2010

Towards Comparative Web Content Mining using Object Oriented Model

Titas Mutsuddy
University of Windsor

Towards Comparative Web Content Mining using Object Oriented Model

By

Titas Mutsuddy

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

2010

Your file Votre référence
ISBN: 978-0-494-80235-9
Our file Notre référence
ISBN: 978-0-494-80235-9

Author's 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

Web content data are heterogeneous in nature; usually composed of different types of contents and data structure. Thus, extraction and mining of web content data is a challenging branch of data mining. Traditional web content extraction and mining techniques are classified into three categories: programming language based wrappers, wrapper (data extraction program) induction techniques, and automatic wrapper generation techniques. First category constructs data extraction system by providing some specialized pattern specification languages, second category is a supervised learning, which learns data extraction rules and third category is automatic extraction process. All these data extraction techniques rely on web document presentation structures, which need complicated matching and tree alignment algorithms, routine maintenance, hard to unify for vast variety of websites and fail to catch heterogeneous data together. To catch more diversity of web documents, a feasible implementation of an automatic data extraction technique based on object oriented data model technique, OOWeb, had been proposed in Annoni and Ezeife (2009).

This thesis implements, materializes and extends the structured automatic data extraction technique. We developed a system (called WebOMiner) for extraction and mining of structured web contents based on object-oriented data model. Thesis extends the extraction algorithms proposed by Annoni and Ezeife (2009) and develops an automata based automatic wrapper generation algorithm for extraction and mining of structured web content data. Our algorithm identifies data blocks from flat array data structure and generates Non-Deterministic Finite Automata (NFA) pattern for different types of content data for extraction. Objective of this thesis is to extract and mine heterogeneous web content and relieve the hard effort of matching, tree alignment and routine maintenance. Experimental results show that our system is highly effective and it performs the mining task with 100% precision and 96.22% recall value.

Keywords: Web content mining, Object-oriented mining, Automatic web data extraction, Wrapper induction, Web information integration.

Acknowledgement

I would like to give my sincere appreciation to all of the people who have helped me throughout my education. I express my heartfelt gratitude to my wife and daughters for their support throughout my graduate studies.

I am very grateful to my supervisor, Dr. Christie Ezeife for her continuous support throughout my graduate study. She always guided me and encouraged me throughout the process of this research work, taking time to read all my thesis updates.

I would also like to thank my external reader, Dr. Gokul Bhandary, my internal reader, Dr. Asish Mukhopadhyay, and my thesis committee chair, Dr. Xiaobu Yuan 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.

At last, I would express my appreciations to all my friends and colleagues, for their help and support. Especially, I would like to thank Chris Drouillard and Mohammad Harunorrashid for their help in implementation. Thank you all!

Table of Contents

Table of Figures

Table of Tables

1. Introduction

World Wide Web (WWW) is growing exponentially over the years. So, web documents became a largest repository of information (Kosala & Blockeel, 2000).Web content usually means to those information that a user see in a web document. It also includes some hidden information that helps user interaction with web contents. Web contents are heterogeneous in nature and may be in different forms like text, image, hyperlink, metadata, audio, video and others with their combinations. A complete classification of all these different types of web contents does not exist.

When any mechanism is used to extract relevant and important information from web document or to discover knowledge or pattern from web document, it is then called web content mining. Traditional mechanisms are: providing a language to extract certain pattern from web page, discovering frequent pattern, clustering for document classification, machine learning for wrapper (e.g., data extraction program) induction, and automatic wrapper generation. All these traditional mechanisms are unable to catch heterogeneous web contents together as they strictly rely on web document presentation structure. Annoni and Ezeife (2009) present a model for representing web contents as objects. They encapsulated web contents in object-oriented class hierarchy which enable to catch heterogeneous contents together in unified way without strictly relying on presentation structure.

This thesis studies the idea of modeling web contents in objects and develops a mining process for object-oriented data model for web content integration or comparative mining. The rest of this chapter is organized as section 1.1 introduces web mining and its categories; section 1.2 introduces the phases of web content mining and section 1.3

introduces the idea of object-oriented web content extraction, section 1.4 Thesis contributions and section 1.5 outline of the Thesis Proposal.

1.1. Web Mining

Organizations that have large amount of data need to make decisions that impact their future activities. Data mining is a process of extracting relevant and important knowledge from that large data to facilitate decision making. According to Etzioni (1996) web mining is a data mining technique to automatically discover and extract information from web documents and services. Web mining became important for knowledge discovery in business development, merchandise, personalization, and integration of web information. Borges et al. (1999) categorized web mining into three areas; web structure mining, web usage mining and web content mining. Kosala et al. (2000) defined web structures as inter-document structure of web pages which is represented by hyperlinks within the web itself. These hyperlinks are used in web pages for navigating to other web pages for interested information. From data view of web content mining, Kosala et al. (2000) defines web structure within the web documents (intra-document structure), the way how web content data are represented. Web usages are the history of user's visit to web pages generally stored in chronological order in web log file. Web contents are all the hard data such as text, images, audio, multimedia information in the web pages. Web contents are primary information of a web page. There are some other information or block in the web pages such as advertisement, attached pages, copyright notices. These are also web contents and usually are not considered as part of the primary page information. This unwanted information in a web

page is called the noise information, and usually need to be cleaned before mining the web content (Gupta et al., 2005; Li and Ezeife, 2006).

1.1.1. Web Usage Mining

Web usage information or the history of user's visit to different web pages are generally stored in chronological order in web log file, server log, error log and cookie log (Buchner and Mulvenna, 1998). General format of a web log string is as follows:

137 207 76 120–[30/Aug/2001 12 03 24-0500] "http //www cricinfo com/bangladesh/content/current/team /25 htm HTTP/1 0" 200 2781"

This web log string has certain set of information about user access to web as follows:

Field	Description	Example
Host/ip	Remote client IP address	137.207.76.120
User	Remote log user name	'–' for anonymous user or 'xyz' for particular user.
Date	Date, time and time zone of request	30/Aug/2001:12:03:24-0500
Request URL	User Request Identifier (URI) with the Uniform Resource Locator(URL) string	URI: http, ftp, mailto etc URL: *http://www.cricinfo com/bangladesh/content /current/team/25 html HTTP/1 0"*
Status	Status code returned to client	200 [series of success]
Bytes	Bytes transferred	2781 bytes

Table-01: Web log information

Web usage mining finds the relationships or patterns of the user's visit to different web pages from the access log files. The frequency of certain web page visit and the common traversal paths by the users are important information for discovering the browsing behavior of the web page users. If a web user visits most of the times to a certain type of page of web site, for example: *"http://...../...../products/games/hardware.html"*, which is a path for game hardware, this means that particular customer is interested to buy game

hardware products. Web usage mining helps to get this information and is used by marketing companies to sell their products to the targeted customers.

1.1.2. Web Structure Mining

A web structure defines the structure of a web site (Kosala et al., 2000). Web sites usually consist of a set of web pages. Each page of a web site represents a set of information. Hyperlinks (or links) are used in web pages to navigate from one web page to other web pages of the site for navigating information. Web structure mining is the process of discovering structure information from the web. This type of mining can be performed either at the (intra-page) document level or at the (inter-page) hyperlink level (Kosala et al., 2000). In case of intra-page web structure, Hyper-Text Mark-up Language (HTML) presentation tags represent the page structure and are commonly used for web content mining. By web structure mining, it usually means the inter-page structure mining of web site where structure of a web site is represented as a typical web graph as

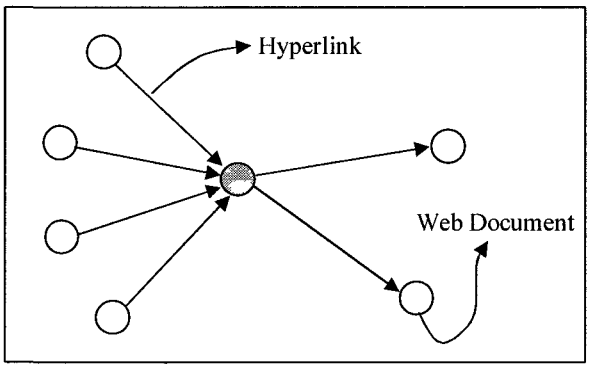


Figure-01: Web structure graph

shown in figure 01. A web graph consists of web pages as nodes, and hyperlinks as edges connecting between two related pages. Revealing web documents structure are effective for navigation purposes. For example, from the links, important web pages can be discovered for a particular keyword, which is a key technique for search engines. Other

important applications of web structure mining are: discovering the topology of hyperlinks and then categorizing the web pages, generating similarities and relationships between different web sites, page ranking, and link-based similarity search.

1.1.3. Web Content Mining

Web contents are the core data or information of a web page. These data can be in the form of text, image, audio, video, multimedia, hyperlinks or in combination of these formats in a web page. Web content data can also be un-structured (e.g., bulk text), semi-structured (e.g., HTML page content), structured (e.g., XML, table, database generated and multimedia data).

Un-structured web content data are represented by a full bag of words or texts or phrase-based feature representations. These features can be Boolean or frequency based and can be reduced using different feature selection techniques. Common text mining techniques like machine learning, statistical and Natural Language Processing (NLP) can be used for mining the unstructured web content data (Kosala et al., 2000).

Multimedia web data are the multimedia data embedded or triggered by web page through a mouse or keyboard event or automatically generated event while browsing through the web page. Multimedia data mining is comparatively a young sector of research area developing in recent years with increasing demand of surfing for music, music video, movie; online music group, large varieties of online communities for sharing personal information, hobbies and interests especially in teenage and young age groups. Common statistical machine learning theory or fuzzy logic theories are used for mining the multimedia web data (Petrushin et. al., 2007).

Semi-structured and structured data are most common types of data format for web documents. Typical HTML web page contents are semi-structured web data, which corresponds to a collection of facts and consists of text, image, hyperlink, structured records such as list, table, and database generated content. These types of featured data are rich and common representation of the web document structure.

Web Content mining is the process of extracting targeted facts from web documents. Web content corresponds to the collection of facts a web page is usually designed to convey to the users. Web content mining aims for a target fact to be extracted from web document or to discover patterns from web document. Common applications of web content mining identify the topic represented by a web document, categorize web document, find similar web pages across different servers, enhance standard query relevance with user or role, recommendation of top relevant documents in a collection, filter documents based on targeted facts.

1.2. Phases of Web Content Mining

Web content mining need several steps of pre-processing before mining it efficiently. First it needs to identify targeted facts in web documents and need to exclude noise contents (Li and Ezeife, 2006). Next step is the extraction of targeted facts from web documents. After extraction, it then needs to classify extracted contents according to their category. This data are then ready for mining to discover knowledge or underlying patterns.

1.2.1. Web page cleaning

A typical web page consists of a set of semantic blocks. Each block contains data contents. There is no unified way to represent data contents and blocks. In general, most web pages have three major blocks; header block, body block and footer block as shown

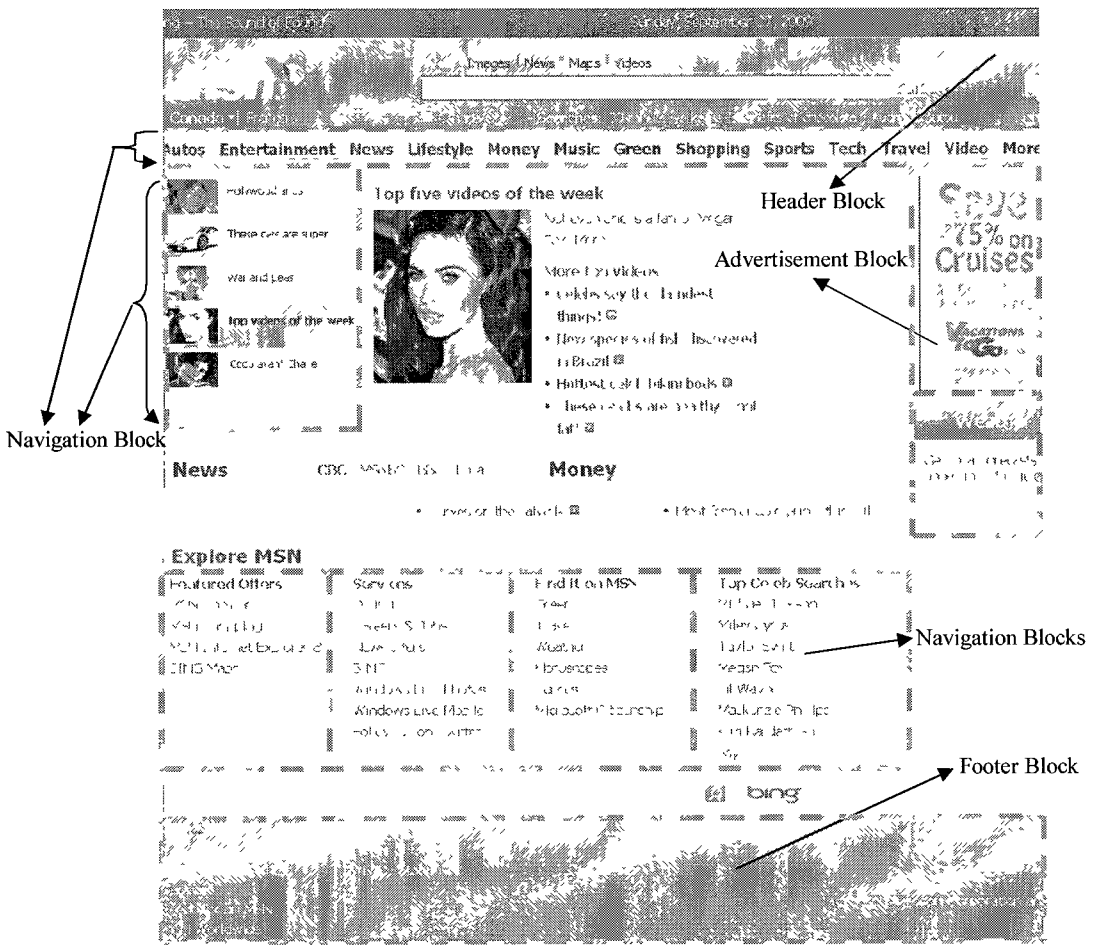


Figure-02: Blocks of a typical web page

in figure 02 (Annoni and Ezeife, 2009). The body block contains the major web document information. Other blocks like advertisement blocks, navigation block may be contained in all these three major blocks. A header block consists of page heading, company logo, advertisement and search option. Body block may consist of a set of

content blocks, navigation blocks, advertisement blocks. This is the core part of the web document including a set of noise blocks. Footer block consists of copyright notice, advertisement and navigation bar. These heterogeneous web pages need pre-processing like cleaning the noise contents for effective page classification/categorization and data warehousing (Chaudhuri et al., 2003; Gupta et al., 2005). It is easy for a person to understand the structure of a webpage by browsing them but difficult job for a machine to do automatically. There is no easy way to differentiate the noise blocks from content blocks (Li and Ezeife, 2005). For example, an advertisement in a page may become important if user is interested in it and it can contain important citation links that are valuable for PageRank (Page et al., 1998). But at the same time, an advertisement block may be considered as noise if user is not interested in it. So, it may deteriorate the page classification and mining quality. Many researchers just only extract data from the web pages (Chaudhuri et al., 2003) but other researchers emphasize on removing the noise contents from the web documents for improving quality of web content mining (Yi et al., 2003; Li and Ezeife, 2006).

1.2.2. Web data extraction

A typical web page contains a set of data objects or records such as a list of products, services and image or text explaining details of their products. These data may be hard coded or generated from databases and encoded by hypertext tags of the web page in leveled or hierarchical structure. There is no unified data model for web pages. So, extraction of web content data is highly dependent on the presentation structure of the web page. Wrapper (e.g., data extraction program) induction (Muslea et al., 1999) and Automatic Wrapper Generation (Liu et al., 2004) are two popular web data extraction

techniques widely used today. Former technique is a supervised learning process and the latter is unsupervised learning process. In supervised learning, it needs a set of sample web pages or training pages for user defined marks / labels to learn extraction rules from these training pages first. Then, the same rule is applied to other pages of the WWW for information discovery. Figure 03(a) shows simple personal information which can be represented in hierarchical structure as shown in figure 03(b) and in leveled structure

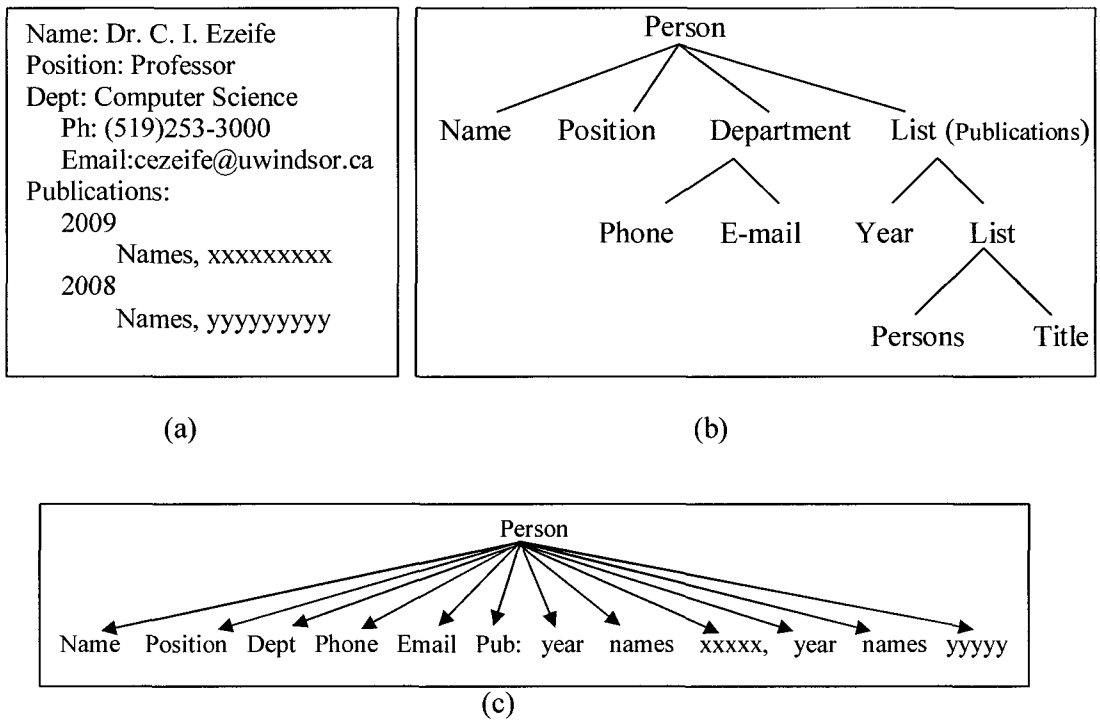


Figure- 03: Semi Structured web content data

as shown in figure 03(c). To extract the targeted information or items from nodes the wrapper needs a rule that will extract information from the parent node. In this process, user first marks some targeted items from a set of similar training web pages (called positive pages), then an algorithm or the rule is developed for these training pages to

extract the information from the nodes and if the defined rule works as expected, then this rule is used for other web pages in the WWW for information extraction. For example:

Problem: We are interested to extract the area code of phone numbers for all branches of PizzaPizza store of Canada.

Training Examples: Let's take four distinctive web pages as training pages from PizzaPizza store branch websites from different corner of the county. The address data block of these training pages are E1, E2, E3 and E4 as shown below:-

E1: 2203 Wyandotte St W, Windsor   ON , Phone 1- **519** -948-5133

E2: 158 Dundas Street, London   ON , Phone (**519**) 667-1111

E3: 7348 Kingsway Burnaby   BC , Phone 1- **604** -519-1111

E4: 5184 Avenue du Parc, Montreal   QC , Phone: (**514**) 737-1111

Extraction rules: To extract area codes of phone numbers from above mentioned training pages, user needs to develop an algorithm based on some extraction rules. One possible set of extraction rules are shown below:

Start Rule:	*End Rule*:
R1: SkipTo (( )	SkipTo ( ))
R2: SkipTo (-<**b**>)	SkipTo(</**b**>)

Here, R1 and R2 are rules, each of which have a *Start rule* and an *End rule*. The wrapper needs to identify the list of store location page from PizzaPizza web site for each province. It then needs to identify data blocks. The wrapper then can start iteration with its R1 *Start rule*, if it succeeds then it ends with R1 *End rule* or it iterates to R2. Here, when wrapper program identified the data block of E1 training example, it will start with R1 *Start rule* which fails and then it will try with R2 *Start rule* (e.g., hits -<**b**>) which succeeds. It will then start extracting all characters until it hits the R2 *End rule* (e.g., hits

). Similarly, E3 will be extracted by R2 and E2 and E4 will be extracted by R1. The algorithm ends when all the positive examples are covered.

Problem with this kind of supervised learning is the need for heavy manual labeling of training pages. This is labor intensive, time consuming and needs regular wrapper maintenance effort. So, automatic wrapper generation for data extraction is becoming more popular over the years. In this technique, a single page (positive page) or a set of pages are given with multiple data records and then it generates the extraction patterns from the WWW. Common technique is to identify the data regions and data records through string matching or document tree matching (Muslea et al., 1999; Zhao et al., 2005; Liu, 2007). The string matching technique (discussed in section 2.2.A) needs to identify the edit distance for matching and the document tree matching technique (discussed in section 2.2.C) is needed for matching web presentation tree and its alignment. For example, automatic wrapper generation from a set of positive pages by tree matching based of "Road Runner" algorithm (Crescenzi et al., 2001) is shown in figure 04. Here, authors use multiple sample pages: each contains one or more data records. At the beginning, a sample page is taken as the wrapper. This wrapper is then refined by solving the mismatches between the tokens of wrapper and each sample page. Using this set of sample pages, a wrapper as regular expression is generated. Figure 04 shows string mismatch and tag mismatch and the generated wrapper after solving mismatch between two sample pages.

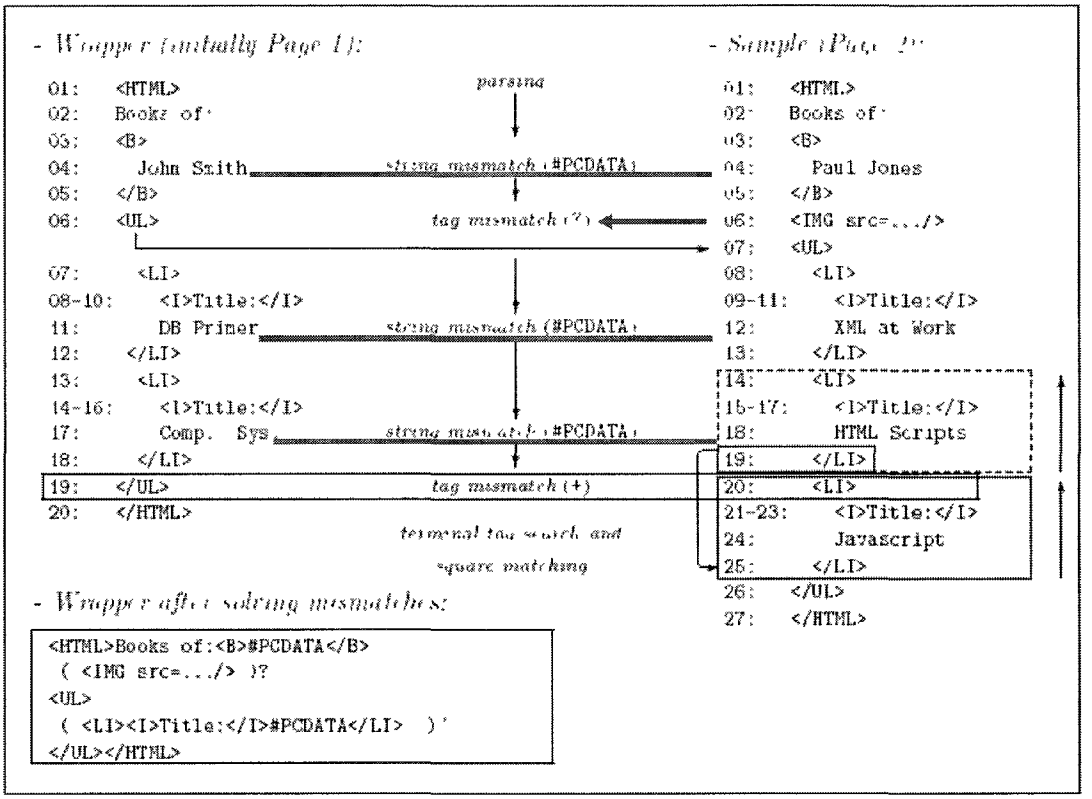


Figure-04: Tree matching for Wrapper Generation (Liu, ACL Tutorial 2007).

Recently, Annoni and Ezeife (2009) modeled web documents as object oriented web data model for automatic web content extraction. Section 1.3 discusses in details of this new approach.

1.2.3. Web Page classification and categorization

Web page classification or categorization is the process of automatically assigning web pages into a set of predefined categories. In general, certain types of web data are usually found in certain pages of a web site. For example, an online product sales website or Business to Customers (B2C) website contains a set of web pages and navigation sequence (Ai et al., 2006) as shown in figure 05. This means, for example, that visiting a

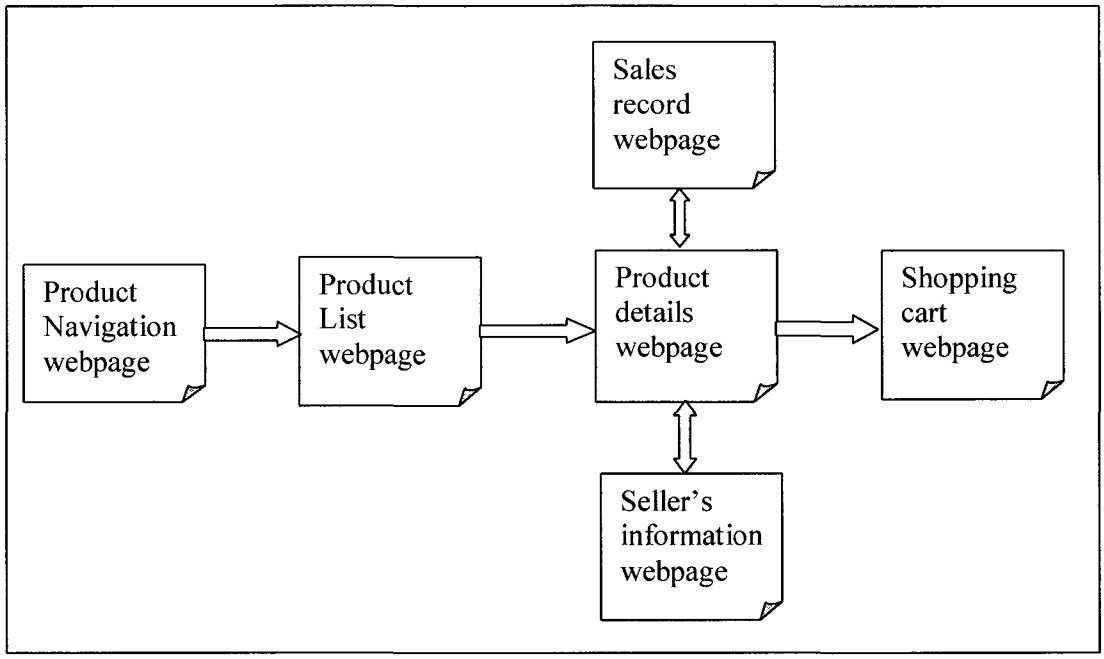


Figure-05: Common B2C web site structure (Ai et al., 2006)

B2C retail store website like "Future Shop" (www.futureshop.ca), one first finds an index page or product navigation page. Clicking on a link on this page brings up product-list page, list of all products or certain category of products including product image, product name, product Id, short description and price of each product are listed in product-list page. Clicking to a specific product in the list will bring up product-details page, which gives detailed information of that product and so on until we hit the "Shopping Cart" page. User navigation is generally restricted beyond this page after putting personal and payment information for buying products. Figure 06 shows the monitor page of CompUSA.com, which is an example of product-list page. We will use this page of figure 06 and example 1.1 (page 19) as running example in subsequent sections of this thesis. We have selected product-list page because in the entire structure of a retail store website (shown in figure 05), product list page is the most data-rich page for comparative mining.

