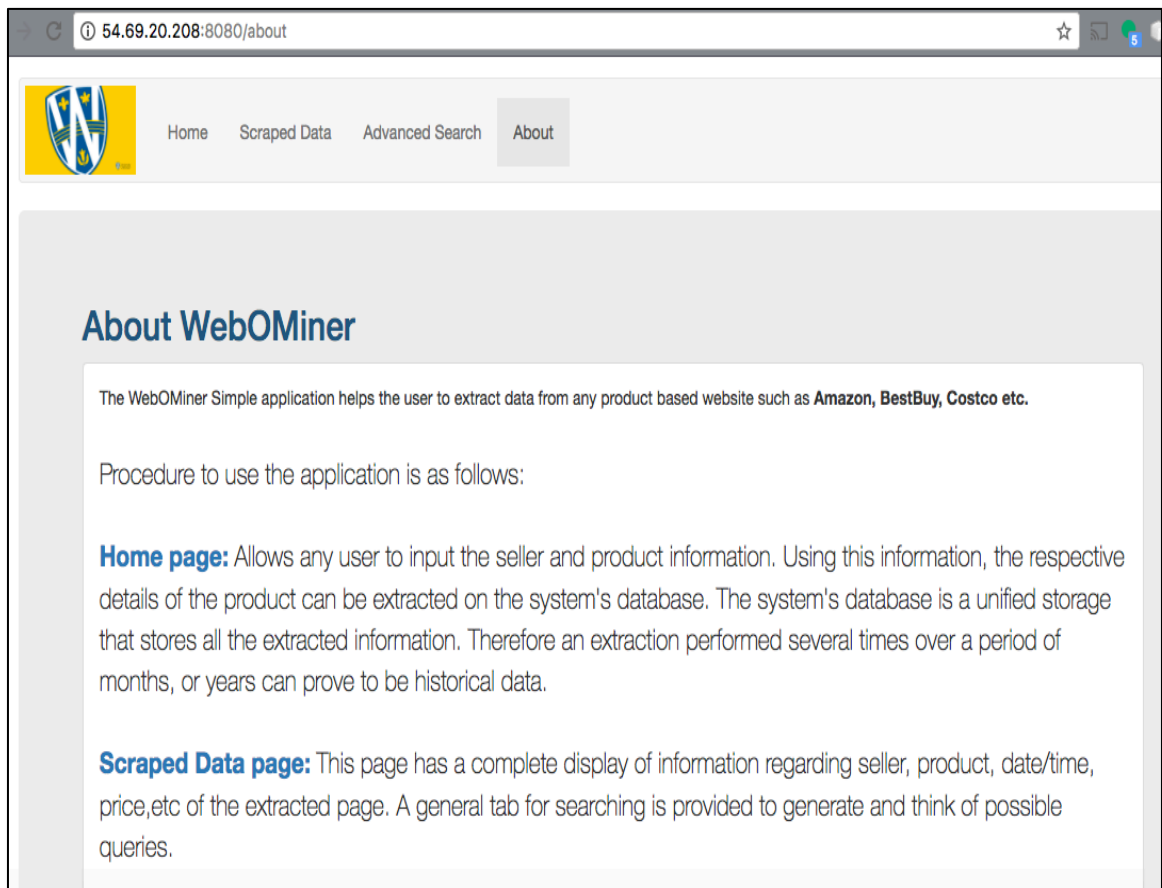
Iphone 6s 128 gb price on 03-04-2017 → 49.99\$

Iphone 6s 128 gb price on 04-04-2017 → 399.99\$

**Result: price for iphone 6s by BestBuy was different on 03-04-2017 and on 04-04-2017.**

**WebOMiner\_SRec system therefore helps in storing and querying historical data records.**

WebOMiner\_SRec application also has the “Information guide” as to understand how the system works. It is a user friendly application supported for data querying and analysis. It also has links directed to all the tabs provided on our system. The guide page looks as follows:



**Figure 28: About page guide**

#### **4.4.Comparing WebOMiner\_SRec with the existing web data extraction systems and the WebOMiner\_S:**

1. WebOMiner\_SRec extends the WebOMiner\_S for adding a GUI and querying features that is mounted on the web. It also provides a way to store extracted data records into a single data warehouse table rather than creating new tables for every extraction. This helps in integrating those data records for data comparison, knowledge discovery and historical querying.
2. WebOMiner\_SRec develops a GUI interace, which is simple to use, and is equally robust as other web data extractors available on the Internet such as “Googleshop, NextTag, etc.”
3. The WebOMiner\_SRec allows the users to query historical data, which is not a feature provided by “Googleshop” or any such existing application. It also allows the user to make queries specifying any user inputs. Whereas, existing applications provide only a limited scope for querying data records.