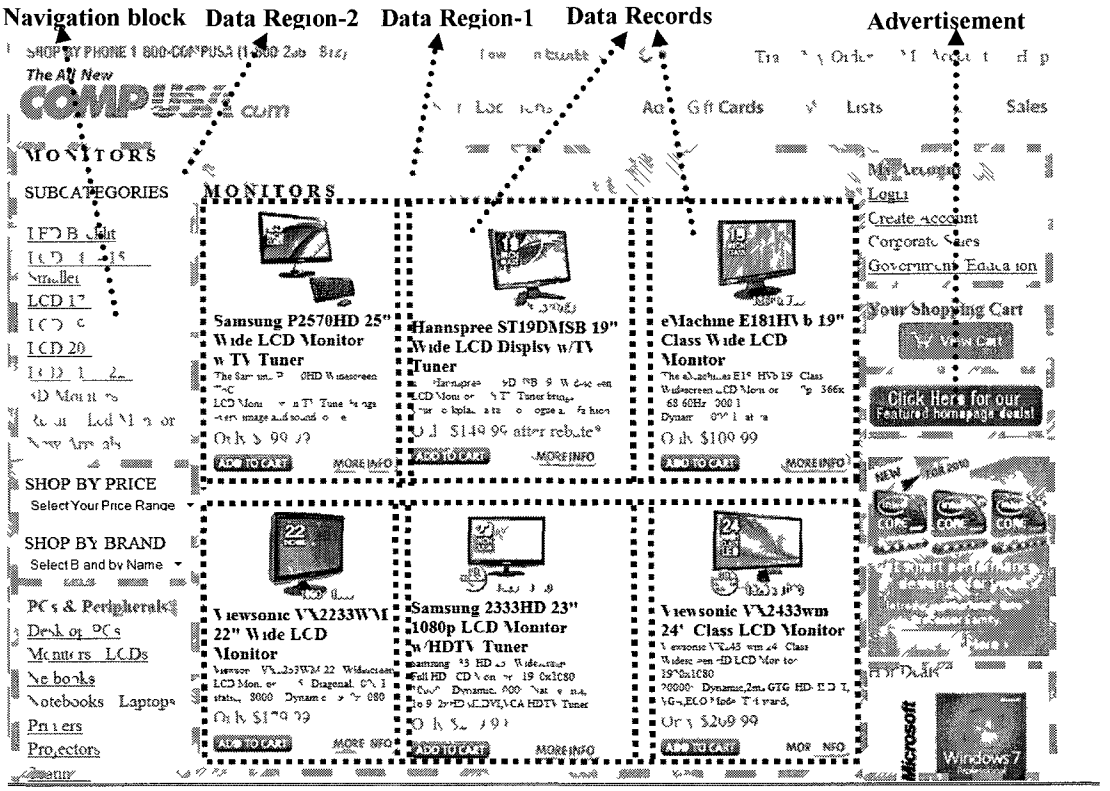


Figure-06: Data regions and data blocks

It gives brief information about different products such as product image, brand, model or product number, short description, price and navigation to more information such as details specification and user feedback. A set of all these information for a specific product is called "data record". Similar category data records are in general organized in "data region" of a product-list page. Figure 06 shows data region-1 consists of six similar category data records. Other interesting information in a product-list page is set of navigation blocks and advertisements. A navigation block consists of a set of pair of a hyperlink or URL and a text Users read the text and if interested, click on that hyperlink to bring up that page Advertisements are generally pair of hyperlink and image. This

image is about the product or company and the related hyperlink brings user to the related page. Advertisements in general bring the user to a new website.

1.2.4. Web Data Warehousing

Data warehouse refers to a database that is maintained separately from the operational database. It allows heterogeneous database integration with a variety of application systems. Web content data are scattered in different web sources and need to integrate in a warehouse environment for analysis and discovery process. In relational database management system (RDBMS), data warehousing is comparatively easy because data are historical and nonvolatile. But web content data are updated frequently, it is volatile and not historical (Bhowmick et al., 1999; Dung et al., 2007). The maintenance of a data warehouse based on the web content data is not easy in comparison with company based conventional data warehouse. One potential problem is the "multiform" of web data, which needs to convert into unified format to store in physical database. For example, an image data needs to be extracted and stored in different physical location with unique id and its reference needs to be stored in the database. A mapping is required between the database and the physical location to get the image. Some researchers adopted the web data extraction system in virtual approach without creating physical data base and warehouse (Bornhövd, and Buchmann, 1999; Chawathe et al., 1994; Liu, 2007). In this system, the related data are integrated from the WWW in real time on the fly as query response. This technique is called Web Query Interface Integration.

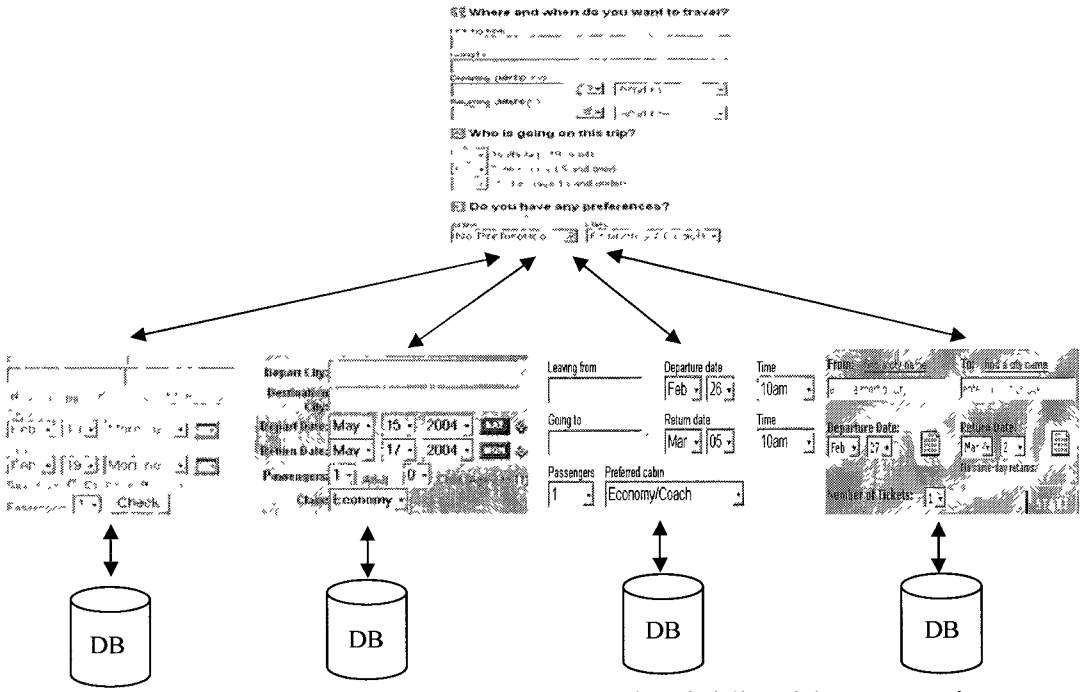


Figure- 07: Query interface from same domain of airline ticket reservation.

Web query interface integration provides a global query interface to the user so that user does not need to manually query individual web source for required information. For example, in case of airline ticket reservation of figure 07, two query attributes of interfaces do not have matching attributes but their domain is same. To create a global query interface for these interfaces, an efficient mapping is required for each attribute. In this case, web content data from different web sources are not replicated and guaranteed the consistency but there is no tight control over the quality of data that are usually obtained from the data warehouse and sophisticated query optimization is not possible.

Other researchers adopt the data warehouse based approach (Zhu, 1999; Darmont et al., 2002; Zhu et al., 2001; Dung et al., 2007). Most of the researchers in this area rely on the already established Online Analytical Processing (OLAP). OLAP is a category of applications and technologies that allow the collection, storage, manipulation and reproduction of multidimensional data with the goal of analysis. The classical data

warehouse approach is not very adequate to deal with multiform data (e.g., texts, images, sounds, videos etc). OLAP is also recognized as inefficient and ill-adapted (Darmond et al., 2002). Zhu, et al. (2001) modeled a semi-automated approach to use the relational model based star schema for transformation task for storing web data into existing data warehouse. Darmont, et al. (2002) modeled a logical XML schema with DTD to transform the multiform web data into XML document and then mapped to relational database which contents were then remodeled in multidimensional way to store in star schema based warehouse.

1.2.5. Mining Web Content

Web content mining is the overall process of discovering potentially useful and previously unknown information or knowledge from the web content. It has four major tasks; Resource finding, Information selection/pre-processing, Generalization and Analysis. Web content mining research can be broadly classified into two streams. First stream is the Information Retrieval (IR) and the second stream is the Information Extraction (IE) from web page. There is a common misunderstanding about Information Retrieval and Information Extraction (Kosala et al., 2000).

Information Retrieval (IR) process tries to retrieve all the relevant documents and at the same time retrieve as few of the non relevant documents. Modern search engines, web document classification are examples of IR process. For example, "google" is an example of IR process. For any "text" keyword, its engine traverses in WWW, classifies relevant web pages of given keyword and indexes them and returns result as per their rank. Information extraction (IE) is the process of extracting the relevant or targeted information from the given documents. Information extraction from general WWW

without any targeted fact is meaningless. This process focuses on specific web sites or contents for extraction and often used in Web Content Data Integration (Liu, 2007) by building a virtual database or finding the schema of web documents or building web knowledge base (Kosala et al., 2000). IE process is also used as a part of web content mining for building the web data warehouse. Annoni and Ezeife (2009) proposed representing web documents as Document Object Model (DOM) Tree based object-oriented model for web content extraction. Section 1.3 discussed about this model. Our research area is in IE process and given below a motivational example for your work.

Motivational Example:

Develop a shopping planner that is able to answer the following type queries:

1. *Given a product type, output all related products on sale around Windsor area right now and advice the user about buying the product based on buyer feedback.*
2. *Given a flight trip plan from Windsor to Delhi, India with 2 night halt in London and want to touch Frankfort airport on the way to Delhi, output all flight in ascending price and advice the user about the best time and airline to plan the trip.*
3. *List all journal and conference publications on "Sequential Pattern Mining" in 2010 by title, authors, journal/conference, place, page and date.*
4. *List all 17" LCD Samsung monitor selling around Windsor with price range less then $200 and show the user graphical variation about its price in last 1 year and where and when is the best time to buy.*
5. *List all songs by albums of singer Lionel Richie with option to play any music with user event and show user feedback or ranking of the music.*

To solve these problem queries using free web data we need to extract relevant information from web and store into any central repository, we then need to either create a data warehouse or analyze query for mining from the repository to result the user. One thing we need to make clear at this point to distinguish our problem with similar approach offer by "Web Query Interface" and "Web Service". "Web Query Interface" is discussed in section 1.2.4 and it is clear that it can not answer our query. The "Web

Service" is a new approach used in "Semantic Web" which generally deals with Business to Business (B2B) data integration. Web Service is a proprietary service and needs to buy the service from the service provider. Semantic web uses XML based Web Service Description Language (WSDL) for enterprise data integration. It does not deal with free HTML web information. Different provider companies' offer their service or data to use by other business or parties through semantic web and can use their service in buyer Company's business webpage. For example, "google tool bar" is a common web service offered by "Google" to use by other business / corporate company web sites for search option. Similarly, "Chapters", "Burns & Nobles", "Amazon" and "e-bay" web service can be used to integrate book domain information integration.

Both "Web Query Interface Integration" and "Web Service" gives instantaneous information and do not hold any historical information. Our system deals with free HTML web information and need historical events to answer these queries. In this thesis, we are working for data extraction problem from the web toward answering the motivational example queries above using object-oriented web data model and we state our thesis problem as following example 1.1:

Example 1.1: Given a product list web page of a retail store shown in figure 06, extract all types of information like:

- *(i) Those related to data records such as product image, product brand, product id, short description, product price.*
- *(ii) Navigation information such as link URL, link id or name.*
- *(iii) Advertisements such as product advertised, image, URL links to related website.*

The extracted information will be stored in the database for comparative mining and querying.

1.3. Object Oriented Web Content Extraction

Annoni and Ezeife (2009) propose an object-oriented web data model for extraction and mining of full diversified web data including contents and page presentation structure. They modeled web data as web content objects and web presentation objects to address a unified way of mining unstructured, loosely or strictly structured data. They divided the web documents into three zones; header, body and foot zone, based on value in content mining and their physical location in web browser. They assumed that a web document should compose of at least one zone object (i.e. body zone) and up to three zones (i.e., header zone, body zone and the foot zone). In web browser, a header zone located at top of the web page, foot zone at bottom of the page and the body zone is the main body of the web page between header and foot zone. The header zone usually consists of page identification information, company logo image, company name, advertisements. This zone is useful for extracting information about the page content, metadata but does not have any importance for data contents and its mining. The body zone is the basket that contains the page contents and most important for content mining. At the same time, it is also crucial to clean effectively for extracting real valued contents. The footer zone usually contains the copyright information, advertisement, and links that have no value in terms of content mining. A web page usually contains a set of hypertext presentation tags including the customs tags defined by page designer. Annoni and Ezeife (2009) propose not to evaluate all the tags of a web document. They tried to ignore less valuable presentation tags of the web page because this is time consuming and is not always meaningful. The question is how these zones and their boundaries are to be identified. A well formed HTML page tag format is

<html><head></head><body></body></html>. This hierarchical format indicates that all the tags of a page should be within the root tag <html> and then it has two child tags <head> and <body>. All the information within the <head> tag is the header information and the content data are within the <body> tag. But the real time web page presentation format is not so simple. Lots of different pre-formatting information, styling information, embedded client side and server side scripting programs in different languages, flash programs, meta data are the puzzling problem for efficient page content data extraction. It is easy to identify that the meta-data, scripting functions have become part of the header zone information but there is not necessarily a clear boundary between the header zone and the body zone.

Annoni and Ezeife (2009) suggest two tag series (a set of at least five or more <a> or <area> sibling tags) to distinguish the boundary between these zones. An <a> or <area> tag in an HTML file represents navigation URL. They observed that, a set of first five or more sibling <a> tags indicate the starting of body zone and called it series-1. The last set of five or more sibling <a> tags indicate the end of body zone of a typical web page and they called it series-2. They used two hypotheses for searching these series as follows:

1. If the search process of series-1 goes over half of the DOM tree size, the web document does not have header zone and series-1 is empty. The body zone's first tag is the first region node child of the closest region node in the sub-tree of "body" root.
2. If the search process of series-2, from half size of the DOM tree to its end returns null, the web document does not have foot zone and body zone's last

tag is the last region node child of the closest region node in the sub tree of "body" root.

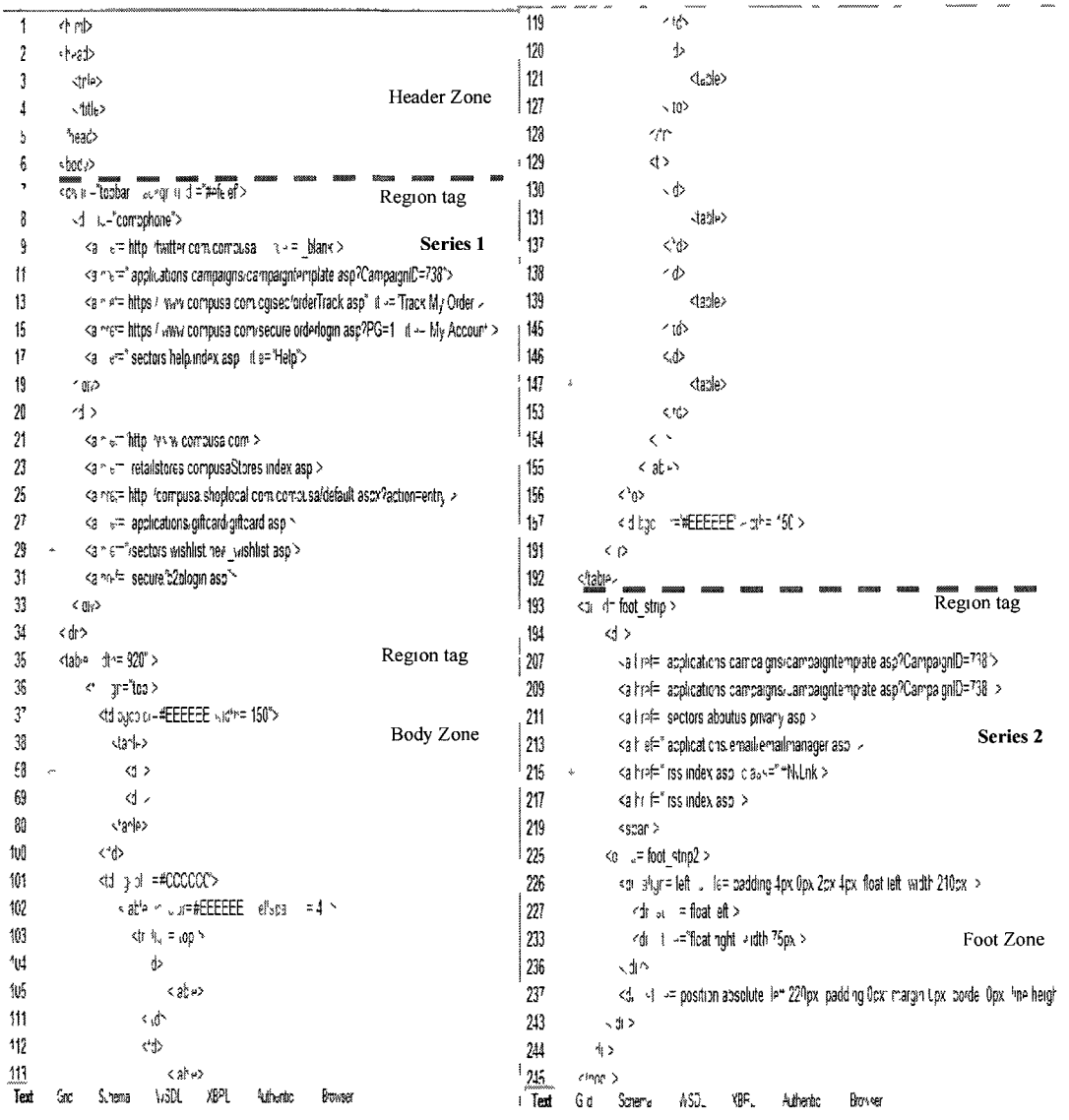


Figure 08: DOM tag tree of CompUSA.com web document for figure 06.

In case of our running example, figure 06 at page 14 shows the page what a user sees in a web browser. The internal page tag structure of figure 06 is given in figure 08 above (generated by XML viewer interface, a product of Altova.com). This hierarchical tag structure represents a tree for the entire webpage and illustrates how the contents of

the webpage are shown to the user. Web contents that are seen in figure 06 are embedded between these tags in this tag tree. Figure 08 shows line number of each tag and the symbol "⊕" represents embedded hidden tag in between the given tag at that line. The symbol "⊖" represents open embedded tag. We kept some child tag structure hidden to keep the figure 08 readable to reader and to identify easily the sibling tag structure. Any gap in line number indicates embedded child tags. For example, in figure 08 line numbers jumped from 58 to 69, that means there are 10 lines of hidden embedded tags at line 58 tag and indicated by the symbol "⊕". Here, line 7 starts with a region tag that consists of two data navigation block " " starts at line 8 and 20. Line 9 to 17 indicates the first five sibling <a> tags as per Annoni and Ezeife's (2009) definition and identifies as series-1. So, line 7 region tag clearly distinguishes between header and body zone. Similarly, line 207 to 217 indicates the last six sibling <a> tag series which is identified as series-2. So, the region tag at line 193 clearly distinguishes between the body zone and foot zone.

1.4. Thesis Contributions

This thesis includes lots of pre-processing work to prepare data for mining that are not addressed by Annoni and Ezeife (2009). We developed the architecture (we call it WebOMiner) for extraction and mining of web contents using object-oriented model. Our architecture has 4-modules: crawler module, cleaner module, extractor module and miner module. We developed algorithms for crawler module, modified freeware software "tagsoup" (http://home.ccil.hangorg/~cowan/XML/tagsoup) for cleaner module, modified and enhanced algorithms for extractor module initially developed by Annoni and Ezeife

(2009) and developed algorithm for miner module. We introduced an approach of generating and using automata for mining web content objects. The following are main contributions of the thesis in extractor and miner module of WebOMiner system:

1. We define data block and data region to ensure consistency between related data that is not addressed by Annoni and Ezeife's (2009). We therefore modified their ProcessContentSibling() algorithm to identify data block and data region.
2. In web page, information about content usually reside as tag attribute. We address to relate HTML tag attribute information with related contents to ensure identification of content, to assign object and other information together.
3. We define object class hierarchies according to our problem domain and defined schema matching to unify similar contents from different web sites.
4. We identify noisy contents in data block and prevent them entering into database table.
5. We implement and materialize object-oriented data model for web content and extract heterogeneous related web contents together.
6. We define a mining algorithm that identifies data block, generates a Non-Deterministic Finite Automata (NFA) based wrapper for extraction of related contents. Then classifies all data blocks of a web page according to their type and checks minimum support to ensure data consistency before entering them into database.

This thesis proposes a two level mining process. The first level mining (as discussed above) extracts and classifies content data from noisy flat object array defined by Annoni and Ezeife (2009), which is ready to enter into database table. This thesis then

recommends data warehouse and second level mining for deep knowledge discovery. Second level mining is similar to traditional data mining process, so excluded from the scope of this thesis work. We also study the advantages/benefits of this new approach over the existing approaches and conducted performance analysis. The second level mining is beyond the scope of this thesis work.

1.5. Outline of the Thesis Proposal

The remainder of the thesis is organized as follows: Chapter 2 reviews related work to this thesis. Chapter 3 details discussion of the problem addressed along with the new algorithms proposed. Chapter 4 gives performance analysis and experimental results. Chapter 5 draws the conclusion of this research and discusses future work.

2. Previous/Related Work

Our research area of web content mining is in Information Extraction (IE), which is focused on structured data extraction techniques and classified into three categories: wrapper programming language, supervised learning like wrapper induction, and automatic wrapper generation. Related works in this chapter are organized as wrapper programming language in section 2.1, wrapper induction in section 2.2, and automatic wrapper generation in section 2.3.

2.1 Wrapper Programming Language

Wrapper programming language provides some specialized pattern specification languages to help user construct extraction programs. Most of them provide visual interface to hide their complexities under simple graphical wizards and interactive processes. There is a wide body of research on Wrapper Programming Language based extraction and some examples are Lixto (Baumgartner et al., 2001), DEByE (Laender et al., 2002), Wargo(Rapaso et al., 2002), WebOQL (Arocena et al., 1998) and WICCAP(Zhao and Ng, 2004). All these extraction systems use Graphical User Interface (GUI) to interact with user input and hide internal complexity from the user. Lixto system internally uses logic-based declarative language "Elog" for extraction of content element from targeted web page given by user in GUI. Resultant outputs are given in XML format. Wargo system internally relies on two wrapper programming languages: Navigation SEQuence Language (NSEQL) for specifying navigation sequence and Data EXTraction Language (DEXTL) for specifying extraction pattern. We discussed the DEByE, and WICCAP System below in section 2.1.1, and 2.1.2.

2.1.1 DEByE: Data Extraction By Example

DEByE is an interactive tool that receives as input a set of example objects taken from a sample Web page and generates extraction patterns that allow extracting new objects from other similar pages (e.g., pages from the same Web Site). DEByE features a GUI that allows the user to assemble nested tables (with possible variation in structure) using pieces of data taken from the sample page. The tables assembled are examples of objects to be identified on the target pages. From these examples, DEByE generates object extraction patterns (OEP) that indicates the structure and the textual surrounding of the objects to be extracted. These OEP are then fed to a bottom-up extraction algorithm that takes a target page as input, identifies atomic values in this page, and assembles complex objects using the structure of the OEP as a guide.

2.1.2 WICCAP : From semi-structured to structured data

Web Data Extraction System (WICCAP) uses Web Data Extraction Language (WDEL), which is one kind of scripting language to provide features transforming web data. Basic unit of this language is *symbol*, which represents the nodes of trees. A *symbol* can have a set of *sub-symbols*. A *Term* is used to represent a tree. A *Term* in tree language can be mapped into tree graph. For example a Book Domain tree language is

Business_Book(Book1(Price, Author(Most_Popular_Item))……)

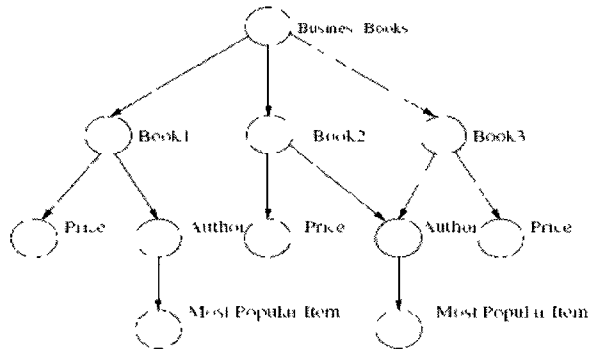
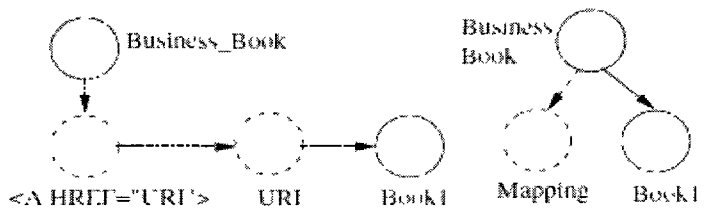


Figure-09: Logical view of WDEL language

given above can be represented logically as tree graph shown in figure 09. Note that, *Business_Book(Book1)*, *Business_Book(Book2)* and *Business_Book(Book3)* are three different *Terms*, because they represent three distinctive tree under *Book1*, *Book2* and *Book3* in tree graph. The authors defines the tree grammar as 4-tuple, $G = \{S, N, F, R\}$. Where, S = Set of root symbols, N = Set of Non-terminal symbols, F = Set of symbols and R = Set of generation rules. Where, $S \in N$ and $N \in F$, R is in the form $A \rightarrow \beta$, here A is a non-terminal node and β is a term over F. For a given *Term* in figure x: *Business_Book(Book1($17.49,Greg))*, the replacement of book title *Book1* by *#book*, price *$17.49* by *#price*, and author name *Grag* by *#name* will enable to symbolize the *Term* as : *Business_Book(#book(#price,#name))*. Therefore, the grammar for extraction of *Term* can be represented as follows:

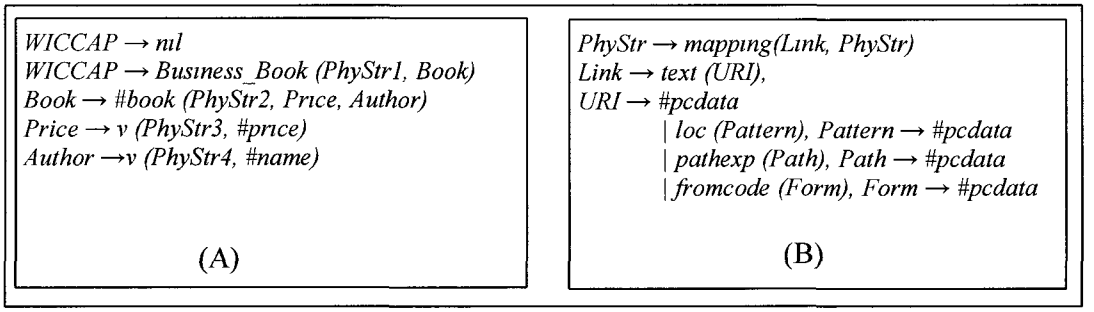
F = {Business_Book, nil, #book, #price, #name, WICCAP, Book, Price, Author}
S = {WICCAP} *// WICCAP = root symbol of the grammar*
N = {WICCAP, Book, Price, Author}
R = *WICCAP → nil* *//Generation rule starts with root symbol*
WICCAP → Business_Book(Book)
Book → #book (Price, Author)
Price → #price
Author → #name

The absolute paths for physical structure of web contents in WWW are in general complicated. So authors use mapping of physical structure with their logical view in tree graphs as shown below:



(a) Full Path (b) Collapse

Therefore rules need to redefine as (A) below and this mapping raises the need for another grammar to describe the physical paths shown in (B) below.