## **CHAPTER 5: CONCLUSION AND FUTURE WORK**

This thesis proposes the Implementation and mounting of the WebOMiner\_Simple system on a Web Server. Extending the WebOMiner\_S data extraction system with a Web GUI for querying extracted E-Commerce data from multiple data sources through UNIX server for web access and availability. Extending the WebOMiner\_S data extraction system such that the extracted product schemas previously extracted from the web, which were stored in the database as generic schema have now been given more meaningful names and all the records are stored in a single data warehouse table for comparative and historical querying. The WebOMiner\_SRec extends the WebOMiner\_S data extraction system such that it can directly answer comparative historical queries on the web through the GUI in much the same way as existing recommendation systems on the web such as GoogleShop and ShopZilla do. It is also as robust as GoogleShop and other similar applications. This system has a scope to extend data extraction from more web sources in the future as a separate table is created to store only the web source information that is integrated with the datawarehouse table. It also has scope for future development in terms of providing meaningful recommendations to the system users.

### ***5.1 Future Work***

We feel that our effort of mining the product data records automatically in a simplistic way has a lot of room for improvement.

1. Now that we have created the data warehouse for products, further mining and querying functionalities are to be extended to analyze the database records and to perform comparative analysis based on a simpler way for querying such as search engines implemented using “Apache Solr” which is an intelligent search engine that can catch any word or sentence or even a synonym of the records that need to be fetched from the applications database.

2. Our application is only capable of extracting the meta-data available on the “product list page”. It can be modified and extended to extract complete product specifications and details from the “detail product page”.
3. One of the main limitations of our system is that we have implemented our system knowing the fact that certain big websites such as “walmart” do not allow the permission to extract data. Work needs to be done to solve this problem.
4. Another important application for future scope is to have user specific data extraction. Where the system can automatically determine which web sources or which kind of products does the user usually search for knowledge discovery. Our system can be further designed to analyze such patterns and automatically extract this information over a period of time.

## REFERENCES

- Annoni, E., & Ezeife, C. (2009). Modeling Web Documents as Objects for Automatic Web Content Extraction-Object-oriented Web Data Model. *ICEIS (1)*, (pp. 91-100).
- Arthur, J., & Azadegan, S. (2005, May). Spring Framework for rapid open source J2EE Web Application Development: A case study. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2005 and First ACIS International Workshop on Self-Assembling Wireless Networks. SNPD/SAWN 2005. Sixth International Conference on* (pp. 90-95). IEEE.
- Bhavik, P. (2010). A survey of the comparison shopping agent-based decision support systems. *Journal of Electronic Commerce Research*, 178.
- Bernstein, P. A., & Newcomer, E. (2009). *Principles of transaction processing*. Morgan Kaufmann, 12-18.
- Bharanipriya, V., & Prasad, V. K. (2011). Web content mining tools: a comparative study. *International Journal of Information Technology and Knowledge Management*, 4(1), 211-215.
- Chakrabarti, S. (2000). Data mining for hypertext: A tutorial survey. *ACM SIGKDD Explorations Newsletter*, 1-11.
- Chang, C.-H., Kayed, M., Girgis, M., & Shaala, K. (2006). A survey of web information extraction systems. *Knowledge and Data Engineering, IEEE Transactions on*, 1411-1428.
- Codd, E. F., Codd, S. B., & Salley, C. T. (1993). Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate. *Codd and Date*, 32.
- Cooley, R., Mobasher, B., & Srivastava, j. (1997). Web mining: Information and pattern discovery on the world wide web. *Tools with Artificial Intelligence, 1997. Proceedings., Ninth IEEE International Conference on* (pp. 558-567). IEEE.
- Eagle, M. (2004). Wiring your web application with open source java. 2004-04-07. <http://www.onjava.com/pub/a/onjava/2004/04/07/wiringwebapps.html>.

- Escoffier, C., & Hall, R. S. (2007, March). Dynamically adaptable applications with iPOJO service components. In *International conference on Software composition* (pp. 113-128). Springer Berlin Heidelberg.
- Escoffier, C., Hall, R. S., & Lalanda, P. (2007, July). iPOJO: An extensible service-oriented component framework. In *Services Computing, 2007. SCC 2007. IEEE International Conference on* (pp. 474-481). IEEE.
- Ezeife, C., & Mutsuddy, T. (2012). Towards comparative mining of web document objects with NFA: WebOMiner system. *International Journal of Data Warehousing and Mining (IJDWM)*, 1-21.
- Ezeife, C. I., & Peravali, B. (2016, July). Comparative Mining of B2C Web Sites by Discovering Web Database Schemas. In *Proceedings of the 20th International Database Engineering & Applications Symposium* (pp. 183-192). ACM.
- Gupta, P., & Govil, M. C. (2010). MVC Design Pattern for the multi framework distributed applications using XML, spring and struts framework. *Int J Comput Sci Eng*, 2(4), 1047-1051.
- Hammer, J., McHugh, J., McHugh, J., & Garcia-Molina, H. (1997). Semistructured Data: The TSIMMIS Experience. In *Proceedings of the 1st East-European*, 1-7.
- Hammer, Joachim, J., McHugh, J., & Garcia-Molina, H. (1997). emistructured Data: The TSIMMIS Experience. *Advances in Databases and Information Systems*.
- Hogue, A., & Karger, D. (2005). Thresher: automating the unwrapping of semantic content from the World Wide Web. *Proceedings of the 14th international conference on World Wide Web* (pp. 86-95). ACM.
- Hsu, C.-N., & Dung, M.-T. (1998). Generating finite-state transducers for semi-structured data extraction from the web. *Information systems*, 521-538.
- Hüsemann, B., Lechtenbörger, J., & Vossen, G. (2000). *Conceptual data warehouse design* (pp. 6-1). Universität Münster. Angewandte Mathematik und Informatik. Kosala, R., & Blockeel, H. (2000). Web mining research: A survey. *ACM Sigkdd Explorations Newsletter*(1), 1-15.

- Inmon, W. H. (1996). The data warehouse and data mining. *Communications of the ACM*, 39(11), 49-51. Inmon, W. H. (1996). The data warehouse and data mining. *Communications of the ACM*, 39(11), 49-51.
- Inmon, W. H., Imhoof, C., & Battas, G. (1996). Building the Operational Data Store, John Wiley & Sons.
- Johnson, R. (2005). J2EE development frameworks. *Computer*, 38(1), 107-110.
- Kosala, R., & Blockeel, H. (2000). Web mining research: A survey. *ACM Sigkdd Explorations Newsletter*(1), 1-15.
- Ladley, J. (1997). Operational data stores: building an effective strategy. *Data Warehouse: Practical Advice from the Experts* (Englewood Cliffs, NJ: Prentice Hall, 1997).
- Laender, A., Ribeiro-Neto, B., & da Silva, A. (2002). DEByE—data extraction by example. *Data & Knowledge Engineering*, 121-154.
- Laender, A., Ribeiro-Neto, B., da Silva, A., & Teixeira, J. (2002). A brief survey of web data extraction tools. *ACM Sigmod Record*, 84-93.
- Lee, G. H., & Jung, J. (2007). Web framework with Java and XML in multi-tiers for productivity. *Future Generation Computer Systems*, 23(2), 263-268.
- Liu, B. (2007). *Web data mining: exploring hyperlinks, contents, and usage data*. Springer Science & Business Media.
- Liu, B., & Chen-Chuan-Chang, K. (2004). Editorial: special issue on web content mining. *Acm Sigkdd explorations newsletter*, 6(2), 1-4.
- March, S. T., & Hevner, A. R. (2007). Integrated decision support systems: A data warehousing perspective. *Decision Support Systems*, 43(3), 1031-1043.
- Meng, X., Hu, D., & Li, C. (2003). Schema-guided wrapper maintenance for web-data extraction. *Proceedings of the 5th ACM international workshop on Web information and data management* (pp. 1-8). ACM.