These grammars describe the schema of original web documents to be extracted, schema of extracted data and relationship between them. WDEL script is an instance of *Terms* that can be generated corresponding to WDEL grammar. WICCAP generates the output in portable XML data format that can easily be stored in relational database or can be viewed as required.

Main problem for wrapper language based extraction system is their reliability on diversified wrapper languages. None of these languages are standard, used by vast users and therefore fail to become popular in vast user community.

2.2 Wrapper Induction

Wrapper induction is either supervised or semi-supervised learning process for extraction and mining of web contents. It needs a set of sample web pages (called training pages) for user to define marks / labels to learn extraction rules from these training pages first. Then the same rule is applied to other pages of the WWW for information discovery. We now like to discuss about three popular algorithms that most of the Information Extraction (IE) systems use. These are String Edit Distance, Center Star Method, and Simple Tree matching algorithm.

(A) String Edit Distance:

String edit distance is a popular and widely used string matching/ comparison technique to find the same type encoded instances. The edit distance of two strings, s_1 and s_2 is defined as the minimum number of point mutation required to change s_1 into s_2. The point mutation may be to change a character, insert a character and delete a character. If c_1 and c_2 are two last characters of s_1 and s_2 respectively, then edit distance between s_1 and s_2 can be denoted as $d(s_1, s_2) = d(s_{1-}+c_1, s_{2-}+c_2)$. If $p(c_1, c_2)$ denotes the penalty for changing, inserting or deleting a character, then

$$p(c_1, c_2) = \begin{cases} 0, \text{ if } c_1 = c_2 \\ 1, \text{otherwise} \end{cases}$$

For example, let $s_1 = X\,G\,Y\,X\,Y\,X\,Y\,X$ and $s_2 = X\,Y\,X\,Y\,X\,Y\,T\,X$ are two strings. The edit distance is computed as follows:

$$ED(s_1, s_2) = \frac{d(s_1, s_2)}{(|s_1| + |s_2|)/2}$$

The edit distance matrix is given in figure 10. The final edit distance value is 2, which is the value in the bottom right corner cell. Figure 10 also shows the trace back path. Notice that a diagonal line means match or change, a vertical line means insertion, and a horizontal line means deletion. Thus, the final alignment of our two strings is:

s_1:	X	G	Y	X	Y	X	Y	-	X
s_2:	X	-	Y	X	Y	X	Y	T	X

The time complexity of the algorithm is $O(|s_1||s_2|)$ to fill the matrix.

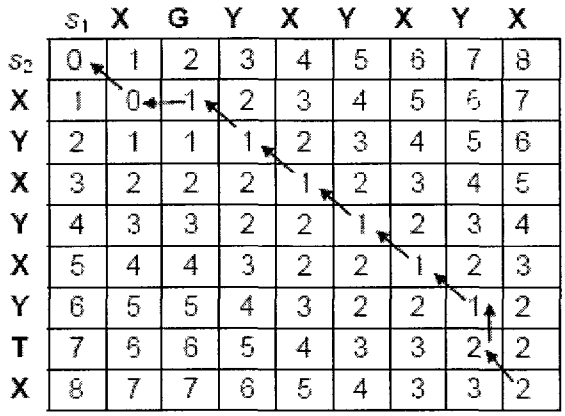


Figure-10: Edit distance matrix and back trace path (Liu, 2007).

(B) Center Star Method:

It is a classical technique for multiple string alignment. Chang et al., (2001) introduced it for data extraction based on alignments of HTML strings. In this method a set of strings are assumed as aligned as 'S'. A string s_c is selected as center string that minimizes:

$$\sum_{s_i \in S} dist(s_c, s_i) \quad \text{.....Equation (3)}$$

Here, $d(s_c, s_i)$ is the distance between two string s_c *and* s_i. The algorithm then iteratively computes the alignment of the rest of the strings with s_c. Figure 11 illustrates an example of string alignment using center star method and latter we will illustrate how it is using for web content extraction.

Let us have three strings, S = {ABC, XBC, XAB}. According to Center Star method, first string selects as center string. So, ABC is selected as the center string s_c and initializes multiple sequence alignment M by string ABC. Now we need to align other strings iteratively with respect to ABC.

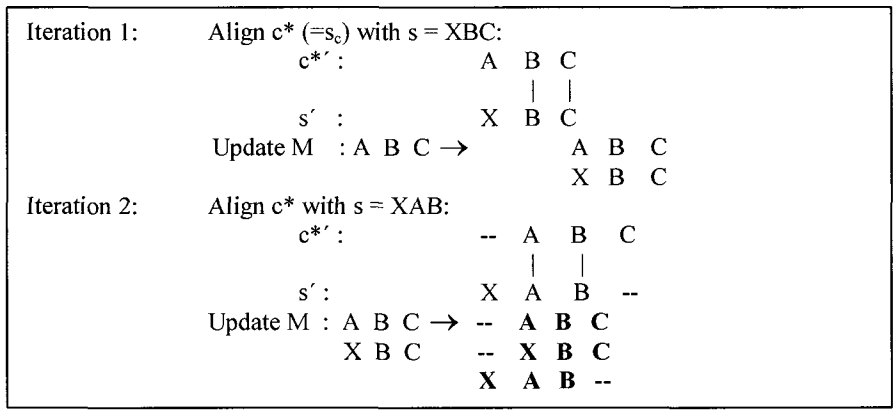
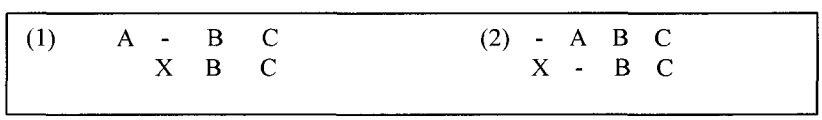


Figure-11: Example of Center Star Method

In figure 11, next string "XBC" is taken from the set S and aligned in iteration 1 by matching character B and C. The algorithm then updates a multiple sequence alignment M by including XBC aligning B and C with center star. Next, empty space is created to each string in M to accommodate for next alignment. In iteration 2, it aligned XAB matching A and B with center star and update M accordingly. There is a question of aligning ABC and XBC in iteration 1 since we don't know which of the following alignments is better:



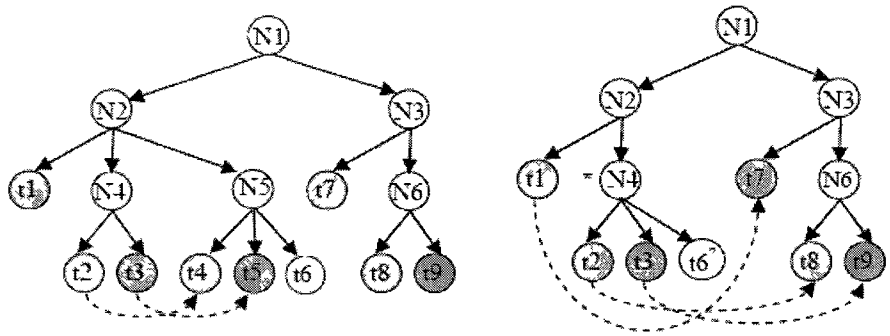
The authors did not resolve the problem in the paper. If there are k strings in S and all strings have length n, then time complexity of pair-wise alignment is $O(kn^2)$ and overall time complexity is $O(k^2n^2)$. So, this algorithm runs slowly for pages containing many data records and /or data records containing many tags.

(C) Simple Tree Matching:

Tree matching technique is similar to the string matching technique. It is introduced by Yang et al (1991). Zhai et al. (2005) used this Simple Tree Matching

(STM) algorithm to extract web data in flat and nested records. This algorithm associate minimum set of operations needed to transform one tree to another. In this classic formulation, the set of operations used to transform one tree into another includes, node removal, node insertion and node replacement. Algorithm takes two inputs: root of the tree and a threshold value, and outputs extracted data in relational table.

For a given tree root, if tree depth ≥ 3, the algorithm recursively traverses down the tree and performs matching operation between two child sub-trees of a node. It then aligns and links matched data items. For figure 12 below algorithm compares root of two



(X) (Y)

Figure-12: Tree Matching and aligning in (X), Aligned data nodes under N1 in (Y)

sub-trees A and B of input tree first (for example, sub-tree under node N4 and N5 of figure 12(X)). If roots contain identical symbols, then the algorithm recursively finds the maximum matching between first-level sub-trees of *A* and *B* and saves it in a *W* matrix. Maximum matching between A and B is denoted by *W (A, B)* and defined as below:

$$W(A,B) = \begin{cases} 0 & \text{if } R_A \neq R_B \\ m(\langle A_1, \ldots A_k \rangle, \langle B_1, \ldots, B_n \rangle) - 1 & \text{otherwise} \end{cases}$$

$m(\langle\rangle, \langle\rangle) = 0$ ⟨⟩ represents an empty sub-tree list

$m(s, \langle\rangle) = m(\langle\rangle, s) = 0$ s matches any non-empty sub-tree list

$$m(\langle A_1 \quad A_k \rangle, \langle B_1 \quad B_n \rangle) = \max \begin{cases} m(\langle A_1 \quad A_{k-1} \rangle, \langle B_1 \quad B_{n-1} \rangle) - W(A_k, B_n) \\ m(\langle A_1 \quad A_k \rangle, \langle B_1 \quad B_{n-1} \rangle) \\ m(\langle A_1 \quad A_{k-1} \rangle, \langle B_1 \quad B_n \rangle) \end{cases}$$

W(A,B) is a matrix populated based on weight parameter. If root of *A* and *B* do not contain identical symbol, the weight is zero. Otherwise, its value will be based on the number of pair node matching of their sub-tree. The definition of *m(.. , ..)* is similar to string edit distance but authors compute the maximum matching rather than distance.

To find the maximum matching between sub-trees N2 and N3 of figure 15(X), their root compared first. Since N2 and N3 contain identical symbols, $M_{2,3}[3, 2] +1$ (as per $R_A = R_B$ condition in above *W(A,B)* equation, here 2, 3 denote Node N2 and N3) is returned as maximum matching value between trees N2 and N3. $M_{2,3}$ matrix is computed based on $W_{2,3}$ matrix. Every entry in $W_{2,3}[i, j]$ is the maximum matching between i^{th} and j^{th} first level sub-trees of A and B. At level N4-N5, t2-t4 and t3-t5 are matched, so they are aligned and linked. At matched level N2-N3, it will only align N4 sub-tree and N6 sub-tree as shown in figure 12(Y), N5 will be omitted since it has same structure as N4. In this case, t2-t8 and t3-t9 are linked. T1 and t7 also linked as they matched. N4 is marked with '*' in figure-12(Y) as it is turned into prototype data record of match algorithm.

(D) DOM Tree building:

Building DOM tree from input pages is a necessary step for many data extraction algorithms. There are two existing approaches for building DOM Tree; Using Tags Alone and Using Tags along with Visual Cues. First approach only uses the tag pairs (e.g., start tag < > and end tag </ >) for building the DOM Tree but HTML mark-up language also allows non-pair tags (e.g.,
) as well as freedom from the necessity to close some inline tags (e.g., <li>, <hr>, <p>, etc). HTML is a flexible mark-up language and page

designer's error in using tag is mostly accepted. So, DOM Tree building by using the Tag Alone requires HTML code cleaning before building the tree.

Second approach is using the Tags and Visual Cues, which use the visual information (i.e., the physical location of the information on the computer screen by using web browser) along with the rendering tags. This approach is more robust because of its error tolerance. In this approach, four boundaries of the targeted rectangle of web page are located first by calling any rendering engine of a browser. It then follows the sequence of open tags and checks for containment to build tree. Containment check means checking if one rectangle is contained in another. In figure 13, there are three

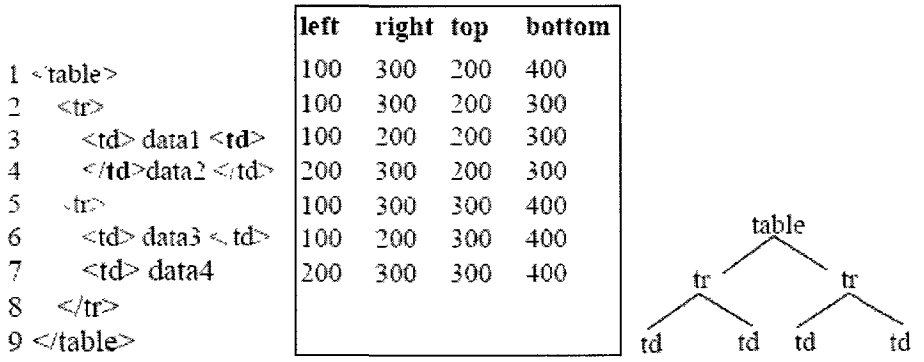


Figure-13: Boundary co-ordinates and resulting tree (Liu, 2007).

errors in HTML tag structure at lines 3,4 and 7 but this HTML segment can be rendered correctly in a browser. Boundary condition is shown in the figure which can be used to build the tree shown in the right side of figure 13.

2.2.1 STALKER: Hierarchical Approach to Wrapper Induction

Muslea et al., (1999) introduces this supervised learning approach, where a set of extraction rules is learned from a collection of manually labeled pages or data records. The rules are then employed to extract target data items from other similar formatted

pages. Data are embedded into the webpage in presentation tag tree. To extract data from any node of interest, the wrapper uses the presentation tree of the webpage and defines a set of extraction rules including *Start Rule* and the *End Rule*. The *Start Rule* indicates the starting point of the data extraction and the *End Rule* is the rule where to finish the extraction. Muslea et al., (1999) developed "Stalker", which is the main algorithm for wrapper induction. An example of this algorithm is given below:

Consider a data block representing the address and phone number of chain restaurants. The presentation tree and a training example data are shown in figure 14

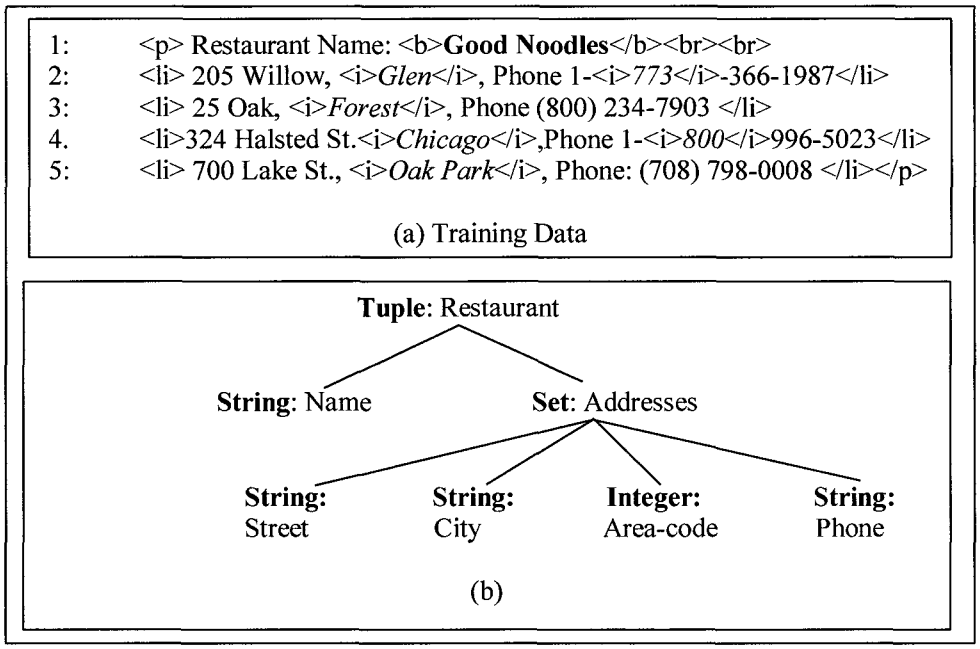


Figure-14: (a) Training data blocks, (b) Logical presentation.

Figure 14(a) is an example training page of Good Noodles restaurant having four distinctive branch addresses. This page shows its name in line 1 and then followed by four data blocks shown in line 2,3,4,5 showing addresses of its branches. We want to

extract area code of phone numbers from all branch addresses of this restaurant. The wrapper needs to go through the following steps for extraction:

2. Identify entire list of addresses. We can use the start rule *SkipTo(

)*, and the end rule *SkipTo(</p>)* (note: A different rule is required to reach the data block).
3. Iterate through the list (lines 2-5 in figure 14(a)) to break it into four individual records. To identify the beginning of each address, wrapper can start from first token of parent and repeatedly applies the start rule *SkipTo (<li>)* to the content of the list. Successive address starts from where the previous one ends. Similarly, we can use the end rule *SkipTo ()* to identify the end of each address.
4. Once each address record is identified, we need to use disjunctions. Possible disjunction *Start Rule* and *End Rule* can be as follows:

	Start Rule	*End Rule*
R1:	*SkipTo(( )*	*SkipTo( ))*
R2:	*SkipTo(-)*	*SkipTo()*

Once wrapper is generated, it is applied to other web pages that contain similar data and are formatted in the same way as the training examples. So, if the web site presentation changes, it needs to repair the wrapper. The verification of wrapper's validity in advance of any change is a big problem until we identity the garbage data. Secondly, it is not easy to be able to repair the wrapper automatically with the identification of change in webpage.

2.2.2 IEPAD: Information Extraction Based on Pattern Discovery

IEPAD (Chang et al., 2001) is one of the first IE systems that generalize extraction patterns from unlabeled Web pages. This method exploits the fact that if a

Web page contains multiple (homogeneous) data records to be extracted, they are often rendered regularly using the same template for good visualization. Thus, repetitive patterns can be discovered if the page is well encoded. Therefore, learning wrappers can be solved by discovering repetitive patterns. IEPAD uses a data structure called PAT trees, which is a binary suffix tree, to discover repetitive patterns in a Web page. Since such a data structure only records the exact match for suffixes, IEPAD further applies the center star algorithm (discussed in section 2.2(B)) to align multiple strings which start from each occurrence of a repeat and end before the start of next occurrence. Finally, a signature representation is used to denote the template to comprehend all data records. For example, in the following web page shown below contains repeating pattern and so can be used as input to IEPAD.

```
<html><title>xyz</title>
 <body> Book name </b> Data Mining
      <b>Reviews</b>
      <ol><li>
            <b>Reviewer name</b> Jeff
            <b>Rating</b> 2
            <b>Text</b> Some text-1
          </li>
          <li>
            <b>Reviewer name</b> Jane
            <b>Rating</b> 6
            <b>Text</b> Some text-2
          </li>
      </ol>
 </body>
</html>
```

By encoding each tag as an individual token (e.g., "T") and any text between two adjacent tags as a special token "T," IEPAD discovers the following pattern with two

*"
TTTT TT
"*

occurrences. The user then has to specify, for example, the second, fourth, and sixth "T" tokens, as the relevant data (denoting reviewer name, rating, and comment, respectively).

2.2.3 Instance based Wrapper Learning

Zhai and Liu (2007) introduced this instance-based wrapper learning, which is another approach for wrapper building without learning extraction rules. It extracts target items in a new instance/page by comparing the prefix and suffix token strings with those of the corresponding items in the labeled example. If some item in an unlabeled example cannot be identified, it is sent for labeling. The need for labeling is to identify and handling for any missing item in the page. Let us take an example for extracting "name", "image" and "price" of a product from a web page. The template (T_J) for this extraction can be represented as below:

$$T_J = \langle pat_{name}, pat_{image}, pat_{price} \rangle$$

Each pat_i in T_J consists of a '*prefix*' string and a '*suffix*' string of item '*i*', for example, if product image is embedded in the following presentation tag source:

... <table><tr><td> **<img>** </td><td></td> ...

then we can use the following pattern to identify token:

pat_{img} = (*img*, *prefix*:□<table><tr><td>□, *suffix*:□</td><td></td>□).

For each unlabeled example '*d*', it tries to identify every target item in '*d*' by matching the '*prefix*' and '*suffix*' tokens. It saves the candidate sequence of tokens which are partially matched and if a sequence of '*prefix*' and '*suffix*' tokens match uniquely, the targeted item is extracted. All those partially matched sequences are then discarded. If any token does not match it calls a function for labeling the token. The label page function tries to identify if the targeted token is missing or not. If it found it as missing, the entire token identifies as partially matched sequence. The key to instance-based learning is the similarity or distance measure. It measures whether an item in the new

page is similar to, or is of the same type as a targeted item in a labeled page. Figure 15 below shows how we can identify target item "price" from an HTML source:

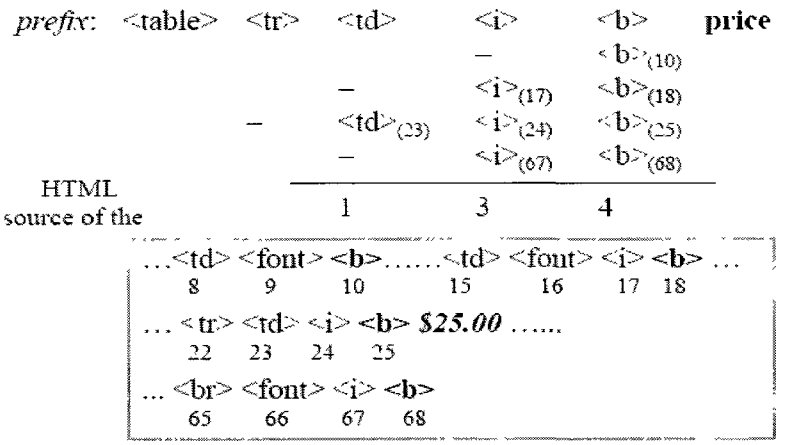


Figure-15: Similarity measure for identifying "price"

In figure 15, five tokens *"<table><tr><td><i><b>"* are saved as prefix string of item "price" from a labeled example. For a given HTML source shown in figure 15, we found four <*b*>'s, three <*i*><*b*> together, and only one <*td*><*i*><*b*> together, which match some prefix tokens of "price" that can be defined as "sufficient match" for identifying "price" item.

Wrapper induction techniques become more popular then wrapper language based approaches because of its freedom to use popular programming languages. But still it suffers for the requirement of heavy manual labeling of training pages. This is labor intensive, time consuming and needs regular wrapper maintenance effort. So, automatic wrapper generation for data extraction is becoming more popular over the years.

2.3 Automatic Wrapper Generation

Researchers studied the problems and limitations of wrapper induction and concluded that automatic or unsupervised extraction is possible, because data records in a web page are usually encoded using a very small number of fixed templates or patterns. Their study focused on two types of extraction process: Extraction based on single page and Extraction based of multiple pages.

For automatic extraction based on single list page, the entire web page code as a single string *'S'*, which contains k non-overlapping substring $s_1, s_2, s_3 \ldots . s_k$ with each s_i encoding an instance of certain set type and contain a collection of non-overlapping sub-substring of tuple type. For example, figure 06 (page 14) shows two data regions: Data Region 1 and Data Region 2. We can represent them by two substrings s_1 and s_2 of that web page string *'S'*. Substring s_1 contains a set of non-overlapping data records, $s_1 = enc_1 \{i_1, i_2, i_3, i_4\}$ and $s_2 = enc_2 \{i_1, i_2, i_3 \ldots\}$. Here, s_1 contains four encoding of data records, $enc_1 (i_j), j \in \{1, 2, 3, 4\}$ of tuple type σ_1. Similarly, substring s_2 contains data encoded records $enc_2 (i_j), j \in \{1, 2, 3 \ldots\}$ of tuple type σ_2. An algorithm needs to work on the string *'S'* to find each substring and construct the tuple type by generating a pattern from each substring representing the mark-up encoding function enc_i.

For extraction based on multiple pages (Grumbach and Mecca, 1999), input consists of a collection of *'k'* encoding, $enc (i_1), enc (i_2), \ldots, enc(i_k)$ of instances of type σ, where a collection of *'k'* HTML strings encodes *'k'* instances of same type. An algorithm works on the encoded instances and constructs a pattern.

2.3.1 **RoadRunner:** Towards Automatic Data Extraction from Large Web Sites

RoadRunner considers the site generation process as encoding of the original database content into strings of HTML code. As a consequence, data extraction is considered as a decoding process. Therefore, generating a wrapper for a set of HTML pages corresponds to inferring a grammar for the HTML code. The system uses the ACME matching technique to compare HTML pages of the same class and generate a wrapper based on their similarities and differences (figure 4, page 12). It starts by comparing two pages, using the ACME technique to align the matched tokens and collapse for mismatched tokens. There are two kinds of mismatches: string mismatches that are used to discover attributes (#PCDATA) and tag mismatches that are used to discover alignments, RoadRunner adopt UFRE (union-free regular expression) to reduce the complexity. The alignment result of the first two pages is then compared to the third page in the page class. In addition to the module for template deduction, RoadRunner also includes two modules, Classifier and Labeler to facilitate wrapper construction. The first module, Classifier, analyzes pages and collects them into clusters with a homogeneous structure, i.e., pages with the same template are clustered together. The second module, Labeler, discovers attribute names for each page class. iterators (+) and optional (?). Figure 4 (page 12) shows both an example of matching for the first two pages of the running example and its generated wrapper. Since there can be several alignments, RoadRunner adopts UFRE (union-free regular expression) to reduce the complexity. The alignment result of the first two pages is then compared to the third page in the page class. In addition to the module for template deduction, RoadRunner also includes two modules, Classifier and Labeler to facilitate wrapper construction. The first

module, Classifier, analyzes pages and collects them into clusters with a homogeneous structure, i.e., pages with the same template are clustered together. The second module, Labeler, discovers attribute names for each page class.

2.3.2 DEPTA: Data Extraction based on Partial Tree Alignment

Zhai and Liu (2006) introduced this enhancement of Simple Tree Matching algorithm (discussed in section 2.2.C) for web content data extraction. In their algorithm of partial tree alignment, authors align multiple DOM trees by progressively growing a seed tree (T_s). The seed tree is initially picked to be the tree with the maximum number of data fields. The reason for choosing this seed tree is to have a good alignment with data fields in other data records. Then, for each $T_i(i \neq s)$, the algorithm tries to find for each node in T_i a matching node in T_s. When a match is found for node $T_i[j]$, a link is created from $T_i[j]$ to $T_s[k]$ to indicate its match in the seed tree. If no match can be found for node $T_i[j]$, then the algorithm attempts to expand the seed tree by inserting $T_i[j]$ into T_s. The expanded seed tree T_s is then used in subsequent matching. Figure 16 shows aligning multiple trees:

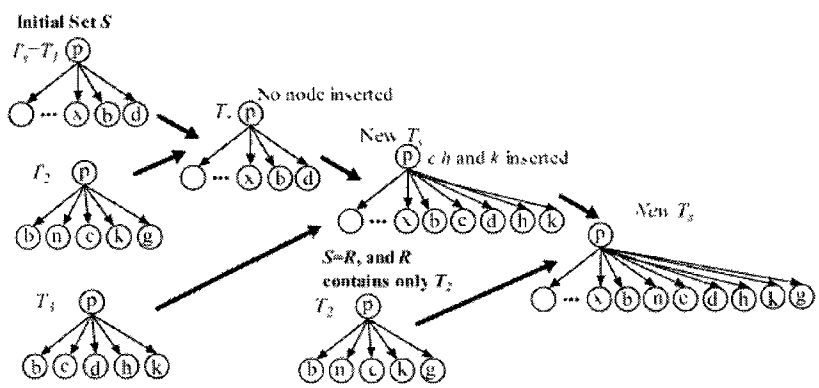


Figure-16: Iterative Tree alignment with two iterations

The partial tree alignment algorithm input a set of trees, S= {T_1, T_2 and T_3}. Initially it sorts S by descending order according to not aligned data items. then sets T_1 *as* T_s and remove T_1 from S. It then aligns each of the rest trees against T_s until end of trees in S. For next unaligned tree, the algorithm matches tree and finds all the matched pairs by tracing the matrix results. In Figure 16, T_s and T_2 produce one match, node b, whereas nodes n, c, k, and g have no matching nodes in T_s. it then attempts to insert them into T_s. In Figure 16, none of the nodes n, c, k, and g in T_2 can be inserted into T_s because no unique location can be found. So it inserts T_2 into R, which is a list of trees that may need to be further processed. Since T_3 is the last tree in S, when matching T_3 with T_s, all unmatched nodes c, h, and k can be inserted into T_s. Thus, T_3 will not be inserted into R and set "flag :=true" to indicate that some new alignments/matches are found or some unmatched nodes are inserted into T_s. It then check for stopping conditions: "S = 0, and flag = true", which means that we have processed all the trees in S, and some new alignments are found or insertions are done. Then, trees in R should be processed again. In Figure 16, T_2 is the only tree in R, which will be matched to the new T_s in the next round.