- Muslea, I., Minton, S., & Knoblock, C. (2001). Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 93-114.
- Muslea, I., Minton, S., & Knoblock, C. (1999). A hierarchical approach to wrapper induction. *Proceedings of the third annual conference on Autonomous Agents* (pp. 190-197). ACM.
- Novotny, R., Vojtas, P., & Maruscak, D. (2009). Information extraction from web pages. *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology-Volume 03* (pp. 121-124). IEEE Computer Society.
- Razina, E., & Janzen, D. S. (2007). Effects of dependency injection on maintainability, 53-84.
- Srivastava, J., Cooley, R., Deshpande, M., & Tan, P.-N. (2000). Web usage mining: Discovery and applications of usage patterns from web data. *ACM SIGKDD Explorations Newsletter*, 1(2), 12-23.
- Wang, J., & Lochovsky, F. (2003). Data extraction and label assignment for web databases. *Proceedings of the 12th international conference on World Wide Web* (pp. 187-196). ACM.
- Wu, S.-T., Li, Y., Xu, Y., Pham, B., & Chen, P. (2004). Automatic pattern-taxonomy extraction for web mining. *Web Intelligence, 2004. WI 2004. Proceedings. IEEE/WIC/ACM International Conference on* (pp. 242-248). IEEE.
- Zaiane, O., & Han, J. (1998). Webml: Querying the world-wide web for resources and knowledge. *Proc. ACM CIKM'98 Workshop on Web Information and Data Management (WIDM'98)*. Citeseer.
- Zhai, Y., & Liu, B. (2005). Web data extraction based on partial tree alignment. *Proceedings of the 14th international conference on World Wide Web* (pp. 76-85). ACM.
- Zhou, M., & Zhou, S. (2007, June). Internet, open-source and power system simulation. In *Power Engineering Society General Meeting, 2007. IEEE* (pp. 1-5). IEEE.
- Zhu, J., Nie, Z., Wen, J.-R., Zhang, B., & Ma, W.-Y. (2006). Simultaneous record detection and attribute labeling in web data extraction. *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 494-503). ACM.

## **APPENDIX - A**

# **WebOMiner\_SRec System Manual**

Developed by Amrutraj Chachad

## TABLE OF CONTENTS

1.0	System Architecture .....	84
2.0	User Interface .....	86
	2.1 How to debug, run the program and get the result.....	87
3.0	Operating System .....	88
4.0	Programming Environment .....	88
5.0	Installation of the system.....	88
	5.1. Installation of Spring STS.....	88
	5.2. Installation of MySql workbench.....	89
6.0	Data Base, Schema and File format.....	90
7.0	Tomcat Installation .....	91
8.0	Information regarding Web Application .....	91

## 1.0 System Architecture

Our WebOMiner\_SRec system architecture consists of three modules: Presentation layer, middle tier and data access layer. The overall architecture of the Web recommendation system is shown in figure 1 below:

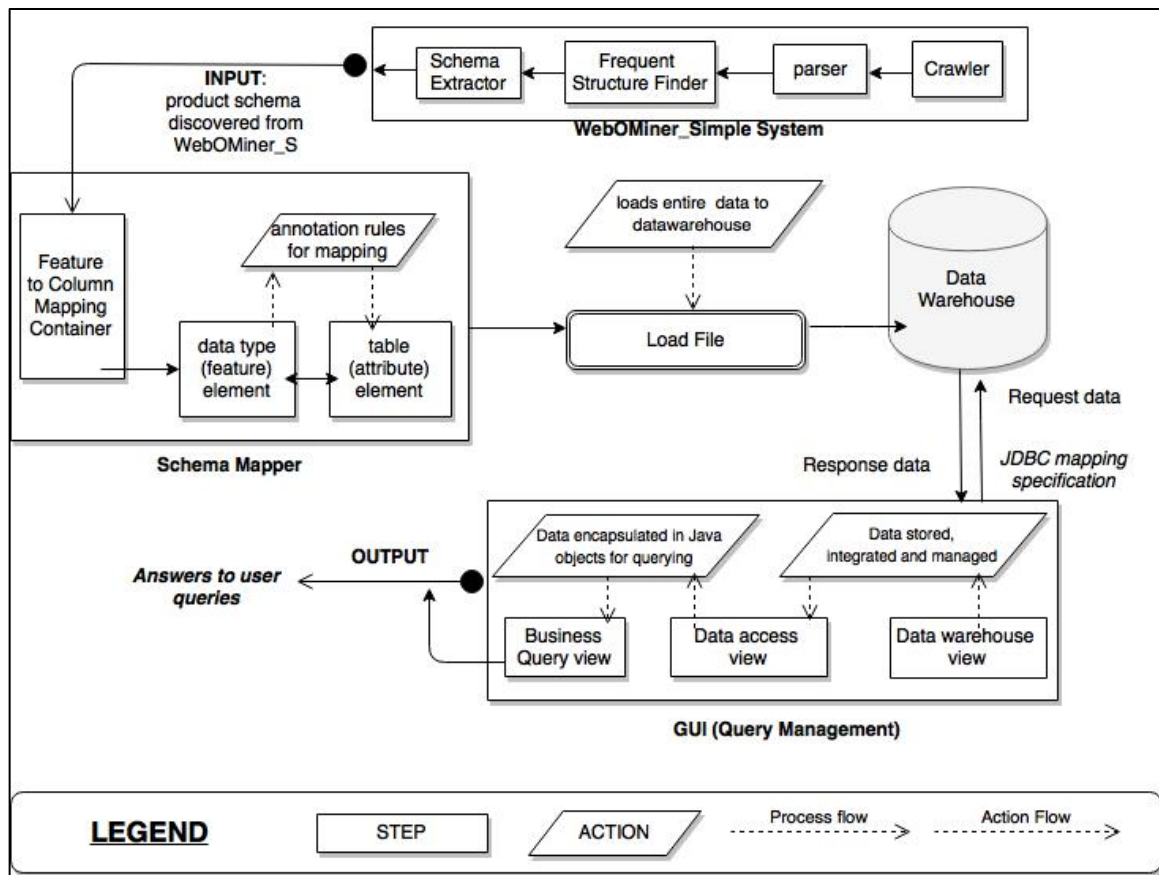
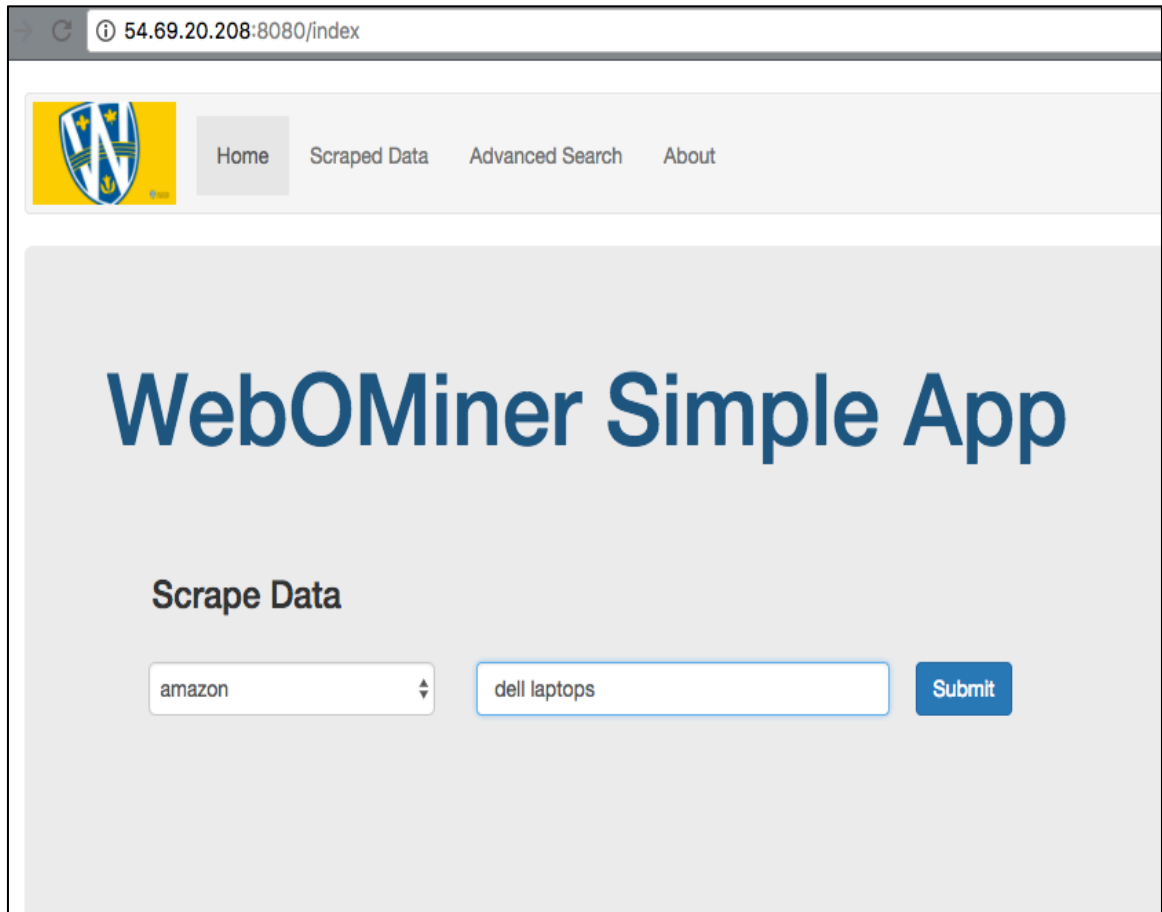