2.3.3 Modeling Web Documents as Objects for Automatic Web Content Extraction.

This is the theme paper of this thesis. Annoni and Ezeife (2009) proposed this idea of encapsulating heterogeneous web contents into object class hierarchy to extract and mine web contents in a unified way. All papers discussed so far in this related work of web content extraction rely on the web content presentation tree structure and extract only a limited targeted facts from the web page. The overall extraction and discovery of

all contents of a web page is not aimed. The main purpose of this paper is to propose a new data model for semi-structured web contents so that user can extract all kinds of heterogeneous web data together without loosing their relationships.

Abiteboul S. (1999) dreams for such a data model in Object Exchange Model (OEM). OEM is a directed graph containing object as vertices with unique object id (*oid*) and labels on the edge. Abiteboul S. (1999) tries to identify possible functionality requirements of this new data model and outlines web content data model as shown in figure 17. He defines two basic web content types: atomic type (e.g., integer, real, string,

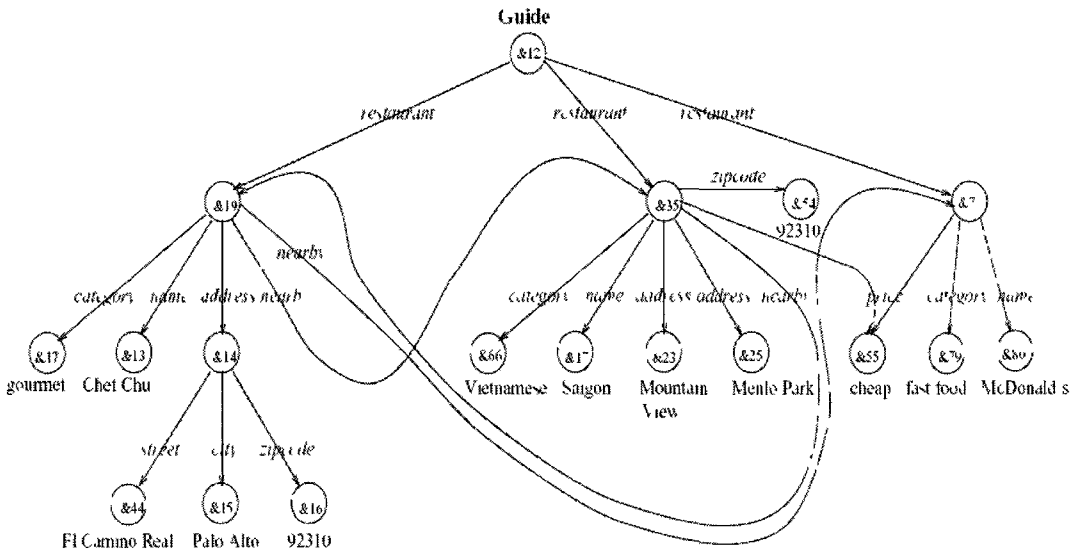


Figure-17: Object Exchange Model Graph (Abiteboul, S. 1999)

gif, html, audio, java, etc) and complex type (e.g., structure, table data, etc) and shows that OEM database can be viewed as a relational data with binary relation *VAL(oid,atomic_value)* to specify values of atomic objects and *MEMBER(oid,label,oid)* to specify values of complex objects. A suitable query language can be implemented to extract information from this OEM database model.

Annoni and Ezeife (2009) proposed object-oriented paradigm to model web data to capture both content and presentation objects of a web document. Toward mining web contents using object-oriented model, their paper have two major contributions for web content extraction: (A) They define and give framework of object-oriented data model and (B) They give the idea of how to extract web objects from the web page. They give a high level algorithm called OWebMiner() for web object extraction (figure 19, page 48) and an algorithm called ProcessPresentationSibling() for presentation (e.g., web page tag structure) object extraction process. Their anticipated use of presentation objects is to associate with content objects for mining process. Annoni and Ezeife's (2009) proposed framework for object-oriented data model is based on the following concepts:

1) They agree with Yu. et al. (2003) and Song et al. (2004) that related documents share same space and web page presentation tag structure. The web document segmentation work uses DOM tree, data location features and data presentation features to distinguish data blocks.
2) Unlike Yu. et al. (2003) and Song et al. (2004), Annoni and Ezeife (2009) proposed not to evaluate all HTML tags because all HTML tags are not always meaningful. They observed that main HTML tags (e.g., non-empty tags such as <table>, <link>, <form> tags) have impact in content and presentation and pre-formatting and in-line tags such as <pre>,
 should be avoided.

Annoni and Ezeife (2009) thus rely on DOM tree of web document and use "vision based context structure" for data x-coordinate and y-coordinate location of webpage features, web document zone, data's width, height and center location, and data presentation features such as style, type, fonts and spaces to identify data blocks. They also propose

not to evaluate the pre-formatting and in-line tags but they did not included any guideline or idea about how to filter these unwanted tags from the DOM tree in automatic content extraction.

Annoni and Ezeife (2009) define the web document zone to represent the entire web document as an object named *WebZone* object as shown in figure 18. A *WebZone* object is represented by *WebElement* and *WebRender* objects. So, from content view, a *WebZone* is a composition of *WebElement* objects which are divided into three zones: *HeaderZone*, *BodyZone* and *FootZone* as discussed in section 1.3.

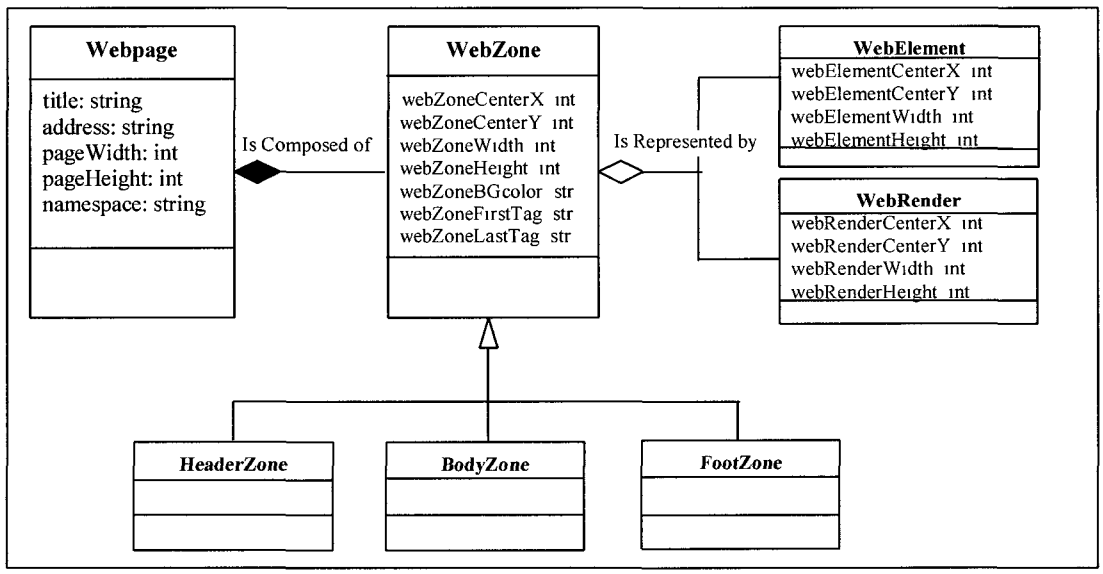


Figure-18: Hierarchy of web object model

They classify *WebElement* into six web content types by relying on four basic content types initially proposed by Levering and Cutler (2006): Text, Image, Form and Plug-in content. In addition to these four types they define two new types: Separator element and Structure element (discussed in section 3.2). Annoni and Ezeife's (2009) main algorithm OWebMiner() is given in figure 19:

```
Algorithm OWebMiner()

Input: a set of HTML files (WDHTMLFile) of web documents.
Output: a set of patterns of objects.

Begin
      For each WDHTMLFile
          (A) Extract web presentation objects and content objects
              sequentially with respect to their hierarchical dependencies.
          (B) Store the object hierarchies into a database table
      endFor
      (C) Mine patterns lying within objects
end
```

Figure 19: OWebMiner() algorithm (Annoni and Ezeife, 2009)

In this algorithm, they basically say that the algorithm will take a set of webpages (WDHTMLFile) and for each WDHTMLFile, line (A) of the algorithm will extract all the content and presentation objects into two separate object arrays according to their DOM hierarchical dependencies. Line (B) will store web objects into database. Line (C) will mine the extracted contents from the database.

They also developed sub-algorithm (A) of their main algorithms OWebMiner() called PresWebObjectScan() and ContWebObjectScan(). ContWebObjectScan() uses array data structure ContentObjectArray[] to store content objects. Process began with root of DOM Tree "<html>". When it hits series-1 (discussed in section 1.3), it calls algorithm PrecessContentSibling() (modified version is given in figure 46) to start extraction of content objects and continue until it hits series-2 (discussed in section 1.3). ProcessContentSibling() algorithm inputs DOM Tree, a pointer called "TTag" which indicate current tag to process in DOM Tree, ContentObjectArray[] and a variable "indTag" which is a global index for labeling content objects per zone. The algorithm recursively traverses DOM tree block-level tags by depth-first search until it hits non-block level tag and reset "TTag" pointer to represent current processing tag. If depth-first

search hits a non-block level tag, it processes all it's siblings into an array called "tagArray". For all non-block level tags in "tagArray", the algorithm then associates a content object to tag value. Otherwise it recursively calls itself to advance "TTag" pointer. The algorithm finally returns the ContentObjectArray[] with full content objects from body zone of web page. Annoni and Ezeife (2009) stops at this point in their paper and left the remaining mining from the content object array as future work.

3. OO Web Content Mining

As discussed in section 2.3.3, Annoni and Ezeife (2009) proposes object-oriented data model for extraction and mining of heterogeneous web contents. They gave the framework for web content elements, web presentation elements, and an algorithm (called OWebMiner) for extraction of web objects. The entire architecture of the system, definition of data base schema and mining technique were pending to develop. We studied their work and propose two-level mining process for knowledge discovery. This thesis develops the architecture (we call it WebOMiner) for web content mining using object-oriented model. It develops, extends and modifies necessary algorithms for WebOMiner system. It also defines the data base schema and gives guideline for automatic database schema generation. This thesis addresses the following problems in Annoni and Ezeife's (2009) work toward development of WebOMiner system.

3.1 Problem Addressed

1. ProcessPresentationSibling and ProcessContentSibling algorithms proposed by Annoni and Ezeife (2009) called from their OWebMiner algorithm splits presentation and content objects from DOM tree and store into separate flat array data structures in sequential order as per their hierarchical dependency. So, all content objects are added into the array sequentially until the end of body zone. Within DOM tree all related data are structured as data block but in their flat array data structure, content data are loosing their relationships. Their algorithm does not address the requirement for identification of data block and data region. It is important to extract related data together or create clear separation between data blocks and data regions. We address this in our thesis in section 3.4.1.

2. They proposed "vision based context structure" to locate data using x-coordinate and y-coordinate location features as discussed in section 2.3.3. This feature is useful when using browser rendering engine, but for automatic extraction process without use of web browser, co-ordinate location of any feature is not possible.

3. Annoni and Ezeife (2009) defines separator element as follows:

> *"Spaces between contents which emphasize them and make them instinctively meaningful for human beings such as line, blank and empty space. They could be enclosed within HTML tags <hr>,
."*

This definition is ambiguous and specific purpose is not clear. They did not discuss about how this separator element will be used and their algorithm did not address the use of separator element in content or presentation object extraction. We define the use of separator element for identification of data block and data region in our problem context as discussed in section 3.4.3.

4. They did not define the object classes, size of object classes, object class hierarchy, object class dependencies, and functionalities of object classes. They only classify the web content elements but did not associate object types with contents, nor discuss how to control the creation of expensive objects.

5. Annoni and Ezeife (2009) did not address the issue of associating leaf level tags with specific contents. A leaf level tag contains important information about the associated content. It is important to associate leaf level tags before assigning an object to a content type. For example in a data block there are three image tags as shown below:

<img ***id= "line"*** *src= "http://....."* ***alt = "line"*** */>* ……(1)
<img ***id= "monitor"*** *src= "http://....."* ***alt = "monitor"*** */>*……(2)
**……(3)

Here HTML tag at line (1) and (2) have three tag attributes: "id", "src" and "alt". line (3) have two attributes: "src" and "alt". Tag attributes are variable inside a tag and each attribute should have a value. First image tag of line (1) is a line separator as identified from the value of attribute "id" and "alt", second image tag of line (2) is for "monitor" as identified from "id" and "alt" attribute and the third image tag is for "Add to Cart" hyperlink identified from "alt" attribute. If we don't care about tag attribute of a source image, we will not be able to identify the image we want. We resolved this problem by analyzing tag attribute in our thesis.

6. Annoni and Ezeife (2009) did not address the issue of preventing noisy data entry into database table. Their algorithm does not refine contents before entering into database table. We address this issue by cleaning noises from data tuple.

Our approach to address these problems is discussed in next consecutive sub-sections of this chapter as: web content objects in section 3.2, challenges for extraction and mining in section 3.3, thesis problem domain and approach to solution in section 3.4, Mining technique in section 3.4.3, our proposed architecture of the system and algorithms in section 3.5 and warehouse and mining for integration in section 3.6.

3.2 Web Content Objects

The State of the Art for the web data model proposed by Annoni and Ezeife (2009) is the wrapping of web data in objects to use object-oriented approach which they claim enables to mine in a unified way. It raises the problem of classifying web contents in object type. A complete list of web content types do not exist yet and over the evaluation and advancement of technology, user demand, business and marketing demand, new type of contents are adding over time. So, they rely on the following four

basic content types proposed by Levering and Cutler (2006). We also use these content types in our thesis:

3.2.1 Text content: These are the textual web content data found in the leaf level of the DOM tree. These could be raw text with or without alignment or the List text in ordered or unordered form. A simple format of web content data are given in figure 20:

```
<html>
  <head>
    <title>CS 60-140</title>
  </head>
  <body>
    <div align= "center">
      <h2>Intro. Algorithms</h2>
      <b>by C. I. Ezeife</b>
    </div>
    <p> This is the text book for
"Introduction to algorithm" course of
computer science department, University of
Windsor, Canada</p>
    <p align = "left">
      copyright@cs.uwindsor.ca
    </p>
  </body>
</html>
```

```
<body>
        Basic requirements of this course are to
fulfill the following tasks:
      <ul><li> Four assignments. </li>
          <li> Two mid term exam. </li>
          <li> One Final exam. </li>
      </ul>
Students are required to attend the following lab
sessions for this course.
      <ol><li> Problem-01 of page 34 </li>
          <li> Problem-10 of page 99 </li>
          <li> Problem-06 of page 120</li>
      </ol>
Some tips for the students:-
       <dl><dt>Want good grade?</dt>
          <dd>Go to every class.</dd>
          <dt>Want job soon ?</dt>
          <dd>Do all assignments and labs</dd>
       </dl>
</body>
```

Figure-20: Example of simple static web textual data

3.2.2 Image content: Image or pictures embedded into the web documents are image contents. There are two types of image content in web documents, image and map. Image is a simple picture referring to a physical image document in any physical location. For example, or , , are simple links to different formatted image files at different physical locations that are embedded into the web document. When an image is associated with a mapping defined by HTML tag, it is called the image map. For example, in case of client side mapping:

```
<map name = "brainmap">
        <area shape="rect" cords="15, 15, 220, 100" href="fantacy.htm" />
</map>
<img src = "brain.gif" usemap="#brainmap" />
```

Similarly, for server side mapping <ismap> attribute is also used.

3.2.3 Form content: Web page forms are normally used to gather information from web page users such as user feedback about any topic related to web content, orders through internet, other information from the reader of the web page. These form contents are enclosed within the keyword tag <form> and different input formats are used to gather the information. For example, by the HTML tag <textarea>, users are allowed to type any command or textual information, within the tag <select>, users are allowed to select any pre-defined option from a set of options by <option> tag. By check box (i.e., <input type= "checkbox" >), a user can select one or more pre-defined options.

Similar approach is used for long time in case of Dynamic web pages by interactive pages. Programmers are using popular programs like Java applets, JavaScript, CGI programming, php programs, ActiveX control for dynamic interaction with the web page readers to gather information from the user or help user to get required information.

3.2.4 Plug-in content: Plug-in contents are dynamically generated contents in the web pages from either server side database or automated calculation by the functions or programs. Two types of computer programs are used for Plug-in contents; client-side and server-side programs. In case of client-side programs, the controlling computer functions or programs along with database accessibility are embedded into the webpage, so it is more vulnerable in terms of database security. Server-side programs which are embedded into the webpage are usually interacted with another controller program at server and

generate the dynamic contents supplied by the server. Within HTML embedded CGI, php, visual basic program codes are example of plug-in contents. For example:

CGI Code: *< -- #command exec ="scriptName" -- >*
Visual Basic Code: *<% program %>*
Php code: *<? php program ?>*

Annoni and Ezeife (2009) uses these four basic web content types as discussed above; moreover they propose two additional content types "Separator" and "Structure" in their literature. They define the "Separator" element as spaces between contents such as line <ln>, blank <&nb> and empty spaces <tb>. They could also be enclosed within HTML tags horizontal rule line <hr />, a line break
 etc. The "Structure" element, they mean the database generated structured data of different content types. There is no specific tag associated to structure content. It can be generated within any HTML tag.

3.3 Challenges and Thesis Approach to Solution

We face a set of challenges toward implementation and development of object-oriented web content extraction and mining algorithms as discussed below:

01. Requirement of a crawler algorithm that can automatically identify the positive web pages (e.g., web pages within our problem domain) from the WWW and the functionality for extraction of HTML document, its contents including image-files, video-files into local directory. We developed mini-crawler and extractor algorithm that crawls into given input URL or a set of input URLs that sequentially extract HTML documents including image-files into user defined local directory. Our crawler algorithm does not have the functionality for automatically identification of positive web pages (e.g., product list pages as per section 1.2.3 in our problem domain) for data extraction from WWW.

02. Majority of HTML documents in WWW are not well-formed as per W3C standard. A well-formed document structure is the pre-condition for building DOM tree. Current commercial vendor specific HTML code cleaning software's (like java HTML tidy or "tidy" by fourceforge.net) are not robust enough to handle most of the current commercial web pages. We use free open source software "tagsoup" ("http://home.ccil.org/~cowan/XML/tagsoup/", licensed under Apache License, version 2.0) for embedding missing closing tags. We modified its functionality to make ill-formatted web pages well-formed and to exclude inline tags.

03. Inline and decorative tags are another problem for extraction of content data and for DOM creation. Decorative inline tags split contents in DOM tree. For example the following "<a>" tag encloses a single line of text.

<a> This is a test example for DOM tree.

But in HTML document, this text may be represented in different ways for attraction to viewer like the following:

<a> This is a test example for DOM tree.

Hare "" tag is used in part of the text to view in italic font and tag is used to bold only the "DOM" word. The DOM representation is shown in figure 21 below. Here original text content is split into four parts in two different levels in DOM sub-tree. When we traverse through DOM for object creation, it creates four different text objects without structural relation. To overcome from this problem, we need to filter unnecessary decorative/formatting tags and inline tags. The use of Java DOM filter

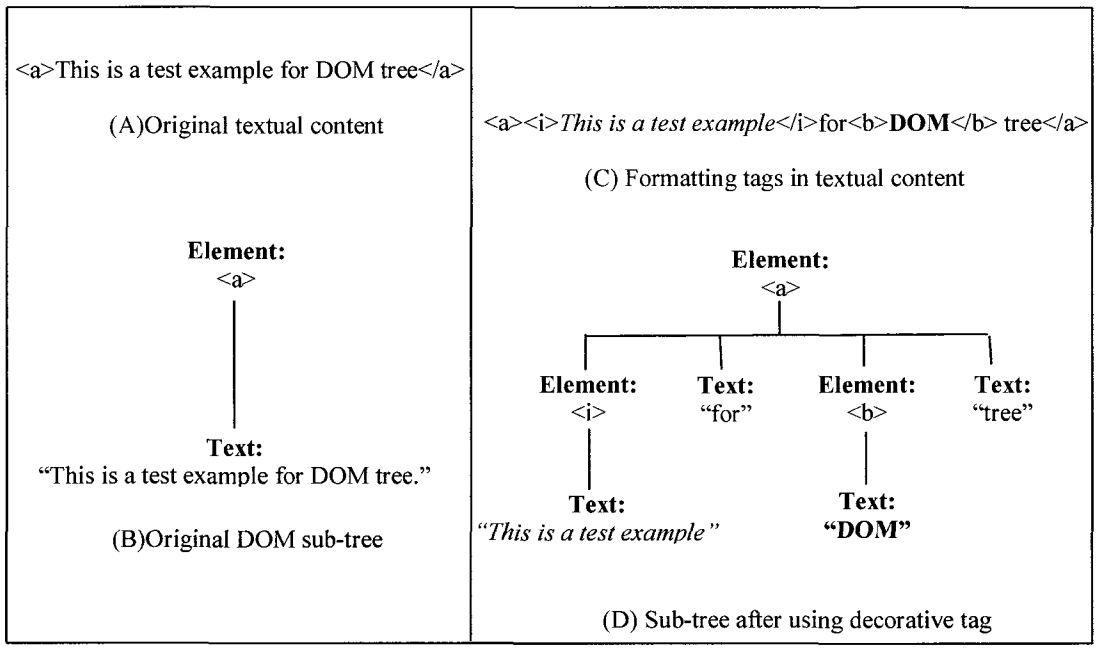


Figure-21: Formatting tags within textual fragment

class filters-out the selected element from the DOM tree including its contents that is unwanted. So, we modified "tagsoup" module to filter out decorative tags from HTML tag structure.

04. Java DOM package itself can not handle real time long tag attribute values. For example in the following "<a>" tag is from "compUSA.com" website.

"<a href="http://compusa.shoplocal.com/compusa/default.aspx?action=entry& pretailerid=-98291&siteid=147&adref=[any&storeID=2595483">"

Here "href" attribute contains a long string value. Within this "href" attribute value there are five "=" signs and Java DOM parser looks for value enclosed by " " for each equal signs while creating DOM tree. Further improvement of our filter is required in the future to handle these types of long attribute values. We therefore used simplified version of mirrored or simulated commercial web pages by reducing length of tag attribute value to test and evaluate our system.

05. Schema matching in web data context is a challenge for information integration. For example, two schemas for customer C_1 and C_2 in figure 22 are for the same information from two different web pages.

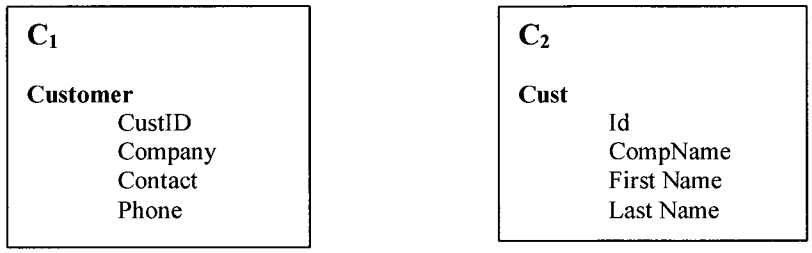


Figure-22: Difference in schema for similar information

To match these two schemas a representative schema mapping is required. Schema matching mostly relies on semi-automatic matching in specific domain. Researchers addressed this problem with domain specific approach. In our problem context, schema matching is handled during object creation.

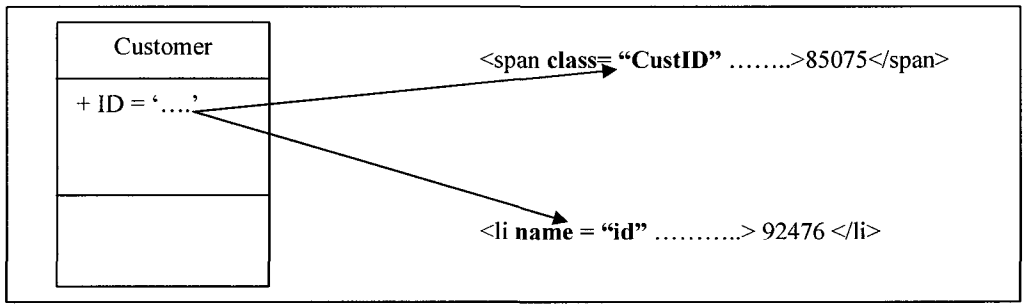


Figure-23: Schema matching at object creation

In this thesis, we use linguistic approach (e.g., equality in names or synonyms) for mapping synonym with 1: m match cardinality for string matching. For example, "Customer" object attribute "id" in figure 23 is mapped to "CustID" and "id" in two different types HTML tag attribute during object assignment and then store respective value in object. This ensures the consistency in database entry.

3.4 Problem domain

For the specific domain of B2C websites, we have selected to mine a most common data-rich web page, which is the product list page (discussed in section 1.2.3). From the common B2C webpage structure shown in figures 05 and 06 (page 13 and 14), product list webpage is commonly a data rich page. We observed that, a product list page usually contains brief list of all or specific types of products. There is a set of product list pages in a B2C website. We define a product list page as follows:

Definition 1: *If 'w' is a B2C website and 'p' is a webpage in 'w' such that* $w = \Upsilon_{j=1}^{n} p_j$, *then a page* $p_j \in w$ *where* $j \geq 1$, *is a product list page iff* 'p_j' *contains a set of tuples* τ *of type* α, *where* $\alpha \geq 1$, *having distinct instance type.*

In case of our running example shown in figure 06, it consists of a set of content data blocks that are arranged in different data regions (discussed in section 3.4.1). One region usually contains similar categories of data. Advertisement region contains hyperlinks with a set of services. Main product data region contains a set of data blocks. Each data blocks are hyperlinked (by "MORE INFO") with separate product details page and contains some key information like image of the product, product name, product number, brand or manufacturer, product price and a hyperlink for shopping the product. This information is defined as instances or objects of distinctive type. This page also contains other blocks like list of other products and services (e.g., navigation block), advertisements (e.g., noise block). Our target is to pick up this key information systematically from each data block and store in a data base for mining.

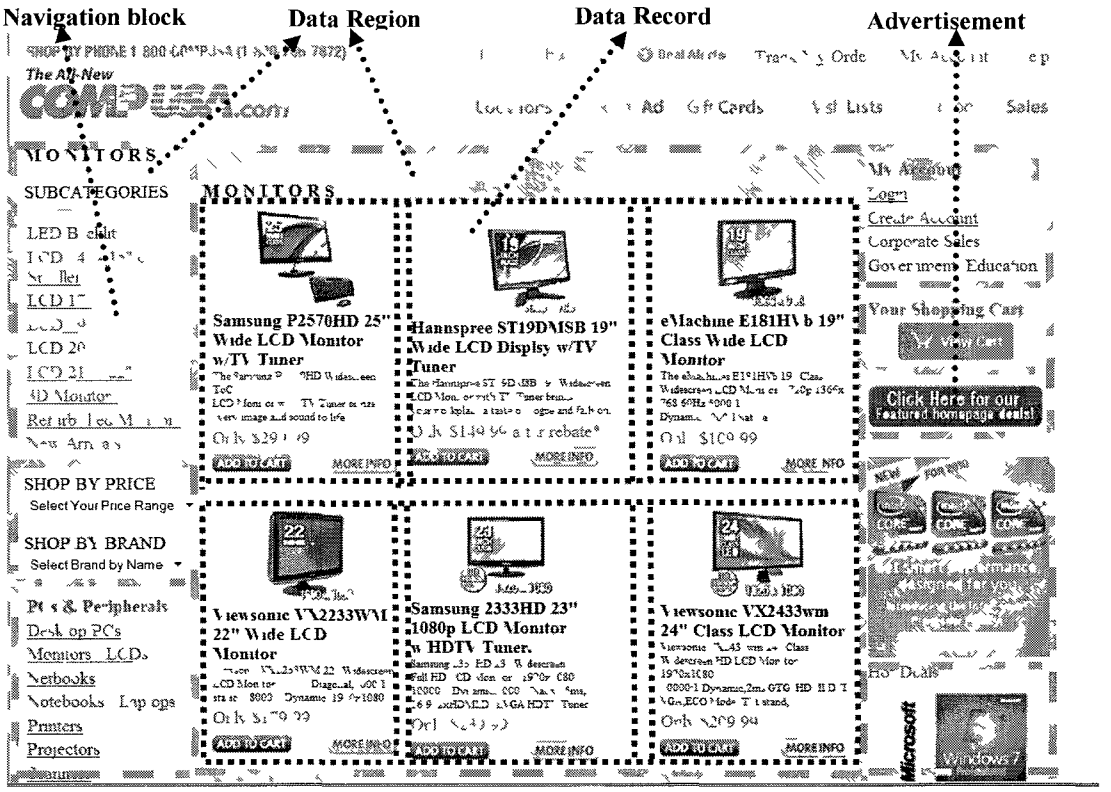


Figure-06: Data regions and data blocks.

There is no easy way to pick up this information. This set of key information is similar for almost every product list pages but their schemas may be different.

3.4.1 Data Region and Data Block Identification

Annoni and Ezeife (2009) defines HTML tags into two categories; "block-level" tags and "non-block" level tags. They define a block level tag as HTML tags that are either the child of sub-tree "body" or another block-level tag and should be the parent of other tags. For example: <table>, <head>, <body>, <p>, <li>, <form>. A non-block level tag is an inline tag or text level tag which is the child of a block level tag and mainly lies in DOM tree leaf. A complete set of block-level and non-block level HTML tags does not exist. In dynamic web pages, designers are allowed to define their own tags using the tag

library. Moreover, block-level and non-block level tags are not disjoint. Same tag can be used as block-level or non-block level tag.

In case of our running example, the DOM tag tree (figure 08 at page 22), data blocks are represented in nested table at lines 105, 113, 121, 131, 139 and 147. Each of this block-level '<table>' tag contains a set of block-level '<tr>' and '<td>' tag. In some cases, block-level and non-block level tags may not necessarily be disjoint as shown in figure 24, where <td> tag is used both as block-level and non-block level tag.

```
<table> <!- - Subcategories listing begins - ->
      <tr><td>Item Number </td>                    //Non-block level <td> tag
          <td><a href = "http://....." >           //Block-level <td> tag
                  <span class = "prodtitle" ....> LCD Monitor </span>
              </a></td>
          <td> .....</td>
          "
      </tr>
      "
</table>
```

Figure -24 Intersection of block level and non-block level tag

A data region and data block is enclosed by one to many block level tags. There is no easy way to identify this region and data blocks. In this thesis, we define the following three important observations to identify data regions and data blocks based on DOM tree.

Observation 1: *If* Γ *represents the DOM tree of product list page* $p_j \in w$, *and* $\Re_1, \Re_2, \Re_3, ..., \Re_n$ *represents data regions in* Γ, *then* $\Re_i$ *is a sub-tree of* Γ *and* $\forall \Re_i \in \Gamma$ *are disjoint.*

Observation 2: *A data region* $\Re_i$ *consists of a set of data records* τ. *All data records* τ *in a data region* $\Re_i$ *typically represent similar list of objects and* $\forall \tau_i \in \Re_i$ *are contiguous in* $\Re_i$.

Observation 3: *All data records τ of any data region $\Re_i$ are formed by some sub-tree of same parent node and are disjoint.*

Observation 1 states that, in our problem domain, if "Γ" represents a DOM tree of an entire web page tags including contents, then it contains a set of data regions. They are disjoint and are sub-trees of Γ. Our running example (figure 08 at page 22) shows data region nodes at line 7, 35 and 193. Observation 2 states that, a data region in DOM Tree consists of a set of data records (defined as tuple in this thesis) and all data records in a region, in general, represent similar set of data and are contiguous in a data region. In case of our running example, data region of line 7 have two similar navigation data blocks at line 8 and 20, region of line 35 have a set of eight similar data blocks at lines 58, 69, 105, 113, 121, 131, 139 and 147. All these data blocks are contiguous in sub-tree of data regions. Observation 3 states that all data records in a data region formed by same parent node data records are disjoint in DOM tree. To explain this, we graphically represented (partially) our running example DOM tree in figure 25 below. Here, "Γ" represents the root of the DOM tree. This page has three disjoint data regions $\Re_1$, $\Re_2$ *and* $\Re_3$; all are sub-tree of "Γ". Data region $\Re_2$ has eight contiguous data records $\tau_1, \tau_2, \ldots, \tau_6$ Data records τ_1, τ_2 are similar and $\tau_3, \ldots, \tau_8$ are similar.

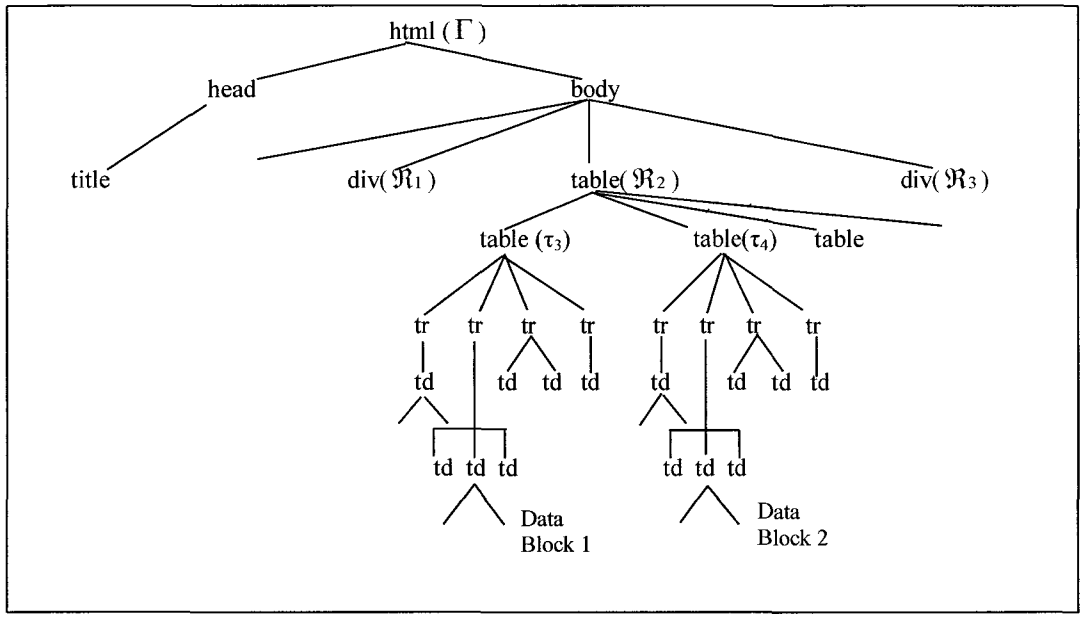


Figure-25: Graphical tree representation of data block and data region.

Figure 25 shows the disjoint characteristics of data regions and data blocks. We observed that a data region or data block can be within any block level mark-up tag but usually lies within tags like , <table>, <tr>, . This set is not complete and intersecting with non-block level tags. Observation of positive page tag structure is helpful to identify the region and data block. We denoted the region and data block by the set notation '{', '}'. In our case, we used and <table> tag as region and data block.

3.4.2 Data Model

Structured data of web page are generally encoded with HTML mark-up and in nested relation. Data records or blocks are related information about any fact. For example, figure 06 (page 14) shows six data records about computer monitors in a single data region. Each record consists of a set of information like image of monitor, brand name, model number, short description, retail price, etc. All these information are related

to a single entity and so six data records in figure 06 represents six distinctive entities. Grumbach et al., (1999) defined data block in nested relation as follows:

- There is a set of **basic types**, $B = \{B_1, B_2, \ldots, B_k\}$. Each B_i is an atomic type, and its domain, denoted by $dom(B_i)$, is a set of constants;
- If $T_1, T_2, \ldots, T_n$ are basic or set types, then $[T_1, T_2, \ldots, T_n]$ is a **tuple type** with the domain $dom([T_1, T_2, \ldots, T_n]) = \{[v_1, v_2, \ldots, v_n] \mid v_i \in dom(T_i)\}$;
- If T is a tuple type, then $\{T\}$ is a **set type** with the domain $dom(\{T\})$ being the power set of $dom(T)$.

We will use similar notations for our data model. We used set notation '{', '}' for representing the data records or blocks. In the context of web content, B_i is usually a text string, image-file, price as string (of type long) representing distinctive related instance. For example, a product data record can be represented as in figure 26:

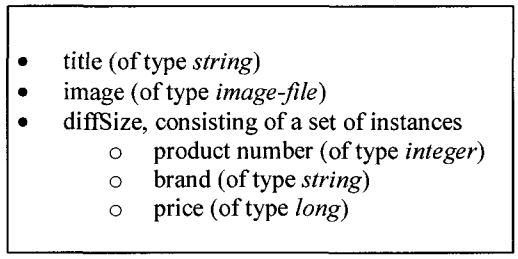


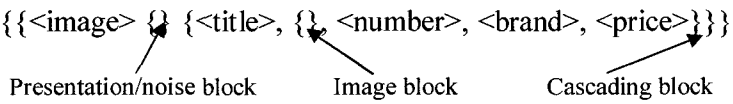
Figure-26: Data block representation (Grumbach et al., 1999)

The set format of mark-up encoded product data can be denoted as:

{<title>, <image>, {<number>, <brand>, <price>}}

Here, the tag '<title>' is not representing the mark-up tag itself but the content encoded by the mark-up tag with value of attribute name or class or id = "title". This data format is not unique and can be different in mark-up encoding and in nested formation for different data regions and web page structures. A nested data block or record usually

contains some additional contents like decorative image, "Add-to-Cart" image or button including link, "More Info" image, link to product details page. These additional content information in general, can be treated as noise content in data block and need to be cleaned up. For example, the same product data can be in the following format in the web page:



We define a data block as data tuple when a data block's nested relation are collapsed to flat relation and clean-up any unwanted instances like separator object for nesting inside a data block. For example, when we will collapse the instances of the above data block and clean-up the internal nested noise block, image block and any cascading block, the resultant data tuple will be as follows:

⟨ <image>, <title>, <number>, <brand>, <price>⟩

We used the notation '⟨' and '⟩' to denote a data tuple τ We define the data tuple as

Definition 2: *A tuple τ is a domain type dom(τ) which consists of a set of distinct related instances of atomic or basic type, $B = \{B_1, B_2, B_3, ...B_k\}$ in flat mark-up encoding relation.*

A mark-up encoding is a pair of mark-up tags *open-tag* '<>' and *close-tag* '</>' respectively. Mark-up encoded data instances reside in the leaf level of tree type encoding and each instance or attribute of a tuple can be encoded differently to distinguish them or unrelated catalyst instance (e.g., decorative <img>) may be used to highlight the importance. A tuple τ denoted by notation '⟨'& '⟩' can be written as, $\tau = \langle B_1, B_2, B_3,....B_k \rangle$. In the context of web content, B_i is usually a text string, image-file,

price representing distinctive instance. The product data tuple of figure 26 can be modeled as shown in 27:

```
Product ⟨ title:           string;
          image:           image-file
          product number:  integer;
          brand:           string;
          price:           long;        ⟩
```

Figure-27: Data tuple of product data block

3.4.3 Tuple formation from Data Block

As stated by Annoni and Ezeife (2009), we observed that a product list webpage contains six basic types of content data blocks. These are Product data block, List or Navigation data block, Form data block, Text data block, Decorative/Singleton data block and the Noise / Advertisement data block. We need to identify data tuple from these content data blocks.

A Product data block is an important data block in product list page. Related information of a typical product data block are: an image of the product, the name or title of the product, product number, brand, and price. Additional information like rebate in tagged price, brief description of the product, etc may exist and not necessarily all page contains all the information. These information or elements are found as either ordered or un-ordered list and in flat or nested HTML tag encoded relation. The set format of product data block in nested relation is denoted as below:

{<image>, {<title>, <number>, ... , <brand>, <price>}}

Some pages may contain less information like: {<image>, <title>, <brand>, <price>}

According to Annoni and Ezeife (2009), these information need to be identified and assign respective object to them (i.e. image object for image element, text object for

text element, etc). Their anticipated use of separator element / object is not clear and their ProcessContentSibling() algorithm did not define or give guideline for the use of separator objects. We redefine the use of separator object to identify data regions and data blocks. Therefore, in object view, proposed product tuple (e.g., a flat product data block after cleaning) will look like the following figure 28:

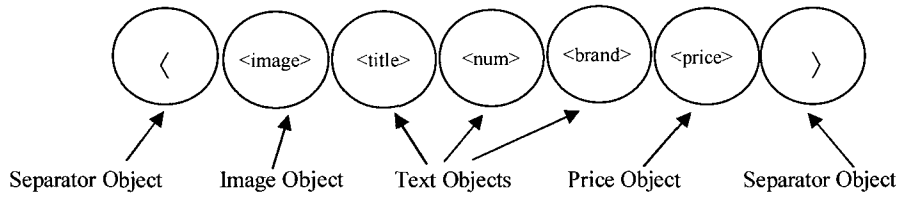


Figure- 28: Content objects of a product data block.

The identification of a data tuple is not easy task. Objects of a data block may exist in different level in a DOM tree. For example in case of our running example the data blocks are as follows:

```
<table><tr> .....</tr>  // Data block
        ”
        ”
      <tr>                                    // Data block
        <td><a href = “.....” ><img src = “.....” /></a></td>
        <td>
          <table><tr><td>   </td>
                  <td><a href = “.....” ></a>
                    <span class = “prodtitle” ...> Title content<br></span>
                    <span class = “prodspec” ...> Specification <br></span>
                    <span class = “prodprice”...> Price      <br></span>
                      ”
                      ”
                  </td></tr>
          </table></td>
      </tr>
      <tr><td>.....</td></tr> //Data block
</table>
```

Figure-29: Example of simple content hierarchy in a data block.

To resolve the problem of identifying data blocks, we used Separator element/object and classified Separator element in two categories: open-separator, denoted by set notation

symbol '{' and close-separator denoted by '}' symbol. An open-separator element represents some predefined block-level open-tag (like <table>, <tr>,) which are candidate tag for root of data block and data region. Similarly, their end-tag (e.g., </table>, </tr>,) is represented by close-separator element. For example, if we represent <table> tag as open-separator and denote it by the set notation '{', and </table> tag a close-separator and denote by a set notation '}', then the product data block of figure 29 can be represented as: {<image> {<title>, <specification>, <price>}}

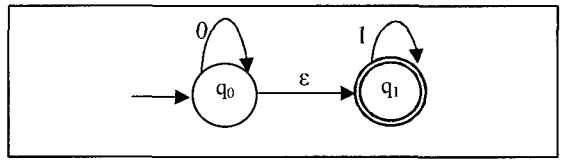
For tuple formation of this data block, it needs to flatten/collapse and keep all object instances at the same level. So interior set notation should be deleted and the outer set notation is replaced by the tuple notation '⟨' and '⟩'. It also needs to clean up the noise block, null block, cascading set notations within the data blocks to build it as tuple. Some websites use price contents as image to highlight the importance. For example, future shop's price tag is like below:

<img id = "**pricepill**" src = "http://....." alt = "**$124.99**"></img>

it may contain other images like, an image for "add to cart" option, "more info" image, other presentation images, block separator images. So it is a challenge to pick up the product image from these different images in a data block. One approach is to match the "alt" attribute of image tag to identify the product image.

For tuple identification, we use a Non-deterministic Finite Automata (NFA) based approach of pattern matching. Details of this NFA formation is discussed in section 3.5.4. An NFA is a finite state machine where number of state is finite and for each pair of state and input symbol there may be several possible next states by consuming input symbol or

without consuming any input symbol by epsilon transition (ε) For example, a NFA for any string beginning with '0's followed by '1's will be as follows:



Any string like "0001111" or "0111" will be accepted by this finite state machine but any string like "01011" will be rejected by it. A finite state machine as defined above is a classical mathematical abstraction used to design digital logic or computer program and can solve a large number of problems An epsilon transition (ε) in an NFA allows a transition from one state to another without consuming any input symbols. This epsilon transition is important to allow transition from one state to several states to consume distinctive symbols. In our case the NFA representation of a product tuple can be represented as figure 30:

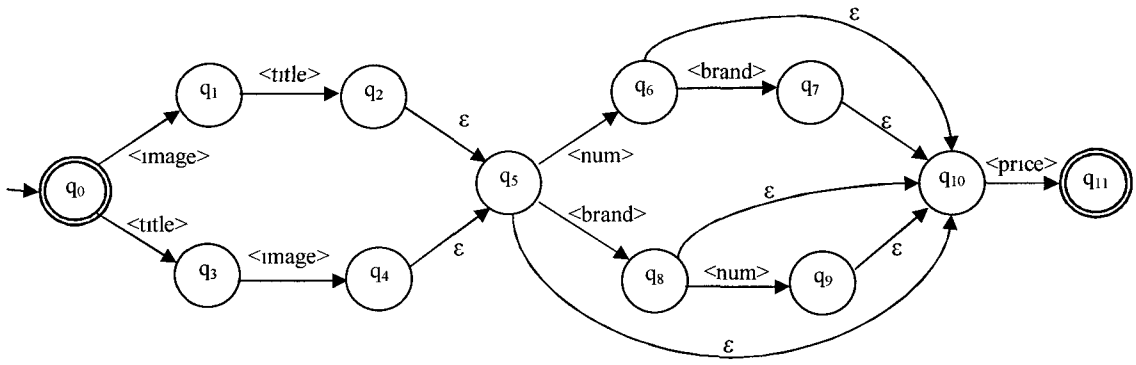


Figure-30: NFA notation for product tuple

This product tuple NFA can be mapped to the following 10 schemas:

Product (title string, image image-file, prodNum string, brand string, price long),
Product (title string, image image-file, prodNum string, price long),
Product (title string, image image-file, brand string, prodNum string, price long),
Product (title string, image image-file, brand string, price long),

Product (title:string, image:image-file, price:long);

Product (image:image-file, title:string, prodNum:string, brand:string, price:long);

Product (image:image-file, title:string, prodNum:string, price:long);

Product (image:image-file, title:string, brand:string, prodNum:string, price:long);

Product (image:image-file, title:string, brand:string, price:long);

Product (image:image-file, title:string, price:long);

The List tuple contains a set of hyperlinks and their related title. List tuple usually redirects the web page users to different resources of the web site. The common format of the List tuple is as follows:

〈<link>, <title>, <link>, <title>, <link>, <title>, <link>, <title>.....〉

For example, in figure 08 at page 22, line 8 to 19 will generate a List tuple where line 8 and 19 will create open brace "〈" and close brace "〉". Line 9 to 13 contains a series of <a> tag with its text contents. Here <a> tag is of type <link> and related contents are of type <title>. So, the resultant tuple looks like the format shown above. This tuple consists of a set of <link> and <title> tags in ordered format. We observed that this pair of tags usually contains at least three to unknown finite length. We redefine this tuple as: $Y_{j=1}^{n}(<link>,<title>)_j$. This expression is useful for NFA generation, because a web site contains set of List tuples with various lengths. The NFA representation of this expression is shown in figure 31:

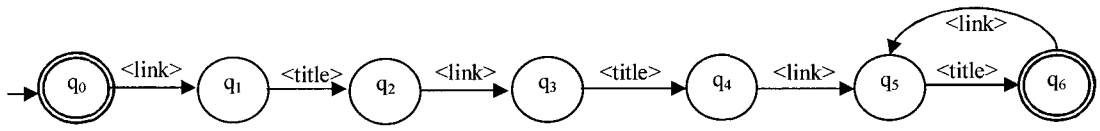


Figure-31: NFA presentation of List tuple

This List tuple can be mapped to the schema: *List (link: string, title: string);*

The Form tuple may be of different kinds. This usually takes the user input as text, or selection of a specific option and is normally user event driven. All content information of a form tuple is contained under the block level tag <form>. The block level <form> tag has important information about the action of the user event. We therefore need to extract attribute information from the <form> tag and defined the Form tuple start with <form> tag followed by a set of leaf level texts under non-block level tag <option>, <select>, <input> and / or <textarea> with unknown length. This tuple looks like the following: ⟨<form>, <text>, <text>, <text>.....⟩

Here, <form> tag is added as key identifier as a part of content object. There are two reasons for using this identifier: it will distinguish the Form content from Text content, and <form> tag contains two important attribute information about its texts; "name" and "action". The "name" attribute gives the information about its embedded texts and the "action" attribute gives us the URL of the webpage where the action will be triggered. We redefined this tuple as $<form>\Upsilon_{j=1}^{n}(<text>)_j$. The NFA representation of this expression is shown in figure 32:

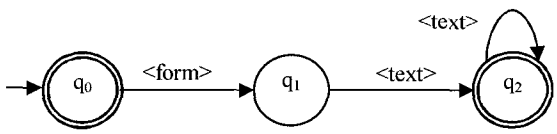


Figure-32: NFA presentation of Form tuple

This Form tuple will map to following two schemas:

Form (form: boolean, name: string, action: string);
FormContent(name: string, text: string);

The Text Tuple may contain raw text in the web page or a bag of text describing something. The tuple may contain a set of Text objects as follows:

⟨ <text>, <text>, <text>, <text>, <text>..... ⟩

We redefined this Text Tuple as $Y_{j=1}^{n}(< text >)_j$. The segmentation of this text instances needs further research in case of problem domain that contains bulk text or text corpus. The NFA representation of this expression shown in figure 33 can be mapped to the schema: *Text (text: string);*

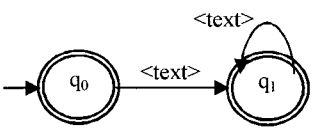


Figure-33: NFA presentation of Text tuple

The Noise / Link Tuples are a set of hyperlink with image. The tuple can be represented by ⟨ <link>, <image>, <link>, <image> ..... ⟩. We redefined tuple as: $Y_{j=1}^{n}(< link >, < image >)_j$. NFA representation of Noise/Link tuple shown in figure 34 can be mapped to the schema: *Noise (link: string, image: image-file);*

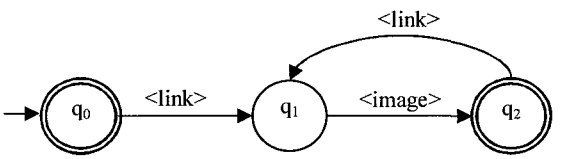


Figure-34: NFA presentation of Singleton tuple

A Singleton Tuple can be anything for presentation purpose. Sometimes some stand alone attractive images with or without links are used in web pages for better representation or make the presentation attractive. This tuple can be represented by ⟨ <image> ⟩ or ⟨ <link>, <image> ⟩ So, a Singleton tuple may have intersection with Noise/Link tuple.

3.5 *Proposed "WebOMiner" Architecture and Algorithms*

We developed the architecture for extraction and mining of web contents using object-oriented model. We call it "WebOMiner" which is shown in figure 35 below:

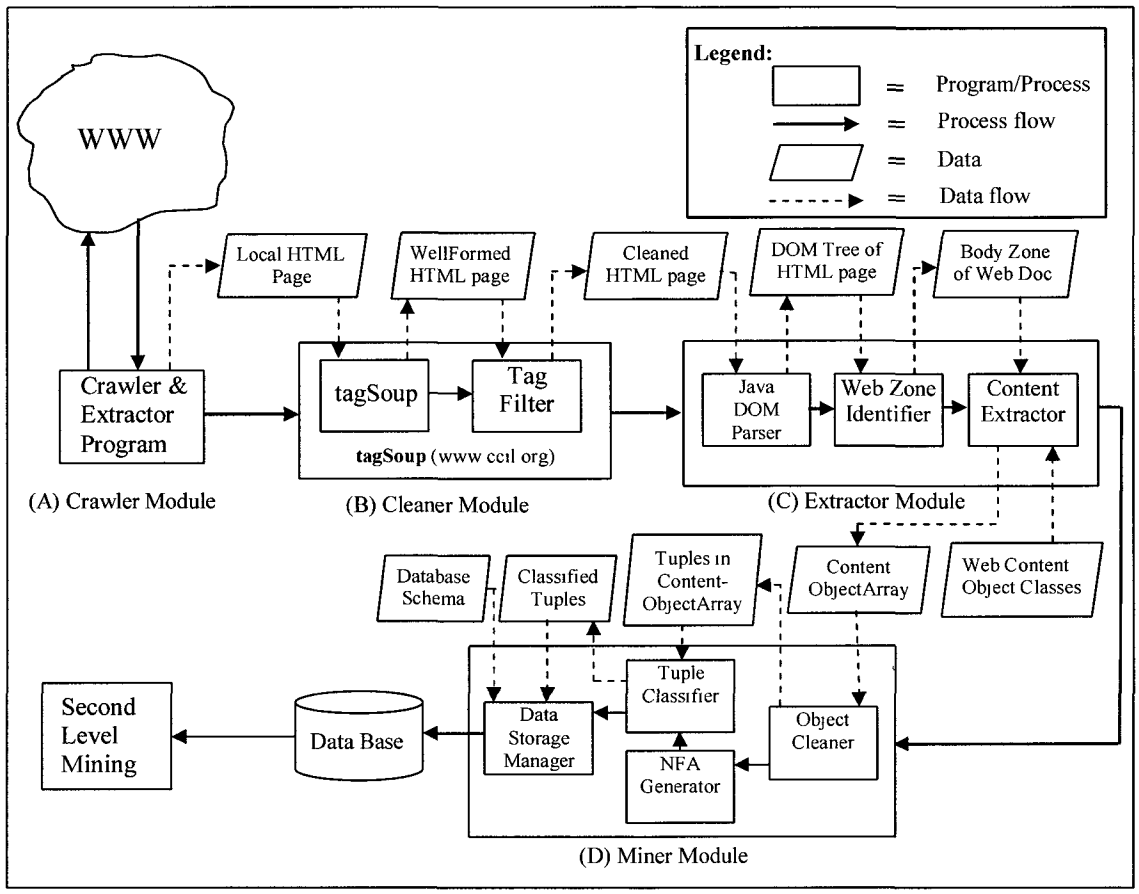


Figure 35: WebOMiner Architecture for Object-Oriented web content mining.