Figure 6: Architecture of Web Recommendation system

Our systems aims at developing a Simplistic user interface by mounting the WebOMiner\_SRec system on the web domain using Java Enterprise functions. The WebOMiner\_S is a web data extraction system for extracting all essential information about products. The WebOMiner\_S on its final stage outputs the schema for product attributes, which is then taken as an input for our WebOMiner\_SRec system. The WebOMiner\_SRec has a module called “Schema\_mapper” for naming all the columns present in the product\_schema provided by the WebOMiner\_S system. It then creates a mapping container that provides an environment based on annotators to map all the column attributes to the appropriate data values. Annotators are the rules or conditions written in a <scriptlet> format that contains an annotation tree to map the column attributes to their exact data types. A Load file element is introduced to perform the transport of all the mapped data to the data warehouse.

The data warehouse creates two tables to store the product and seller information. Therefore, the final step of the WebOMiner\_SRec is to create perfect non-terminating connection between the data warehouse and the interface fields. A connection is created calling the JDBC session connection methods to request data from the warehouse. Java object class method () handles the transaction processing for providing the perfect response to the requested query. The final output of the WebOMiner\_SRec gives the results to the web users based on their queries.

## 2.0 User Interface



The index page of the implementation has two action fields, one for the input seller source such as “BestBuy” and the other is product input such as “iPhone” that will be used for extracting the data and storing it in the database. User has to call the URL “54.69.20.208:8080/about” from a web browser such as Google chrome to run the application.

### ***2.1. How to debug, run the program on windows using ide and get the results:***

To debug and run the system we need to confirm the following first:

1. Conform operating system requirement described in section 3.0.
2. Install java JDK 1.6 or later version and set the class path / path in environment variable.  
Setting an environment variable is discussed in “forums.sun.java” in the flowing link:  
<http://forums.sun.com/thread.jspa?threadID=5450340>
3. For debugging and running the system from STS we need to download and install Spring STS as described in section 5.1.
4. Import “webominers” project in Spring by using following steps:  
File → Import Project → General → File System → select “webominers” from your computer → ok. The “webominers” project will be imported to Spring STS.
5. Install My Sql latest version as described in section 5.2.
6. Install Tomcat version 8.0 as described in section 8.0.
7. Make sure all the jar files needed 1. mysql-connector-java5.1.38-bin.jar 2. commons-lang3-3.4.jar 3. commons-lang3-3.4-javadoc.jar 4. guava-16.0.1.jar 5. org-apache-commons-lang.jar` driver file is in “libraries” folder under “webominers” package.
8. Go to Build path button in Menu bar of the Spring STS, select and click Debug (webominers) button as shown in figure 3. If the program is imported properly a new window will be opened in the web browser with the running WebOMiner\_SRec application.