This architecture (figure 35) has four modules: (1) Crawler module (2) HTML cleaner module (3) Content extractor module and (4) Web miner module. These modules are called sequentially by our main algorithm WebOMiner (shown in figure 36).

```
Algorithm Main
    Input:      Set of HTML files (WDHTMLFile) of web documents.
    Output:     Set of patterns of objects.
    Variable:   ContentObjectArray[].

Begin
      For each WDHTMLFile
          A.  Call SiteMapGenerator() to crawl and extract webpage into local directory  from
              WWW.                                                    /* figure 38 */
          B.  Call tagSoup.html() to clean-up HTML code.      /* called by running test.bat file*/
          C.  Call OWebMiner.BuildDOMTree() to create DOM tree of refined HTML file and
              extract web content objects sequentially from DOM Tree. Store objects in
              ContentObjectArray[].                                   /* figure 43 */
          D.  Call MineContentObject.IdentifyTuple() to identify data records and classify
              records  according to  their pattern.                   /* figure 49 */
          E.  Call CreateDBTable() to store data records into a database table
      endFor                                                          /* figure 55 */

          F.  Mine for knowledge discovery within extracted contents. /* pending to develop */
End
```

Figure-36: WebOMiner main algorithm

We now will explain below the modules of our system and will discuss how our WebOMiner algorithm works.

3.5.1 Crawler Module:

We developed a mini-crawler algorithm that crawls through the WWW to find targeted web page, streams entire web document including tags, texts and image contents and it then creates a mirror of original web document in the local computer. Our crawler module dumps the comments from the HTML document. That means it have the functionality to exclude all comments from the web documents. The class diagram of crawler module is given in figure 37 below.

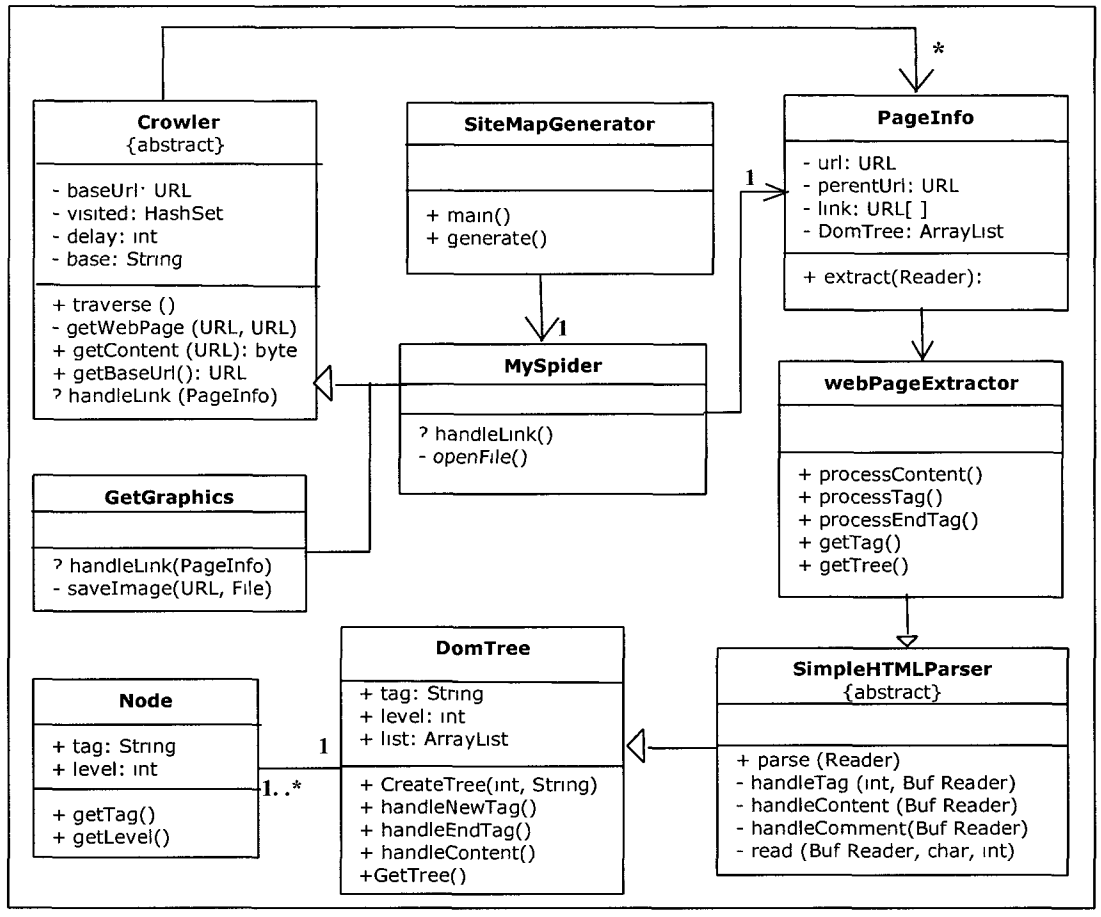


Figure-37: Class diagram of Crawler module

The WebOMiner algorithm line-A starts with calling generate () method of class "SiteMapGenerator" (figure 38). This class contains a private class called MySpider that inherits abstract class Crowler. This algorithm takes a URL string as input and outputs HTML file in local machine. The algorithm sets the input URL string as BaseURL and its "generate()" method calls MySpider's super class method "traverse()" by passing BaseURL. The result outputs an ArrayList of Nodes having tags and contents of HTML file. Node information is then written into the output HTML file.

```
Algorithm SiteMapGenerator.generate()
    Input:  URL String
    Output: null
    Begin
        1. Set baseURL variable = URL String.
        2. PageIngo = Call private class MySpider.traverse() by passing baseURL
                                        //calls super class Crowler.traverse() method
        3. Call MySpider.handleLink() by passing PageInfo object
    End

    Algorithm MySpider.handleLink()
        Input:  PageInfo Object
        Output: HTML file in local computer
        Begin
            1. Arraylist = Call PageInfo.getTree()    //Calls superclass getTree()
            2. do
                    - Extract Node information
                    - Populate buffer string with level
               Until end of Arraylist
            3. Create or Open output file using openfile() method
            4. Write buffer string into output file
        End
```

Figure-38: Algorithm SiteMapGenerator.generate() and MySpider.handleLink()

"traverse()" method is the main method of the abstract class Crowler that creates Http connection for network data transfer and calls "extract()" method of PageInfo class.

```
Algorithm Crowler.traverse()
    Input:  baseURL
    Output: PageInfo object

    Begin
        1. Set delay time and sleep time for network data transfer.
        2. Create HttpURL Connection using baseURL.
        3. Varify connection validity by responseCode, contentType and contentLength.
        4. PageInfo = Call PageInfo.extract() method passing InputStreamReader object.
        5. Close InputStreamReader object.
        6. Close HttpURL Connection.
    End
```

Figure-39: Crowler.traverse() Algorithm

"extract()" is the main method of PageInfo class that inputs InputStreamReader object and returns PageInfo object to the caller class. This method verifies the network connection, contentLength and calls inherited "parse()" method of WebPageXtractor

class by passing the Reader class object. It then calls inherited "getTree()" method of WebPageXtractor class which returns an Arraylist created by "parse()" method.

```
Algorithm PageInfo.extract()
    Input:    InputStreamReader
    Output:   PageInfo object

    Begin
        1.  Check validity of HttpURL connection, content length, socket timeout.
        2.  Call WebPageXtractor.parse() method by passing Reader object
                    //Calls super class SimpleHTMLParser.parse() method
        3.  ArrayList = Call WebPageXtractor.getTree() method that calls superclass
                    DOMTree.GetTree() method.
    End

Algorithm WebPageXtractor.getTree()
    Input:   Reader object
    Output: ArrayList

    Begin
        1.  ArrayList = call DOMTree.GetTree()
        2.  Tokenize content and add to ArrayList
        3.  Tokenize Tag and add to ArrayList
        4.  Tokenize EndTag and add to ArrayList
        5.  Extract <a> tag attribute "href"
        6.  Extract <image> tag attribute "src"
    end
```

Figure-40: PageInfo.extract() and WebPageXractor.parse() algorithm

WebPageXtractor (figure 40) also has some additional methods processTag(), processEndTag(), processContent(), extractHref(), extractSrc() to process the HTML tag, end tag, content, and to extract "<a>" tag attribute "href" and "<image>" tag attribute "src".

SimpleHTMLParser (figure 41) is an abstract class that has "parse()" method which manages the incoming data stream from network and parses by looking ahead of incoming data to determine the type of data stream and handles the data as per their type using the methods handleTag(), handleContent(), handleComment(). Whenever it identifies any comments in the incoming data stream it dumps them. This class uses

```
Algorithm SimpleHTMLParser.parse()
    Input:   Reader object
    Output: null

    Begin
        1. Set 10 characters to read in buffer[] and set sleep time.
        2. Set read-ahead marker to buffer[] index to 3 advance character.
        3. Read input stream until get a tag
        4. if input stream is tag
                - if input stream start with '</', it is endTag
                        i. Append to buffer string until symbol '>'
                        ii. Set type = SimpleHTMLToken.ENDTAG
                        iii. Call DOMTree.createTree(type, bufferstring).
                - Else
                        i. Append to buffer string until symbol '>'
                        ii. Set type = SimpleHTMLToken.TAG
                        iii. Call DOMTree.createTree(type, bufferstring).
            Else
                - if first 3-character are ' < ! - '
                        i. reset read-ahead pointer to original position
                        ii. dump the comments.
                - Else
                        i. Append to buffer string
                        ii. Set type = SimpleHTMLToken.CONTENT
                        iii. Call DOMTree.createTree(type, bufferstring).
    End
```

Figure-41: SimpleHTMLParser.parse() algorithm

enumeration "SimpleHTMLToken" to mark the incoming data type. It inherits the "CreateTree()" method of DOMTree class (figure 42) to encapsulates the data into node object by maintaining tag hierarchy and then adds those objects into ArrayList which it returns to the caller method to write the HTML file into local directory.

```
Enumeration SimpleHTMLToken
    Begin
        Static final TAG = 0;
        Static final ENDTAG = 1;
        Static final CONTENT = 2;
        Static final UNDEFINED = -1;
    End

Algorithm DOMTree.CreateTree()
    Input:  type, string
    Output: null

    Begin
        1.  Check type of input string
        2.  if type = TAG
                2.1 Set input at same level and put into Node.
                2.2 Add Node to ArrayList.
            Else
                2.3 Set input into next level and put into Node.
                2.4 Add Node to ArrayList.
        3.  if type = ENDTAG
                3.1 Set input in previous level and put into Node
                3.2 Add Node to ArrayList
        4.  if type= CONTENT
                4.1 Set input in same level and put into Node
                4.2 Add Node to ArrayList.
    End
```

```
Algorithm DOMTree.GetTree()
    Input:  null
    Output: ArrayList
    begin
        -Return populated ArrayList
    end
```

Figure-42: Algorithm DOMTree.CreateTree()

3.5.2 HTML Cleaner Module:

WebOMiner algorithm line-B calls tagSoup.html() method to start cleaning of a given webpage. "tagsoup" module is an open source software under the Apache license and available from "http://home.ccil.hangorg/~cowan/XML/tagsoup". We used it with some modifications to clean-up the HTML code and make DOM tree well-formed. Our changes in tagSoap are noted below:

(1) File:- "/src/definitons/html.tssl"

- *Removed all <attribute name='{{attributeName}}' default='{{defaultValue}}'/> they were inserting default attributes that weren't present in the webpage.*
- *Line 2166 Added <contains group='M_INLINE'/>*
- *Line 2167 Added <contains group='M_BLOCK'/>*

- *Allow the <a> tag to contain other tags that it normally wouldn't, and act more like it does in browsers*

(2) File:- "/src/java/org/ccil/cowan/tagsoup/CommandLine.java"

- *Line 87:*

  new dst = src.substring(0, j) + "_.html";
  old dst = src.substring(0, j) + ".xhtml";
  changed the the name of the generated output file

(3) File:- "src/java/org/ccil/cowan/tagsoup/XMLWriter.java"

- Function "startElement" line 573 in

  add condition for elements to be removed
  - if it is to be removed, don't write element to file
  - if it isn't
  add condition for self closing element
  if it is self closing write "/>" instead of ">"
- Function "endElement" line 629

  add condition for elements to be removed
  - if it is to be removed, don't write closing element to file
- Line 1177

  new -> char ch[] = atts.getValue(i).replaceAll("\"\"$","").toCharArray();
  old -> char ch[] = atts.getValue(i).toCharArray();
  remove "" (2 double quotes) from end of attribute value
- function "writeEsc" line 1221

  remove switch case that was replacing characters with escaped codes
  add check to make sure characters were valid ascii
  add check to find " (double quote) in attribute value
  replace it with ' (single quote)

These changes in tagsoap module reflect our need for inserting missing tags at appropriate location, handles and removes inline tags
, <hr/>, inserts missing "/" at the end of un-closed <image> tag, clean up unnecessary decorative tags. The result is a refined HTML page in local directory.

3.5.3 Content Extractor Module:

Content Extractor Module creates the DOM Tree from HTML page and extracts the contents from the DOM tree, assigns respective objects as per pre-defined object class

to the contents and sets information into objects and finally puts objects into ArrayList. It also identifies the data regions and data block and used separator object to segment the respective data of a data blocks from other data blocks. We use Java DOM package to create and parse DOM Tree of the webpage.

Our WebOMiner() algorithm line-C calls OWebMiner.BuildDOMTree() method which is given below in figure 43:

```
Algorithm OWebMiner.BuildDOMTree()

    Input:   Refined HTML file of web documents.
    Output:  Populated ContentObjectArray[].

    Begin
            1.  Use Java DOM Package to create DOM Tree.
            2.  Call ContentObjectArray() to identify series-1 and series-2
    End
```

Figure 43: OWebMiner.BuildDOMTree algorithm

We modified Annoni and Ezeife's (2009) ContentWebObjectScan() algorithm as per our requirement to catch body zone content objects according to their definition (discussed in section 1.3). The modified version of ContentWebObjectScan() algorithm is given in figure 44 below.

```
Algorithm ContentWebObjectScan (DOMTree, ContentObjectArray[])

Input:     DOM Tree of the web document, ContentObjectArray[].
Variable:  Pointer series1, series2, TTag;
           Int tagCount, numTag, count, indTag;
           ArrayList SiblingArray[];
Output:    Populated ContentObjectArray[]

Begin,
   1. ContentObjectArray[] = null;
   2. set TTag = "<body>" tag of DOM Tree
   3. numTag = DOMTree.getLength();          // getLength() returns total node of DOM Tree
   4. while(tagCount ≤ int(0.5*numTag))
      4.1 if( TTag is not block-level tag  AND TTag starts with "<a" AND series1 = null)
          4.1.1 repeat
                - Store TTag into SiblingArray
                - Store TTag Siblings into SiblingArray
                until end of Sibling
          4.1.2 for each TTagSibling in SiblingArray
                if (TTag starts with "<a")
                    - count++;
                endfor
           4.1.3 if (count ≥ 5)
                    - series1 = TTag.parent;
                    - break;
                endif
       4.2 else
                - TTag = TTag.next as per DOMTree depth first search
                - tagCount++;
      endwhile
   5. while (tagCount < numTag)
       5.1 if(Ttag is not a block-level tag AND (tagAttribute="CopyRight" OR "PrivacyPolicy"))
                - series2 = TTag.Parent;
                - break;
       endwhile
   6. Call ProcessContentSibling ( series1, DOMTree, ContentObjectArray, series2, indTag);
End
```

Figure-44: Modified ContentWebObjectScan algorithm

This modified version of ContentWebObjectScan() algorithm (figure 44) identifies the starting and finishing point of BodyZone as per definition of Annoni and Ezeife (2009) and sets series-1 and series-2 pointer in the DOM tree. For our running example DOM tree, the ContentWebObjectScan() algorithm sets the TTag initially to the "<body>" tag at line 6 of the DOM tree. We intentionally set it from "<body>" tag to avoid all embedded program code and style sheet information of the web page within "<head>"

tag. Process begins with initializing an array called ContentObjectArray[], when it reaches at line 9 "<a>" tag, it identifies series-1 by scanning siblings of "<a>" tag. The algorithm also identifies series-2 by keyword "<a>" tag attribute "PrivacyPolicy" or "CopyRight", it complies with the definition of "Foot zone" by Annoni and Ezeife (2009).

Line 6 of ContentWebObjectScan() algorithm (figure 44) calls ProcessContentSibling(), which is also a modified version of ProcessContentSibling() algorithm initially defined by Annoni and Ezeife (2009). Our modification of this algorithm is to reflect the identification of data regions and data blocks by using separator element. Modified version of this algorithm is given in figure 45.

In case of our running example DOM tree (shown in figure 08, page 22), this algorithm starts storing content objects into the ContentObjectArray[] until it hits the Foot zone by identifying series 2. Here, series-1 is set to TTag (current pointer at DOM tree) at line 7, which is a " " tag (region node). The algorithm hits at line 2.2 of figure 45 and calls CheckTagObject() of figure 46., this algorithm creates an OpenSeparator object and stores it into the ContentObjectArray[]. TTag is then set to the next child tag " " at line 8 (data block node) and similarly stores another OpenSeparator object into ContentObjectArray[]. The TTag is again set to its child node "<a>" at line 9 and the algorithm recursively calls itself. Since it is a non-block level tag, the algorithm hits at line 1 of figure 45 and stores respective "<link>" followed by "<image>" objects for all five siblings (line 9 to line 17) into ContentObjectArray[] as per line 1.2.2 of the algorithm. Line 19 ends a data block and the algorithm stores a closing separator object

```
Algorithm ProcessContentSibling (TTag, DOMTree, ContentObjectArray, series2, indTag)

Input:   TTag is the HTML tag value which its sibling will be processed
Output:  ContentObjectArray populated by content objects form DOMTree

begin
  1. if TTag is not a block-level tag
      1.1 repeat
              1.1.1 Store TTag in tagArray
              1.1.2 Store TTagSiblings found in tagArray
          until end of TTag sibling
      1.2 for each TTag sibling in tagArray
              1.2.1 if TTagSibling is a block-level tag
                      - Associate respective content object to tagArray[TTagSibling index-1]
                      - Store this object in ContentObjectArray
                      - increment indTag
                      - TTag = next TTagSibling
                      - Call recursive
                        ProcessContentSibling(TTag,DOMTree,ContentObjectArray,indTag)
              1.2.2 else
                      - Associate respective content object to tagArray[TTagSibling index]
                      - Store this object in ContentObjectArray
                      - increment indTag
          endFor
  2. else
       2.1 If (TTag = series2) return;
       2.2 Call CheckTagObject(TTag, DOMTree, ContentObjectArray)
       2.3 TTag is set to next node of DOMTree by depth-first search
       2.4 Call ProcessContentSibling(TTag,DOMTree,ContentObjectArray,indTag)
       2.5 TTag is initialized to next node of DOMTree by breath-first search
     endIf
```

Figure-45: Modified ProcessContentSibling() algorithm.

into the ContentObjectArray[]. Similarly, line 20 starts another data block which ends at line 33. Line 34 ends this data region. Line 35 starts with another data region that ends at line 192. Line 58 and 69 are two text data blocks "SHOP BY PRICE" and "SHOP BY BRAND" as shown in left pane of figure 06 at page 14. These embedded tags and contents are hidden in figure 08. Similarly, line 105, 113, 121, 131, 139 and 147 are six monitor data blocks embedded into hidden tables as shown in figure 08.

```
Algorithm CheckTagObject (TTag, DOMTree, ContentObjectArray)

Comments: This algorithm checks for data block and data region

Input: TTag is the HTML tag value
Output: Null
Variable: OpenTag Enumeration {<table>, <div>, <tr>, <span>}    /* Set of tags usually represents parent
          CloseTag Enumeration {</table>, </div>, </tr>, </span>}   node of data block and data region */
begin
  1.0 if TTag is in OpenTag Enumeration
        1.1 Create an instance of OpenType Seperator element
        1.2 Set attribute value = "{"
        1.3 Store this object in ContentObjectArray
  2.0 Else if TTag is in CloseTag Enumeration
        2.1 Create an instance of OpenType Seperator element
        2.2 Set attribute value = "}"
        2.3 Store this object in ContentObjectArray
  3.0 Else if TTag is "<form ……..>"
        3.1 Get attribute "action" value
        3.2 Create an instance of Form class element
        3.2 Set tag value = <form> and action attribute value
        3.3 Store this object in ContentObjectArray
  4.0 Else
        4.1 return
  end
```

Figure- 46: Algorithm to insert separator object in ContentObjectArray

When the algorithm hits at line 193 it gets series 2 pointer and returns the populated ContentObjectArray[] to the main algorithm WebOMiner() (figure 36, page 74). Two partial snapshots of this ContentObjectArray[] are given in the following figure 47:

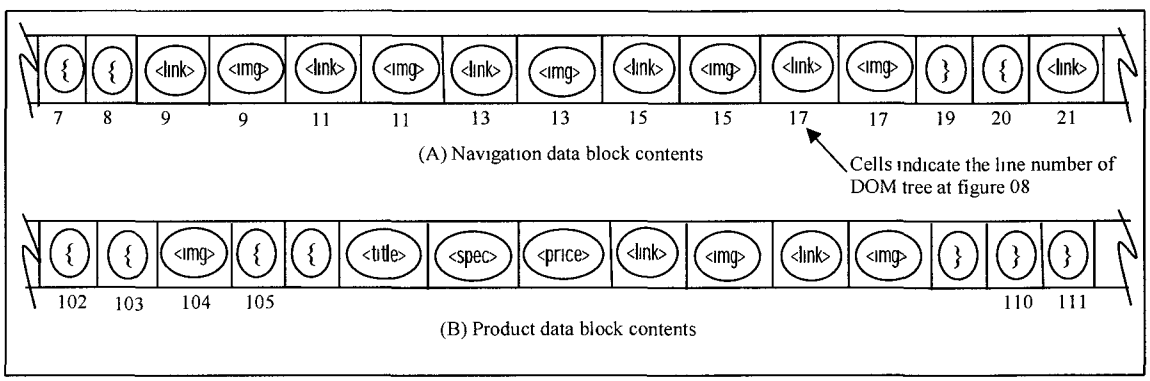


Figure 47: Snapshot of ContentObjectArray[]

3.5.4 Web Miner Module:

Line-D of our main WebOMiner algorithm (figure 36) mines populated ContentObjectArray[] for identification of data blocks and their classification to make contents ready for database entry. Line-D starts with calling our mining algorithm MineContentObjects() (shown in figure 48, page 86). It inputs populated ContentObjectArray[] and outputs a set of content patterns ready to input into database table for content integration. Line 1.0 of this algorithm does the vital job. It scans ContentObjectArray[] for open and close separator object and identifies candidate tuples by matching key objects and minimum support. It then refines separator objects by deleting themselves.

```
Algorithm MineContentObjects(ContentObjectArray)

Input: ContetObjectArray            // Data structure contains content objects
Output: A set of patterns of object's contents

begin
1.0  Call IdentifyTuple() /* Generates NFA & return ContentObjectArray
                             with tuples by refining separator objects; */
2.0  For each tuple in ContentObjectArray
        2.1  Copy objects in TupleList as per generated NFA;
        2.2  Call SqueezeTuple() to refine object tuple;
        2.3  Store Squeezed tuples in a list according to their categories;
     endFor;
3.0  Calculate support for each tuple category;
4.0  If targeted category satisfy the minimum support
        4.1  Call CreateDBTable.insertData() each tuple in data table;
     endIf;
end;
```

Figure- 48: Algorithm to Mine Content Objects

At the same time, this algorithm generates Seed NFA pattern for data blocks. Line 2.1, 2.2 and 2.3 extract objects of all tuples by matching with refined NFA and store identical tuples into TupleList. Line 3.0 counts tuples and checks support for all tuple categories and if they satisfy the support, line 4.1 stores objects into relational database.

In case of our running example, the mining algorithm MineContentObjects() inputs the entire populated ContentObjectArray[] (partial snapshot is shown in figure 47). The algorithm starts with calling another algorithm IdentifyTuple() as shown in figure 49.

```
Algorithm IdentifyTuple(ContentObjectArray[])

Input:   ContentObjectArray[]
Output:  Set of Tuples in ContentObjectArray
Other variable: pointer header, prev, current, DoubleLinkedList PointerArray, Enum PatternTable
begin
    1.0 For each object in ContentObjectArray
        1.1 If object is type open-separator element
            1.1.1 Create a PointerArray node
            1.1.2 If pointer header is null
                        - Refer header = node, current = node, prev = node
                        - node.next1 = current object of ContentObjectArray
                    else
                        - current.next2 = node
                        - prev = current
                        - current = current.next2
                        - node.next1 = current object of ContentObjectArray
        1.2 Else if object is type close-separator element
            1.2.1 Boolean flag = CheckMinSupport();
            1.2.2 If flag = true
                        - Replace current object notation to close-tuple notation
                        - Replace current.next1 notation to open-tuple notation
                        - GenerateSeedNFA();
                        - PointerArray.current.next1 = null
                        - prev.next2 = null
                        - Reset PointerArray count to null
                    Else
                        - Destroy current object
                        - Destroy PointerArray.current.next1 object
                        - set current = prev
                        - set prev.next2 = null
        1.3 else
                    - Increment count of respective objects in Pattern Table
    endFor
end
```

Figure-49: Algorithm for identifying object tuples

This algorithm scans for objects in the ContentObjectArray[]. In case of snapshot A of figure 47, the algorithm hits at line 1.1 and creates a pointer node and points to open-separator object as shown at cell 7 (object of line number 7 from DOM tree) of figure 50.

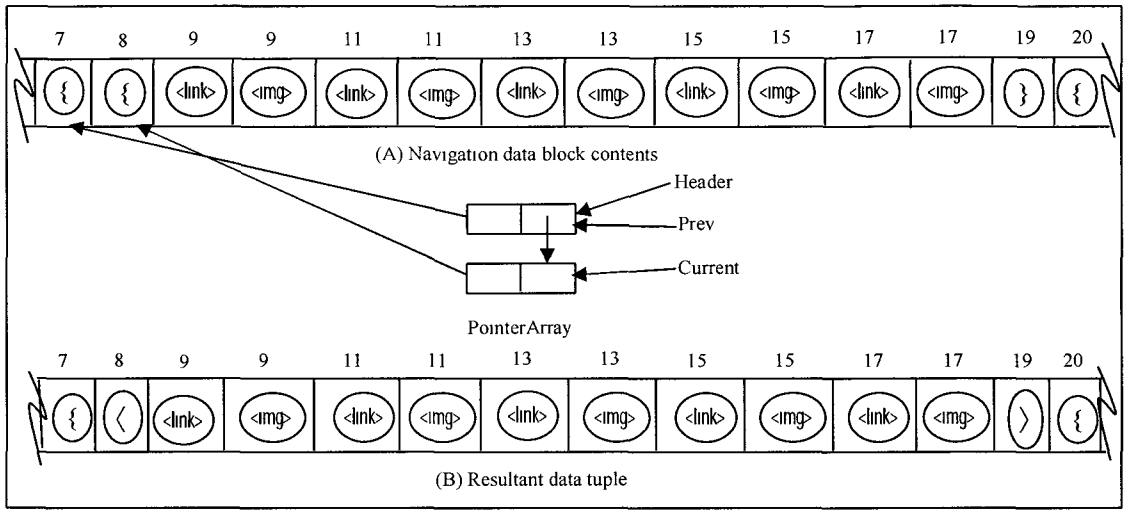
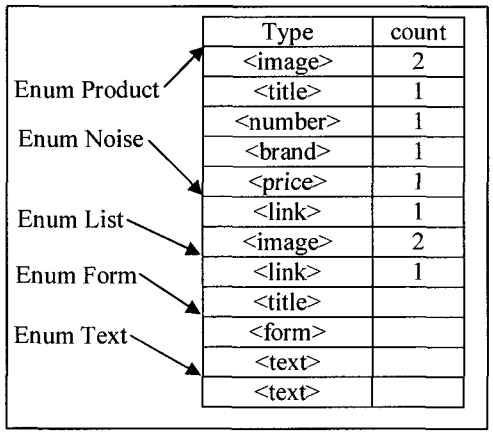


Figure 50: Identification of Data block

At next iteration it scans another open-separator object at cell 8, creates another pointer node and points next to previous one as per line 1.1 of the algorithm. This node points to cell 8 as shown in figure 50. Successive iterations scans a set of repeated pattern of <link> and <image> objects from cell 9 to 17, thus the algorithm increment the count of a data table called "PatternTable" as shown in figure 51 as per line 1.3 else condition of the algorithm.



Type	count
<image>	2
<title>	1
<number>	1
<brand>	1
<price>	1
<link>	1
<image>	2
<link>	1
<title>	
<form>	
<text>	
<text>	

Figure-51: PatternTable (data table)

PatternTable is a data table that contains a set of Enumeration of key objects for candidate tuples. It needs to list some key objects that can distinguish any data block from others. When the algorithm identifies any content object in ContentObjectArray, it increment the count for all rows of same type object in Pattern Table.

We set a criterion for support in favor of identifying any data block. For example, in case of Product data block, it should identify at least 3/5 listed objects. In case of List data block, count of given pair should be at least 3.

When the iteration scans at cell 19, it hits line 1.2 of the algorithm and checks for minimum support for data block at pattern table. If the minimum support satisfies as per line 1.2.1, the algorithm then forms tuple by changing the notation of respective open and close separator object to tuple notation. Resultant data tuple is shown in figure 50(B). Similarly, snapshot B of figure 50 identifies product tuple as shown in figure 52 below.

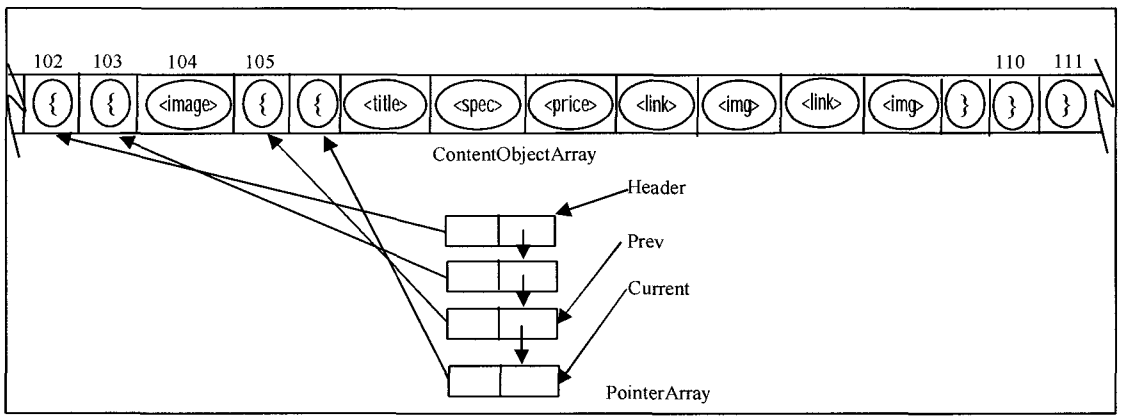


Figure-52: Data block Identification / Tuple formation.

In figure 50A we see three image objects in this data block. Last two encoded <link> and <image> objects are for "Add to Cart" image with link to "shopping cart" page, and "More Info" image with link to "Customer Ranking".

Next step of the algorithm as per line 1.2.2 is to generate Seed NFA by calling the algorithm GenerateSeedNFA() (figure 53). GenerateSeedNFA() automatically generates candidate NFA by second pass iteration through all objects of a identified tuple for

```
Algorithm GenerateSeedNFA (Enum x)

Input:   Enumeration x                          //Pattern Table of specific tuple type x
Output:  Seed NFA of tuple type x

Begin
    1.0  If Seed NFA exist
         1.1  set qc ← q0;
    2.0 else
         2 1  Initialize data structure for NFA, N= (Q, Σ, δ, q0, F);
         2.2  Set Q ← {q0}, δ ← 0, F ← 0;
         2.3  set qc ← q0;
    3.0 For each object 's' in sequence of tuple
         3.1  If ∃ δ(qc , s) = qn or ∃ δ(qc , ε) = qj □ δ(qj , s) = qn in Seed NFA
              3.1.1  set qc ← qn;
         3.2  Else if ∃ δ(qc , s') = qn ; where s' ≠ s          //To create ε transition
              3.2.1 Create new state qa ; a < c
              3 2 2 Create transition δ(qa , ε') = qc ;          // i e , δ ← δ U{((qa , ε'), qc)}
              3.2.3 set qc ← qa , Q ← Q U {qc} ;
              3 2 4 Create transation δ(qc , ε') = qj , qc ← qj ;   here c < j
              3.2.5 Create new state qm and δ(qc , s) = qm ,     // i e , δ ← δ U{((qc , s), qm)}
              3.2.6 set qc ← qm , Q ← Q U {qm} ;
         3.3 else
              3.3.1 Create new state qc+1 and δ(qc , s) = qc+1 ,   // i e , δ ← δ U{((qc , s), qc+1)}
              3.3.2 set qc ← qc+1 , Q ← Q U {qc+1} ;
             Endif
         3.4 If 's' is the last object in tuple
              3.4.1 If Q ∩ qc+1 = 0;
                      Set qc ← F;
              3.4.2 Else
                       Refine Seed NFA to create representation pattern;
                    endif
             endif
       endFor
End
```

Figure- 53: Algorithm GenerateSeedNFA to generate candidate NFA

effective extraction of data from ContentObjectArray[] and wide range of other pages from WWW. It input all data structure of algorithm IdentifyTuple() as global and works with satisfied Enumeration type to create its seed NFA. The algorithm identifies the tuple type from PatternTable and looks for any existing Seed NFA for that tuple type. If not exist, it start creating a new NFA by scanning objects and creating NFA state along with appropriate transition between states as per figure 53. In case of our running example

data tuple shown in figure 47(B), since the existing Seed NFA is null, the algorithm creates the starting state 'q_0' and refer 'q_c'(current state) to 'q_0' as per line 2.0. For <link> object at cell 9 it creates another state 'q_1', refer it as next to the header state 'q_0', store <link> object into 'q_0' and refer 'q_1' as 'q_c' per line 3.3. This ensures a transition from 'q_0' to 'q_1' for <link> object. For second iteration it scans <image> object of cell 9, creates state 'q_2', refer it as next to 'q_1', store object into 'q_1' and refer 'q_2' as 'q_c' per line 3.3. This process continues until the last <image> object of the tuple at cell 17. Since it is the last object, the last state is denoted by 'F' as per line 3.4.1. The next step is the refinement of the generated Seed NFA as per definition of section 3.4.3.

Line 2.0 of our mining algorithm MineContentObjects() shown in figure 48 at page 88 uses function SqueezeTuple() which basically squeezes the tuple length to represent their general pattern. For example our running example nevigation tuple is as follows:

⟨ <link>, <img>, <link>, <img>, <link>, <img>, <link>, <img>,<link>, <img>, ⟩

This length of this tuple is unknown with a set of repeated tags <link> and <img>. These repeated tags follow a general pattern. We can squeeze these tuples with their common pattern of a link tag followed by a title text tag as $\sum_3^n(< link >,< img >)_1$.

```
Algorithm SqeezeTuple(TupleList)

Input: TupleList
Output: Squeezed TupleList

begin
1.0 For each tuple in TupleList
        1.1 Calculate the length of Tuple;
        1.2 If tuple length is more then One
                1.2.1 Set header pointer to first object location;
                repeat
                    If object is not a separator object
                        1.2.1.1 Create an instance of Linked List;
                        1.2.1.2 Put the object in Linked List;
                        1.2.1.3 Store the Linked List at objects original position at TupleList;
                        repeat
                            If current object is "instanceOf" header object
                                1.2.1.3.1 Replace object from TupleList to end of LinkedList;
                                1.2.1.3.2 Increment current pointer;   //Illustration purpose only
                            else
                                1.2.1.3.3 Increment current pointer;   //Illustration purpose only
                        until end of tuple;
                        1.2.1.4 Increment header pointer;
                until end of tuple;
        endif;
endFor;
end;
```

Figure- 54: Algorithm for squeezing object tuples

The need for this squeezing of tuples to their pattern is for the generalization of the same types of data block, so that pattern of any tuple containing any length can be represented in the same category. We use LinkedList data structure to squeeze this tuple without disturbing its data representation order.

Line 4.1 of figure 48 then call CreateDBtable.intertData() algorithm as shown in figure 55 below creates database connection, creates primary and foreign key and checks for tuple type and finally insert data into database table.

```
Algorithm CreateDBTable.insertData()

Input:   ArrayList, String CompanyName
Output:  Populated data table

Begin,
        1.  Register Oracle driver and create Oracle connection.
        2.  PrepareStatement for different Data Tables.
        3.  Check tuple type and company name data coming from and set primary key
        4.  for each object in a tuple
                4.1.  Check object type and retrieve data from object.
                4.2.  SetString to PrepareStatement list.
                4.3.  Insert data into respective data table.
            Endfor
End
```

Figure- 55: Algorithm CreateDBTable.insertData()

Figure 56 shows how we propose to squeeze the List Tuple and a Form Tuple using LinkedList data structure. Here figure 56 (A) is the original tuple and (B) is the squeezed format of the tuple. Figure 56 C is another example of squeezed text tuple.

In our MineContentObjects() algorithm of figure 48, we considered minimum support which we think is important to consider. For example, in a product list page of a

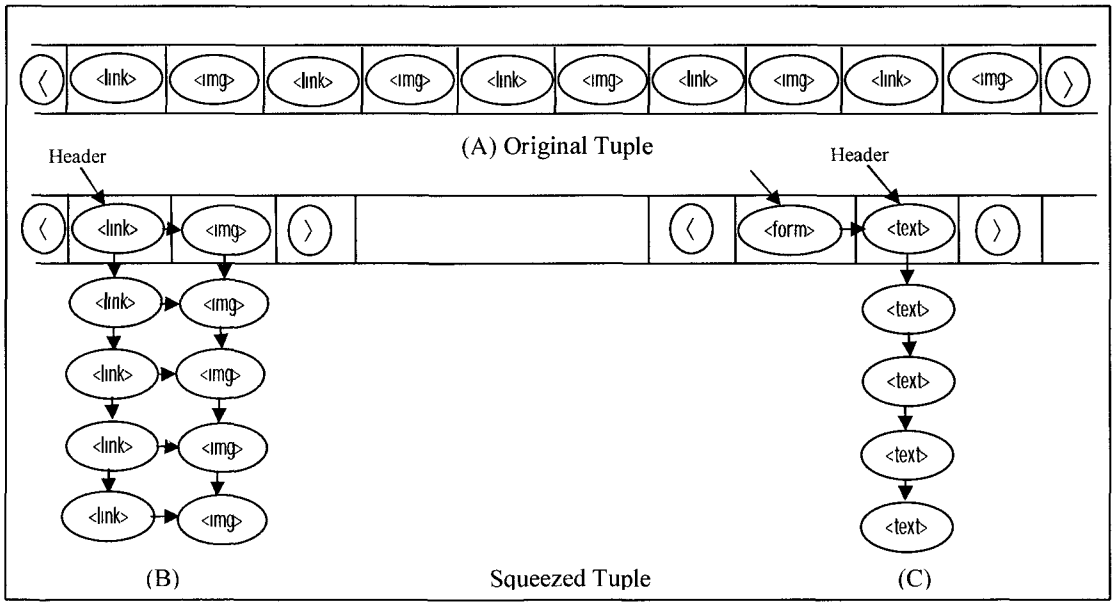


Figure- 56: Example of Squeezing tuples

business to customer web page, major information in body zone is about their products (i.e. Product tuples), hyperlinks to other pages about their services, products (i.e. List tuples), others like advertisements, noises (i.e. Noise tuples) and presentation images (i.e. singleton tuples). Following table 02 gives an overall idea about different category tuples in monitor Product List page of four different web sites:

Type of Tuple	CompUSA	Best Buy	Circuit City	Future Shop	Average	Percentage
List Tuple	21	10	11	13	13.8	37%
Product Tuple	18	7	18	10	13.3	36%
Text Tuple	2	1	3	0	1.5	4%
Form Tuple	0	1	2	0	0.8	2%
Singleton Tuple	0	0	5	0	1.3	3%
Noise Tuple	8	4	11	4	6.8	18%
Total	49	23	50	27	37.3	100%

Table-02: Different tuples in monitor web page of some commercial website

Minimum support is an important measure for identifying the positive web pages. Some other pages may also contain few product data block but mostly emphasized in other information, those pages are not truly the product list page. We don't want to extract information from those pages. In case of those pages, the percentage of Product Tuple will be inconsistent in numbers with an average product list page. It is very important for the data consistency before entering into the data base.

3.6 Warehouse and Mining for Integration

Our target is comparative mining or web content data integration. In section 3.1-3.5 we discussed how we will be able to mine the web content objects, extract and store the related data for integration. This first level mining is sufficient to integrate the related

information from positive pages of different websites containing similar data information as shown in figure 06 for computer monitor.

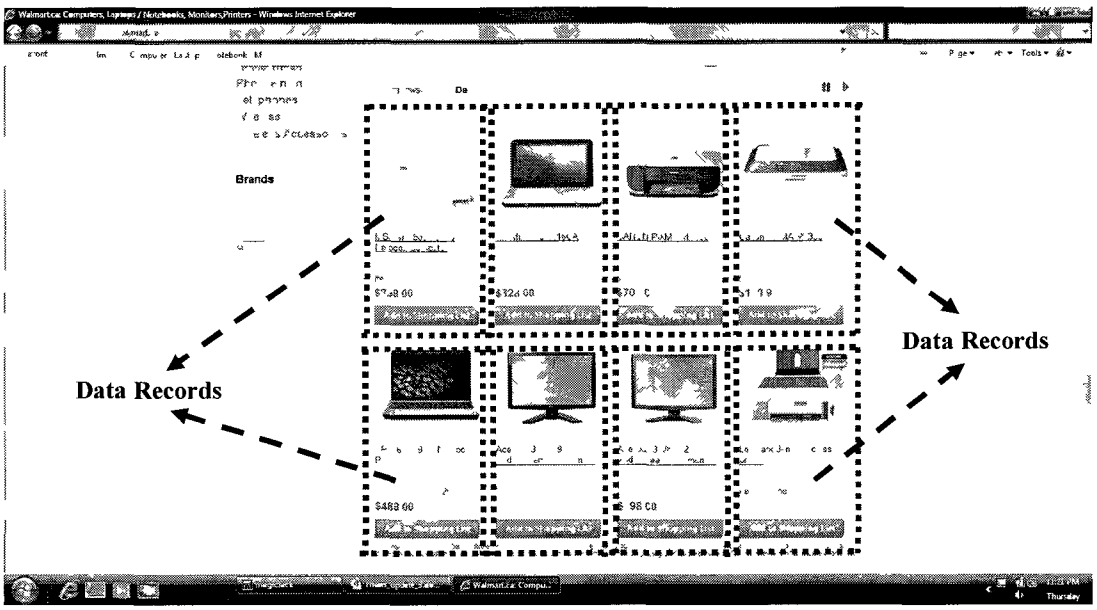


Figure-06: Data blocks

In case of personalization of web content if we want comparative price of monitors, this first level mining will give the comparative information from different pages. But a single product list page may contain information about different products including the monitor as shown in figure 55. So, our database will be a combination of heterogeneous products. This implies the necessity for warehousing. For knowledge discovery from extracted web content data, a suitable database, data warehouse and second level mining is obvious. This section is similar to traditional data mining, which is beyond the scope of this thesis work.

4. Evaluation of WebOMiner System

We are done with basic implementation phase of our algorithms and working to give our algorithms in a scalable, robust and generalized shape. Since our system is a very first effort for mining web contents using object-oriented approach, a valid comparison in performance with other extraction and mining techniques do not exist.

More improvement is required in algorithms of our architecture to make it robust enough to handle vast complexity of corporate commercial websites. Our crawler module needs to create the functionality for automatic selection of the targeted documents from the web, Cleaner module needs to handle long tag attributes as described in section 3.3 (04) to make it robust to handle all kinds of complex webpage to make web documents cleaned and well-formed to create DOM tree. All these are pre-processing work for our thesis problem. More details of the limitations of our system are discussed in Appendix-A. At this point, we therefore, created simplified mirror of six popular commercial websites to test and experiment of our extraction and mining algorithms.

4.1 Strength of WebOMiner

Our WebOMiner system for web content mining using object-oriented model is a novel approach for extraction and mining of web contents. Earlier language based systems are outperformed by semi-supervised and unsupervised wrapper induction and wrapper generation systems. Popular and mile-stone semi-supervised system IEPAD identifies repetitive patterns by building binary suffix tree and use center star method (described in section 2.2.B) for extraction pattern recognition. Unsupervised popular and mile-stone system RoadRunner generates wrapper from a set of webpage by matching and aligning HTML token (tag) and by collapsing the mismatched tokens (figure 04, section 1.2.2). All these popular systems are difficult to compare with our WebOMiner system because of variance in extraction process.

Another mile-stone wrapper generation system DEPTA (described in section 2.3.2) builds DOM Tree to analyze web document and uses single web page for wrapper generation like our WebOMiner system. In compared to IEPED and RoadRunner, DEPTA system is closer in similarity with our WebOMiner system. We therefore compared our system with DEPTA. The comparative analysis is given below:

1. DEPTA does not analyze and correct the HTML code for DOM tree creation. The authors left the job for future work. Creation of DOM tree is not possible unless the entire web document is in local machine and prior correction of all missing and ill-formatted tags accordingly. Our WebOMiner system's crawler module automatically dumps all HTML comments embedded into the HTML documents. For example, in figure 56 (A) below line 1 contains comment which is cleaned in line 1 of figure 56(B).

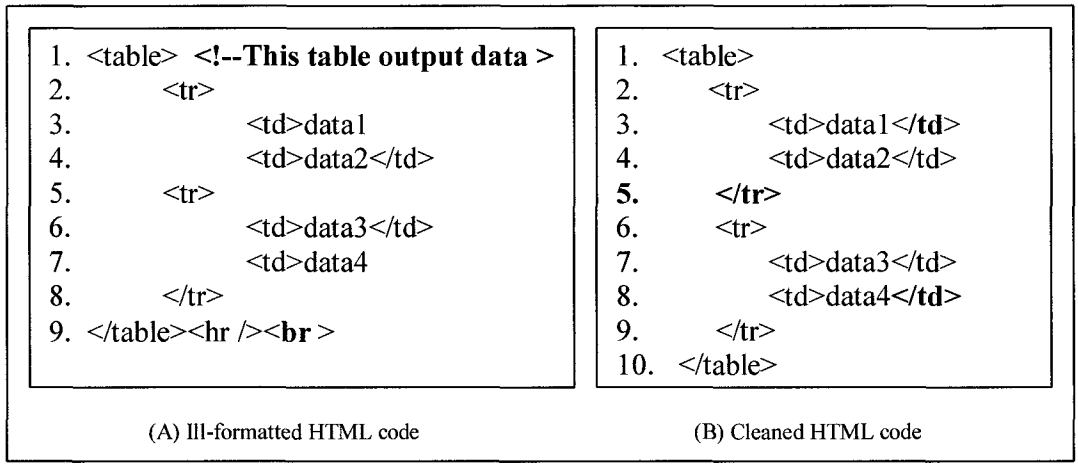


Figure -56: Illustration of HTML code cleaning

The cleaner module of our system has two-fold functionality. Firstly, it automatically analyzes the HTML documents for missing tags and automatically inserts missing tags at appropriate location. For example, figure 56(A) above has two missing end <td> tags at line 3, and line 7, one missing <tr> end-tag at line 5 which are corrected in figure 56(B). In case of inline tags, as shown in line 9 (e.g., <hr/> and
) the cleaner first correct the inline tag
 to
 and then it identifies <hr /> and
 as inline tag and removes them as shown in figure 56(B).

2. DEPTA uses web browser to render web page manually to get visual information, which is then utilized to clean tags and to construct a DOM tree. Requirement of manual rendering is contrary with automatic extraction. Our system is not dependent on manual rendering by web browser. Given a URL string, our system automatically extracts the web document from the WWW, analyze and correct the HTML code automatically to create DOM tree.

3. Formatting tags destroy the structural relationship of the textual contents in DOM Tree. DEPTA uses manual web browser rendering to get visual information of page tag structure to improve the accuracy of data record. Before creating DOM tree, they observe the web document and infer the structural relationship among tags and remove necessary formatting tags in HTML textual contents manually using visual information from browser rendering. For example, the formatting tags of figure 21(C) within the textual content fragments the single text content object into four separate objects in two different levels in DOM tree that destroy the structural relationship of the textual contents as shown in figure 21(C) below:

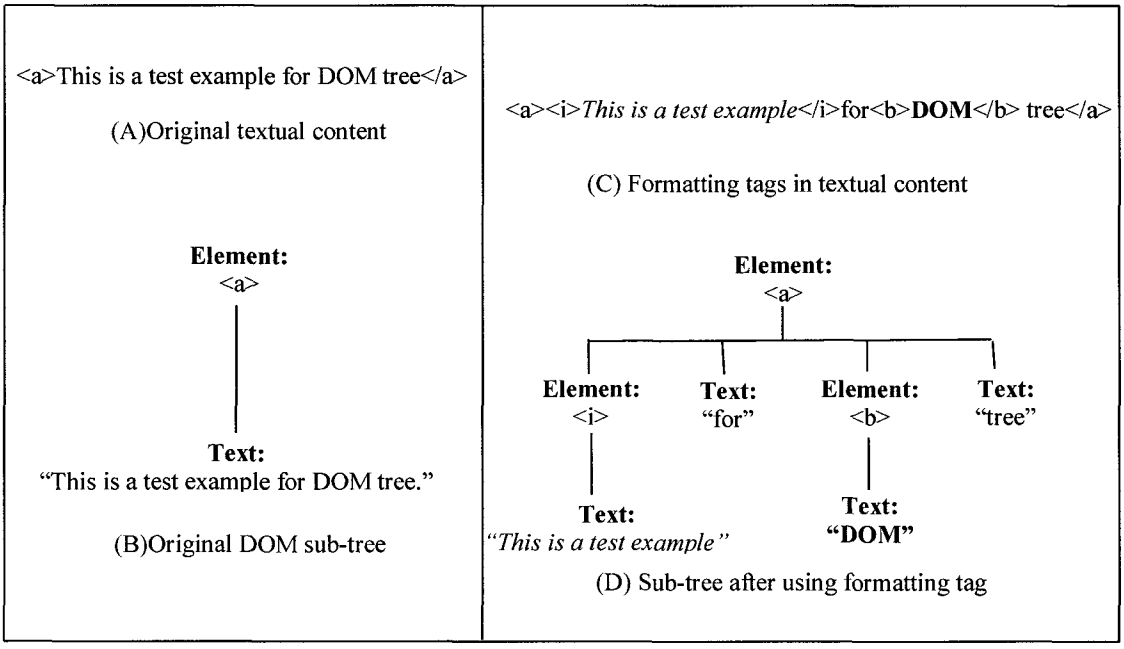


Figure-21: Formatting tags within textual fragment

This is a problem in web content extraction. Figure 21(A) is the textual content when no formatting tag is used and the resultant DOM sub-tree is shown in figure 21(B) that ensures structural relationship of the content. But if the formatting tag <i> and <b> are used within the textual content as shown in figure 21(C), the resultant DOM tree as shown in figure 21(D) destroys the structural relationship of the textual content. DEPTA can not handle the situation automatically and relies on correction by manual observation from web browser rendering.

In our WebOMiner system we defined the formatting tags in filter module to clean-up these formatting tags automatically before building DOM tree without using browser interaction.

4. DEPTA identifies data records using Tree-Distance measure by visual clues (i.e., the physical location of the information on the computer screen by using web browser). Each HTML element in web browser is rendered as a rectangle as shown in figure 18, and each HTML element corresponds to the node in DOM tree. In this approach, four boundaries of the targeted rectangle of web page are located using x-coordinate and y-coordinate first by calling any rendering engine of a web browser. It then follows the sequence of open tags and checks for containment to build tree. Containment check means checking if one rectangle is contained in another. Boundary conditions are then defined (shown in figure 18) for each data records to create tree for each data records. This system is not automated and depends on the manual analysis and use of browser rendering engine.

In our WebOMiner system, we used the observations discussed in section 3.4.1. for data record identification. We observed that all objects of a rata record are contiguous in a DOM sub-tree and each data records are disjoint with other data records. Therefore, there should be a single parent node that represents the sub-tree of an entire data record in DOM tree. Our system identify this parent node for each data records and uses separator object to ensure the integrity relationship in related objects of a data record (discussed in section 3.4.2 and 3.4.3). This system is automatic and not dependent on browser rendering engine.

5. DEPTA aims to extract and mine only the targeted facts from the web page. For example, in case of our running example (figure 06, page 14), data regions are shown in dotted boxes and in data region-1 we have shown the data blocks in blue dotted box. DEPTA only extract information from data region-1. Their tree-matching and tree-alignment technique can handle only one "seed-tree" that produces from multiple data records from data region-1. For each data records in data region-1, they need to execute tree-matching algorithm for each DOM sub-tree for data records and then use tree-

alignment algorithm to create a representative "seed-tree". Since data region-1 consists of six data records, their "seed-tree" is constructed by executing tree matching and tree alignment algorithm for at least 6 times. The result is a single excel data table containing six rows representing six data records of the region-1 of our running example and the columns are created by extracted content data according to generated "seed-tree" tag structure. DEPTA is not able to extract information from other regions of the web page. For example to extract information from data region-2, it needs to generate another "seed-tree", which is not possible as DEPTA can handle only one "seed-tree" for extraction.

On the other hand, our WebOMiner system is most comprehensive in extraction of web content. Our system does not strictly rely on the extraction of only one kind of data from the web page. WebOMiner extracts all kinds of data from all regions of body zone of a web page. Unlike DEPTA and all other existing systems WebOMiner extract data from all regions from bodyzone of web document including product data, navigation data, advertisement, etc. Our system generate NFA wrapper for each type of data record at their first occurrence and then use that wrapper to extract information from subsequent occurrence of data records and at the same time refine the NFA wrapper to give unified form. This NFA wrapper is then ready to create grammar for extraction from subsequent pages.

6. DEPTA is only able to extract textual contents from the web. It is unable to extract the image or any other form of multimedia contents from the web page. This is because of DEPTA does not analyze the tag attributes; it only extracts the tag encoded text from the DOM tree. For example, web document images are image-file those are referred into the <image> tag attribute itself. By analyzing the <image> tag attribute "alt" we can identify the image and "href" refers to the physical location of image file from which it needs to be extracted.

Our WebOMiner system is able to extract heterogeneous web content data because our system analyzes the tag attributes from the DOM tree during the traversal. So, it can effectively identify and extract the images from the data block as discussed in section 3.1.5.

7. DEPTA web content extraction is based on web page tag structure. It evaluates the tag similarity to extract contents without knowledge of content itself. DEPTA analyze only the HTML tag DOM tree (e.g., similar to HTML DOM tree, but only tags are considered) for comparison of adjacent substring. In case of any similar data embedded in intertwined tag, DEPTA wrongfully identify them. For example,

```
<ul>
        <li> Sony </li>
        <span> 17" </span>
        <li> LCD Monitor </li>
        <p> $199.99</p>
</ul>
```

```
<ul>
        <li> 17" </li>
        <span> Sony </span>
        <li> LCD Monitor </li>
        <p> $199.99</p>
</ul>
```

Here in both cases HTML tag alignment is similar but data record '**Sony**' and '**17"** ' are intertwined in <li> and tag. As long as tag alignment is same, DEPTA extracts contents and stores records into respective row in excel sheet. So, data record '**Sony**' and '**17"** ' will be wrongfully stored into wrong column.

But our WebOMiner system is not dependent on HTML tag structure and its alignment. WebOMiner system identifies the data type while extracting and create respective object. So, it can store data records into database robustly.