### **3.0. Operating System:**

System is developed in 64-bit Windows 10 operating system at Intel i5 2.26 GHz processor, 8.00 GB RAM Lenovo Thinkpad Machine. This system is portable in University of Windsor CDF Solaris Operating System on Unix environment. In that case, to compile, execute and run the program type “mvn spring-boot: run” on the Unix terminal command line.

### **4.0 Programming Environment**

The application is programmed and tested in Java 2 Platform Standard Edition version 1.6.0\_16, which is installed on Windows and Unix systems. The advantages of this environment are as follows:

- Regular expression capability.
- The improved Java Doc.
- Low coupling and usability.
- Relatively easy implementation of Object-Oriented design pattern.

### **5.0 Installation of WebOMiner\_SRec system:**

#### ***5.1. Installation of Spring (STS) IDE:***

##### **5.1.1. Installing the Software Bundle on Microsoft Windows:**

Step 1 - Setup Java Development Kit (JDK):

Download latest java (JDK ) version from oracle's java site.

Set to the class path for this JDK version in case Window OS.

- set PATH=C:\jdk1.6.0\_23\bin;%PATH%
- set JAVA\_HOME=C:\jdk1.6.0\_23



Alternatively, on Windows NT/2000/XP, you could also right-click on My Computer, select Properties, then Advanced, then Environment Variables. Then, you would update the PATH value and press the OK button.

To install SpringSource Tool Suite (STS) IDE, download the latest SpringSource Tool binaries from <http://www.springsource.org/springsource-tool-suite-download> . Once you downloaded the installation, unpack the binary distribution into a convenient location.

#### Step 2 - Setup Spring Framework 4.0 Libraries

If you are looking to download the JAR files required to run the Spring 4.0 examples, please download it from here:

<http://repo.spring.io/milestone/org/springframework/spring/4.0.0.RC1/spring-framework-4.0.0.RC1-dist.zip>

Set your CLASSPATH variable on this directory properly otherwise you will face problem while running your application.

### ***5.2 Installation of the MySql software using installer:***

MySQL Workbench can be installed using the Installer (.msi) installation package. The MSI package bears the name mysql-workbench-version-win32.msi, where version indicates the MySQL Workbench version number.

To install MySQL Workbench,

right-click the MSI file → select the Install option from the pop-up menu, or simply double-click the file.

In the Setup Type window you may choose Complete or Custom installation. To use all features of MySQL Workbench choose the Complete option.

## 6.0 Data Base Schema and format

The WebOMiner\_SRec system implementation defines the use of a Data Warehouse as a tool to design relations between database records, as the main purpose of this application is to query historical information for analysis.

There are four web sources tested in this application (Amazon, bestbuy, Costco, craigslist). These are called the seller fields and the product table records are shown in the figure below:

TABLES		Search: <input type="text" value="id"/>		<input type="text" value=""/>		<input type="text" value="Q Search"/>					
fact_domain	id	comments	currency	name	original_link	price	rating	seller	ship_info	time	
seller	1	NULL	NULL	Bell Apple iPhone 6s 32GB Smartphone - Space Grey - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	2	NULL	NULL	Rogers Apple iPhone 6s 32GB Smartphone - Space Grey - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	3	NULL	NULL	Fido Apple iPhone 6s 32GB Smartphone - Space Grey - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	4	NULL	NULL	Rogers Apple iPhone 6s 128GB Smartphone - Space Grey - 2 Year Agreement	NULL	49.99	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	5	NULL	NULL	Fido Apple iPhone 6s 32GB Smartphone - Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	6	NULL	NULL	Bell Apple iPhone 6s 32GB Smartphone - Rose Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	7	NULL	NULL	TELUS Apple iPhone 6s 32GB Smartphone - Rose Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	8	NULL	NULL	Rogers Apple iPhone 6s 128GB Smartphone - Rose Gold - 2 Year Agreement	NULL	49.99	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	9	NULL	NULL	Koodo Apple iPhone 6s 32GB Smartphone - Space Grey - With A Tab Large	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	10	NULL	NULL	Rogers Apple iPhone 6s 32GB Smartphone - Rose Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	11	NULL	NULL	Rogers Apple iPhone 6s 128GB Smartphone - Gold - 2 Year Agreement	NULL	49.99	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	12	NULL	NULL	TELUS Apple iPhone 6s 32GB Smartphone - Space Grey - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	13	NULL	NULL	Virgin Mobile Apple iPhone 6s 32GB Smartphone - Space Grey - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	14	NULL	NULL	Fido Apple iPhone 6s 32GB Smartphone - Rose Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	15	NULL	NULL	Rogers Apple iPhone 6s 128GB Smartphone - Silver - 2 Year Agreement	NULL	49.99	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	16	NULL	NULL	Rogers Apple iPhone 6s 32GB Smartphone - Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	17	NULL	NULL	Bell Apple iPhone 6s 32GB Smartphone - Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	18	NULL	NULL	Bell Apple iPhone 6s 32GB Smartphone - Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	19	NULL	NULL	Rogers Apple iPhone 6s 32GB Smartphone - Silver - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	20	NULL	NULL	Virgin Mobile Apple iPhone 6s 32GB Smartphone - Rose Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	21	NULL	NULL	Koodo Apple iPhone 6s 32GB Smartphone - Rose Gold - With A Tab Large	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	22	NULL	NULL	Fido Apple iPhone 6s 32GB Smartphone - Silver - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	23	NULL	NULL	Bell Apple iPhone 6s 128GB - Space Grey - 2 Year Agreement	NULL	49.99	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	24	NULL	NULL	TELUS Apple iPhone 6s 32GB Smartphone - Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	25	NULL	NULL	TELUS Apple iPhone 6s 32GB Smartphone - Silver - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	26	NULL	NULL	Apple iPhone 6s 32GB Smartphone - Space Grey - Open Box	NULL	649.99	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	27	NULL	NULL	Koodo Apple iPhone 6s 32GB Smartphone - Gold - With A Tab Large	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	28	NULL	NULL	Apple iPhone 6s 32GB Smartphone - Rose Gold - Open Box	NULL	649.99	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	29	NULL	NULL	Virgin Mobile Apple iPhone 6s 32GB Smartphone - Silver - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	
	30	NULL	NULL	Virgin Mobile Apple iPhone 6s 32GB Smartphone - Gold - 2 Year Agreement	NULL	0	NULL	BestBuy	NULL	2017-03-31 11:59:10	

Figure 7: product table records

## 7.0 Tomcat 8.0 Installation

- Goto <http://tomcat.apache.org> ⇒ Under "Tomcat 8.5.{xx} Released" (where {xx} is the latest upgrade number) ⇒ Downloads ⇒ Under "8.5.{xx}" ⇒ Binary Distributions ⇒ Core ⇒ "ZIP" package (e.g., "apache-tomcat-8.5.{xx}.zip", about 9 MB).
- Create your project directory, say "d:\myProject" or "c:\myProject". UNZIP the Downloaded file into your project directory. Tomcat will be unzipped into directory "d:\myProject\apache-tomcat-8.0.{xx}".
- For ease of use, we shall shorten and rename this directory to "d:\myProject\tomcat". Take note of Your Tomcat Installed Directory. Hereafter, I shall refer to the Tomcat installed directory as <TOMCAT\_HOME>.

## 8.0 Web Application

The entire application of WebOMiner\_S is available at “miner.newcs.uwindsor.ca.” To access it as user just type the URL “54.69.20.208:8080/about” in your web browser.

To access it as administrator in order to modify/ improve the existing work ask the network administrator to add you as “sudo user” to the server, so that you can access the contents from your login.

## **VITA AUCTORIS**

NAME: Amrutraj Chachad

PLACE OF BIRTH: Maharashtra, India

YEAR OF BIRTH: 1988

EDUCATION: Thakur Polytechnic Mumbai, MH, 2009

Rajiv Gandhi Institute of Technology Mumbai,  
MH, 2012

University of Windsor, M.Sc., Windsor, ON,  
2017