8. DEPTA generates excel table for extracted web content data. An Excel data table is a data grid, it can not be considered as functional data base. It does not have any identification for each data column. It is because of DEPTA extracts web content data only based on tags and store similar tag encoded contents into same excel column. Manual labeling and transformation is required to store those data into data table to create fully functional database.

On the other hand, our WebOMiner system uses fully functional relational database for storing data records. It identifies content type during the extraction process and create respective object to hold the content and other related information into the object. Our system therefore has prior knowledge in data record type that infers to specific data table and data table attributes are obtained from object class type. This makes our system possible to store web contents into relational database directly.

4.2 Empirical Evaluations

As discussed in the beginning of section 4 of this thesis, we have created simplified mirror of six popular web sites (e.g., futureshop.ca, compUSA.com, bestbuy.ca, walmart.ca, shopping.com, dell.com as of July 10, 2010) for empirical evaluation of our system using different page structures. Our system is implemented in Java programming language. We then run our system in 32-bit Windows Vista Home Premium operating system at Intel Due Core 2.26 GHz, 3.00 GB RAM Sony machine for each of these mirror web sites for empirical evaluation of our WebOMiner system. We use the standard precision and recall measures to evaluate the results of our system. Precision is measured as average in percentage for the number of correct data retrieved divided by the total number of data retrieved by the system. Recall is measured as average in percentage for the total number of correct data retrieved divided by the total number of existing data in the web document. The results of the retrieval by our WebOMiner system is tabulated in table 02 below:

Website	Data records					Data record Extraction		
	Product	List	Noise	Text	Total	Correct	Wrong	Missing
www.futureshop.ca	10	13	4	0	27	27	0	0
www.compUSA.com	18	21	8	2	49	47	0	2
www.bestbuy.ca	7	10	4	1	22	21	0	1
www.walmart.ca	2	4	2	-	8	8	0	0
www.shopping.com	40	4	4	-	48	47	0	1
www.dell.com	14	13	4	-	31	28	0	3
Recall						96.22%		
Precision						100%		

Table 03: Experimental result showing extraction of data records from web pages.

4.3 Experimental Results

The purpose of our experiment is to measure the performance of WebOMiner system for data record extraction. Table 03 shows small scale experimental results as performance measure for our WebOMiner system. We have taken one page per web site for experiment and the numbers in "Data Record" column shows different types of data records in those pages. The Total column shown total number of data records for each

pages. For those pages WebOMiner system is able to identify data records correctly. No wrong data record identification is observed and it makes sense because our system is not based on the prediction. It missed seven out of total 185 data records in all six web pages from different websites.

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

5. Conclusion & Future Work

This thesis includes lot of pre-processing work to prepare data for mining that are not addressed by Annoni and Ezeife (2009). We developed an architecture (we call it WebOMiner) for extraction and mining of web contents using object-oriented model. Our architecture has 4-modules, crawler module, cleaner module, extractor module and the miner module. We developed algorithms for crawler module, modified freeware software "tagsoup" for cleaner module, modified and enhanced algorithms for extractor module initially developed by Annoni and Ezeife (2009) and developed algorithm for miner module. We introduced an approach of generating and using automata for mining web content objects. In this thesis, we used data block and data region concept to ensure consistency between related data, we relate HTML tag attribute information with its content to identify contents. We define object class hierarchy according to our problem domain and address schema matching problem to unify similar contents from different web sites. We also prevent noisy contents entering into database table. Our miner algorithm is based on Automata patterns that have two fold uses: data extraction and automatic database schema generation. We then checks minimum support to ensure data consistency before entering into database. Currently we are working on the implementation of cosign similarity algorithm for automatic classification of tuple from the ContentObjectArray which will eliminate the requirement for the definition of PattaernTable and algorithm SqueezeTuple algorithm.

5.1 Future work

Since this is a very first effort to mine web content data using object-oriented approach, we feel there is plenty of room for improvement and to open new thread. In our WebOMiner architecture further improvement of our algorithms are required to make the system robust and scalable. Our crawler module algorithms require further improvement for automatic identification of positive web pages and functionality to exclude negative web pages from the www. Cleaner module needs the functionality to handle long tag attributes in effective way. There is plenty of scope to improve the extractor module for cleaning of unwanted noisy contents before creating expensive objects. In miner module we introduced automata pattern to mine related contents from specific domain context. Further experiment is required to mine contents from other domain context (such as unstructured content data) and domain independent mining. Automatic database schema generation from the automata pattern is an important task to develop for storing web contents in relational database. Use of Automata for pattern recognition and generation of regular expression from repeated pattern of web content is a new approach in data mining. We observed that generated pattern from B2C websites may not be complete for all possible conditions; further improvement to develop a unified pattern need further experiment. Moreover Annoni and Ezeife's (2009) anticipate use of presentation objects along with content objects for mining web contents is still a research issue.

References

Ai, D., Zhang, Y., Zuo, H., & Wang, Q. (2006). Web Content Mining for Market Intelligence Acquiring from B2C Websites. *Springer-Verlag Berlin Heidelberg,WISE Workshops, LNCS 4256, 159 – 170.*

Annoni, E., & Ezeife, C. I. (2009). Modeling Web Documents as Objects for Automatic Web Content Extraction, *In proceeding of ACM / LNCS sponsored 11th international conference on Enterprise Information Systems (ICEIS 09)* page 91-100, May 6-10, 2009

Arocena, G.O. and Mendelzon, A.O. (1998). WebOQL: Restructuring Documents, Databases, and Webs. *In Proceeding of the 14th IEEE International Conference on Data Engineering.* (ICDE), 24-33, 1998.

Arumugam, S. (2006). Classification Techniques for Categorization of Hypertext Documents. *Lecture Notes on Computer Science, Springer Science + Business Media, LLC.*

Baumgartner, R., Flesca, S., and Gottlob, G. (2001). Visual Web Information Extraction with Lixto. *In proceeding of 27th international conference on Very Large Data Bases*. 2001, 119-128.

Bhowmick, S. S., Madria, S. K., Ng, W. K., & Lim, E. P. (1999). Web Warehousing: Design and Issues. *Lecture Notes of Computer Science, Springer-Verlag Berlin 1552, 93– 105.*

Borges, J., & Levene, M. (1999). Data mining of user navigation patterns. *In Proceedings of the KDD Workshop on Web Mining,* San Diego, California, 31–36.

Bornhövd, C., & Buchmann, A. P. (1999). A Prototype for Metadata-based Integration of Internet Sources. *In Proceeding of CAiSE'99*, Heidelberg, Germany, June.

Buchner, A. G., & Mulvenna, M. D. (1998). Discovering Internet Marketing Intelligence through Online Analytical Web Usage Mining, *SIGMOD Record,* Vol.27, No.4, New York, NY, USA, 1998. Pages 54-61.

Ceci, M., & Malerba, D. (2007). Classifying web documents in a hierarchy of categories: a comprehensive study. *Lecture Notes on Computer Science, Springer Science + Business Media, LLC.*

Chakrabarti, S. (2003). Mining the Web: Discovering Knowledge from Hypertext Data. *USA, Morgan Kaufmann Publishers.*

Chang, C., & Lui, S. L. (2001). *IEPAD:* Information extraction based on pattern discovery. *In proceeding of the 10th international conference on WWW* Hong Kong, page: 681-688.

Chaudhuri, S., Ganjam, K., Ganti, V., & Motwani, R. (2003). Robust and efficient fuzzy match for online data cleaning. *In proceeding of the ACM SIGMOD International Conference on Management of Data*, San Diego, CA, ACM Press, 313-324.

Chawathe, S., Garcia-Molina, H., & Hammer, J. (1994). The TSIMMIS project: Integration of heterogeneous information sources. *In Proceeding of IPSI'94, Japan, March.*

Chow, C. K. (1957). An optimum character recognition system using decision functions. *IRE Transactions 247–254.*

Chriisment, C., Dousset, B., Karouach, S., & Mothe, J. (2004). Information mining: extracting, exploring and visualising geo-referenced information. *SIGIR-04 Workshop on Geograpghic information retrieval.*

Cimiano, P., Handschuh, S., & Staab, S. (2004). Towards the self-annotating web. *International WWW conference, NY, USA, 462-471.*

Crescenzi, V., Mecca, G., Merialdo. P. (2001). RoadRunner: Towards Automatic Data Extraction from Large Web Sites. *Proc. of VLDB 2001,* Rome, September 2001, pp. 109-118.

Darmont, J., Boussaid, O., & Bentayeb, F.(2002). Warehousing Web Data. *Conference on Information Integration and Web-based ..., 2002*

Dung, X. T., Rahayu, W., & Taniar, D. (2007). A high performance integrated web data warehousing. *Cluster Computing,* Volume-10 Issue-1, Kluwer Academic Publishers, March 2007.

Etzioni, O. (1996). The World Wide Web: Quagmire or gold mine. *Communications of the ACM* 39(11): page 65-68, 1996

Ezeife, C. I., Saeed, K., & Zhang, D. (2009). Mining Very Long Sequences in Large Databases with PLWAPLong. *In proceedings of the 13th ACM sponsored International Database Engineering & Applications Symposium,* Cetraro, Calabria, Italy, 16-18 September 2009 (IDEAS 09).

Grumbach, S. & Mecca, G. (1999). In search of the lost schema, ICDT-99, *Lecture Notes in Computer Science,* Springer, Vol. 1540, page 314-331.

Gupta, S., Kaiser, G., & Stolvo, S. (2005). Extracting context to improve accuracy for HTML content extraction. *In Proceeding of International WWW Conference, Japan, May 10-14.*

He, B., & Chang, K. (2003). Statistical schema matching across web query interfaces. *In: SIGMOD '03.*

Jupiter Media Corporation (2007). XML Parsers: DOM and SAX Put to the Test. *Retrieved from: http://www.devx.com/xml/Article/16922/1954*

Kaufman, L., Rousseeuw, P.(1990). Finding Groups in Data: An Introduction to Cluster Analysis. *John Wiley & Sons, 1990.*

Kosala, R., & Blockeel, H. (2000). Web mining research: a survey. *SIGKDD Explor. Newsletter. 2(1): 1-15.*

Laender, A.H.F., Neto, B. R., and A.S. da Silva, A. S. (2002). Debye-Date Extraction by Example. Data and Knowledge Engineering, 40(2):121-154.

Levering, R., Cutler, M. (2006). The portrait of a common html web page. *In DocEng'06, ACM symposium on Document engineering,* NY, USA. ACM, 198-204.

Li, J. and Ezeife, C.I. (2006). Cleaning Web Pages for Effective Web Content Mining. *In proceedings of the 17th International Conference on Databases and Expert Systems Applications,* DEXA 2006, Krakow, Polland, Sept 4-8, published in LNCS, pp. 560-571, Springer Verlag.

Liu, B. (2007). Web Data Mining; exploring hypertext, content and usage. *Lecture Notes on Computer Science, Springer Science + Business Media, LLC.*

Liu, B., & Chen-Chung-Chang, K. (2004). Editorial: special issue on web content mining. *SIGKDD Exeplor. Wewsl., 6(2): 1-4*

McCallum, A., & Nigam, K. (1998). A comparison of event models for Naive Bayes text classification. *In AAAI-98 Workshop on Learning for Text Categorization.* Menlo Park California: AAAI, 41-48.

McLachlan, G. J. (1992). Discriminant Analysis and Statistical Pattern Recognition. *Wiley, New York.*

Muslea, I., Minton, S., & Knoblock, C. (1999). A hierarchical approach to wrapper induction. *In AGENTS'99: Proceedings of the third annual conference on Autonomous Agents, ,* New York, USA. ACM, 190-197.

Page, L., Brin, S., Motwani, R., & Winograd, T. (1998). The Page Rank Citation Ranking; Bringing order to the Web. *Technical Report, Compute Science Department,* Stanford University, 1998.

Pant, G., & Srinivasan, P. (2005). Learning to Crawl: Comparing Classification Schemes. *ACM Transactions on Information Systems,* Vol. 23, (4), October 2005, 430–462

Petrushin, Valery A., & Khan, L. (2007). Multimedia Data Mining and Knowledge Discovery. (Eds.) 2007, *XXVI, 526 p. 213 illus., Hardcover ISBN: 978-1-84628-436-6.*

Rahm, E. and Bernstein, P.A. (2001). A survey of approaches to automatic schema matching. *The VLDB Journal 10: 334-350 (2001), Springer.*

Raposo, J., Pan, A., Alvarez, M., Hidalgo, J., A. Vina, A. (2002). The Wargo System: Semi-Automatic Wrapper Generation in Presence of Complex Data Access Modes. *In proceeding of 13th international workshop on Database and Expert Systems Applications,* 2002, 313-320.

Song, R., Liu, H., Wen, J-R., Ma, W-Y. (2004). Learning block importance models for web pages. *In KDD '03,* pages 203-211, Ney York, NY, USA, ACM.

Stonebraker, M., & Hellerstein, J. M. (2005). Content Integration for E-Business. *ACM SIGMOD* 2001 May 21-24, Santa Barbara, California USA. Copyright 2001 ACM 1-58113-332-4/01/05.

Tan, X., Yen, D. C., & Fang, X. (2003). Web warehousing: Web technology meets data warehousing. *Elsevier Science Ltd, OH, USA, page 131-148.*

Wu, W., Yu, C., Doan, A., & Meng, W. (2004). An interactive clustering-based approach to integrating source query interfaces on the deep web. *In proceedings of the ACM SIGMOD international conference on Management of data,* 2004, 95 – 106

Yang, W. (1991). Identifying syntactic differences between two programs. *Softw. Pract. Exper.*, 21(7):739–755, 1991.

Yi, L., Liu, B., & Li, X. (2003). Eliminating noisy information in Web pages for data mining. *SIGKDD-2003,* August 24-27, 296-305.

Yu, S., Cai, D., Wen, J-R., Ma, W-Y. (2003). Improving pseudo-relevance feedback in web information retrieval using web page segmentation. *In WWW'03,* pages 11-18, Ney York, NY, USA, ACM.

Zhai, Y., & Liu, B. (2006). NET – A System for Extracting Web Data from Flat and Nested Data Records. *Lecture notes in computer science, Springer,* Vol 3806, 487-495.

Zhai, Y. and Liu, B. (2007). Extracting Web Data Using Instance-Based Learning. *Lecture notes in computer science, Springer, vol. 10(2) pages 113-132*

Zhao, H., Meng, W., Wu, Z., Raghavan, V., & Yu, C. (2005). Fully automated wrapper generation for search engines. *In WWW'05: Proceeding of the 14th international conference on WWW,* NY, USA, ACM, 66-75.

Zhao, L. and Ng, W. K. (2004). WICCAP: from semi-structured data to structured data. *In proceeding of 11th IEEE international conference and workshop on Engineering and Computer based Systems.* Brno, Czech Republic, May 24-27: 86-93

Zhu, Y. (1999). A Framework for Warehousing the Web Contents. Lecture Notes in Computer Science, Springer Berlin / Heidelberg, Volume 1749/1999, 773-*799*

Zhu, Y., Bornh¨ovd, C., & Buchmann, A. P. (2001). Data Transformation for Warehousing Web Data. *In Proceedings of the Third Int'l Workshop Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS'01),* 1530-1354/01, 2001

A P P E N D I X – A

WebOMiner System Manual

Developed by Titas Mutsuddy

Table of Contents

1.0 System Architecture

Our WebOMiner system architecture consists of four modules: Crawler module, Cleaner module, Extractor module and Miner module. The overall architecture of the WebOMiner system is shown in figure 1 below:

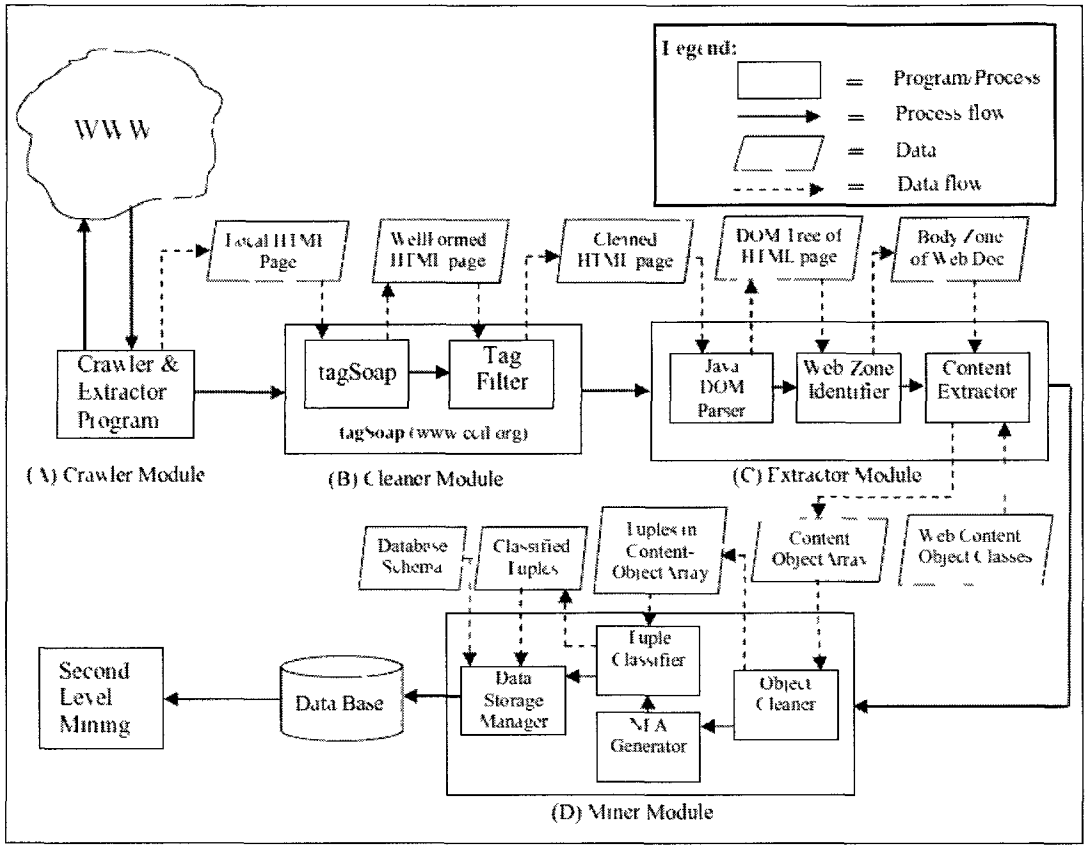


Figure -1: Main program of WebOMiner system

Here, the crawler module crawls through the WWW to find targeted web page, streams entire web document including tags, texts and image contents and it then creates a mirror of original web document in the local computer. Our crawler module also discards all comments from the HTML document. That means it have the functionality to exclude all comments from the web documents. Cleaner module first looks for ill-formatted HTML tags and missing end-tags and insert missing tags at appropriate location. It then filters inline tags to conform structural relationship of text contents. The resultant of the cleaner module is a web page in local directory which is well-formed and cleaned. Extractor

module take this web page as input and extracts the contents from the body zone of the web page, creates objects and store content objects into ContentObjectArray and Miner module mines extracted contents from the ContentObjectArray and store contents into database WebOMiner program entry points and process flow sequence of algorithms are discussed in section 3.5 of the thesis.

2.0 User Interface

At this point WebOMiner system does not contain any Graphical User Interface (GUI) for its user. The system is possible to run from command line or from any integrated development environment (IDE) which is compatible with Java development environment The system is developed using NetBeans IDE 6.7.1., compiled and debugged from the entry point "Main" program. During testing, the program takes one URL at a time at the main program as shown in figure 2 below, it is possible to put a set of URL string in an input file and run those URL strings as input for the Main program one after another sequentially by calling that input file.

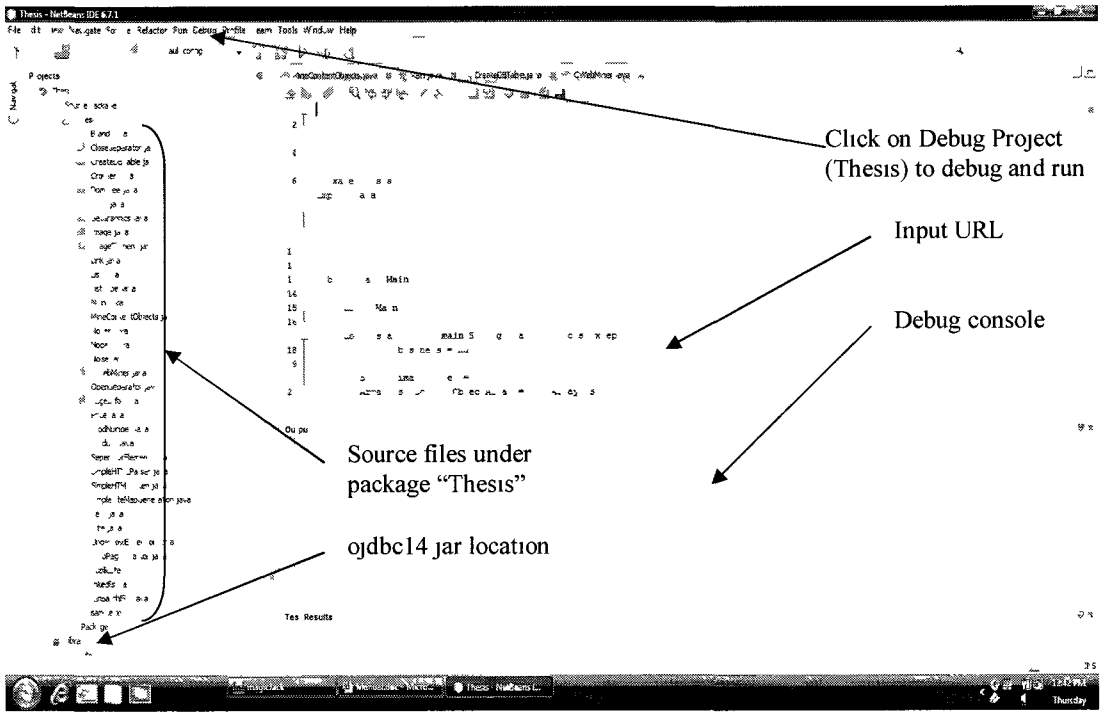


Figure -2 Main program of WebOMiner system

2.1 How to debug, run the program 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 SDK 1.5 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 IDE we need to down load and install NetBeans IDE as described in section 5.1.
4. Import "Thesis" project in NetBeans IDE by using following steps:

   File → Import Project → Browse → select "Thesis" from CD → ok. The "Thesis" project will be imported to NetBeans IDE.
5. Install Oracle 10g XE or Oracle10g or later version as described in section 5.3.
6. Install JDeveloper IDE as described in section 5.4.
7. Create database tables using JDeveloper IDE as per database schema described in section 6.0.
8. Place the "ojdbc14.jar" driver file in "libraries" folder under "Thesis" package as shown figure 2.
9. Prior to any experiment or enhance / modification of the system. First try to compile, run and get the results from sample mirror webpages given under "NetBeansProjects/ Thesis". All respective photos are inside the "NetBeansProjects/Thesis/photo" folder.
10. Rent a domain name and upload the sample webpages and respective images into the domain. Check from browser if the webpages are running appropriately from the rented domain.
12. Note that the domain should be forced advertisement free. Forced advertisements destroy the HTML tag structure and destroy DOM building. Using the web browser, right click and click on "view source" to check for any forced advertisement embedded into the webpage by the domain name providers.
12. Write down/copy the web address from the address bar of the web browser.

13. Open the main program of the "Thesis" package and paste or write down the url address into Input URL string as shown in figure 2.
14. Go to Debug button in Manu bar of the NetBeans IDE, click it and select and click Debug (Thesis) button as shown in figure 2. If the URL is linked properly, we will see the list of image downloading sequence in the output console shown in figure 2.
15. To look at the populated database tables,

    Go to start Manu →All Programs → Oracle Database 10g → Database Homepage → Enter username and password in Oracle database login.
16. After login to Oracle database, go to SQL Manu → select SQL Command→Enter command.
17. From the new window enter SQL command and click on "Run" button. The results will be shown in "Results" window below as shown in section 6.0.

3.0 Operating System

System is developed in 32-bit Windows Vista Home Premium operating system at Intel Due Core 2.26 GHz, 3.00 GB RAM Sony machine. This system is portable in University of Windsor CDF Solaris Operating System on Unix environment. In that case, only minor changes from Windows based syntax to Unix compatible syntax transformation is needed to compile, execute and run the program.

4.0 Programming Environment

The application is programmed and tested in Java 2 Platform Standard Edition version 1.6.0_16, which is installed in Windows Vista Home Premium edition. 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 the system

5.1. Installation of NetBeans IDE:

5.1.1. Installing the Software Bundle on Microsoft Windows:

To install the software, we must need to have administrator privileges on our system. The installer places the Java Runtime Environment (JRE) software in `%Program Files%\Java\jre6`, regardless of the specified JDK install location.

Note: This installer does not displace the system version of the Java platform that is supplied by the Windows operating system.

Both the JDK and IDE have been tested on the following Windows platforms:

- Microsoft Windows XP Professional (SP3), Windows 7 Professional/Windows Vista Home/Professional

Before Installation:

1. We need to verify our system to meet or exceed the following minimum hardware requirements:
    - 800MHz Intel Pentium III or equivalent
    - 512 MB of RAM.
    - 750 MB of free space

    Note: The installer uses the `%USERPROFILE%\Local Settings\Temp` directory to store temporary files.

2. First need to verify that we have administrator privileges on our system.
3. Then download the `jdk-6u21-nb-6_9_1-windows-ml.exe` installer file.

Installing the Software:

1. We need to double-click the installer `jdk-6u21-nb-6_9_1-windows-ml.exe` file to run the installer.

2. At the JDK Installation page specify which directory to install the JDK into and click Next.
3. At the NetBeans IDE Installation page, we need to do the following:
    1. Specify the directory for the NetBeans IDE installation.
    2. Accept the default JDK installation to use with the IDE or specify another JDK location.
4. Review the Installation Summary page to ensure the software installation locations are correct.
5. Click Install to begin the installation. When the installation is complete, we can view the log file, which resides in the following directory: `%USERPROFILE%\.nbi\log`.

5.1.2. Installing the Software Bundle on Solaris OS (SPARC and x86 Platform Editions) and Linux Platforms

We can install the JDK software and NetBeans IDE in directories of our choice. We do not need to be a root user to use this installer unless we choose to install this bundle in a system-wide location.

Note: This installer does not displace the system version of the Java platform that is supplied by the operating system.

Both the JDK and NetBeans IDE 6.9.1 have been tested on the following:

- Solaris 10 OS (x86 and SPARC)
- Ubuntu 9.10

Before we install:

1. If we need to install on Solaris OS, install the required Solaris OS patches before proceeding further. If we install this bundle without having first installed the proper Solaris patches, we may experience unexpected behavior with the installer or the Java platform.

2. Verify that our system meets or exceeds the recommended minimum hardware requirements as follows:
    - Ultra 10 workstation, UltraSparc II 450 MHz, AMD Opteron 1200 Series 1.8 GHz, or Pentium III workstation, 800 MHz
    - 512 MB of RAM
    - 650 MB of free space

    Note: The installer uses the `/tmp` or the `/var/tmp` directory for temporary files.

3. We need to download the installer file and save it on our system.
    - For Solaris OS SPARC Platform Edition, the installer file name is `jdk-6u21-nb-6_9_1-solaris-sparc-ml.sh`
    - For Solaris OS x86 Platform Edition, the installer file name is `jdk-6u21-nb-6_9_1-solaris-x86-ml.sh`
    - For Linux operating system, the installer file name is `jdk-6u21-nb-6_9_1-linux-ml.sh`
4. Navigate to the directory into which we downloaded the installer file and type: `chmod +x <installer-file-name>` to change the installer file's permissions so it can be executed.

Installing the Software:

1. Type the following command from the directory where we placed the installation file:

    ```
    ./<installer-file-name>
    ```

2. At the JDK Installation page, we need to specify the directory where to install the JDK and click Next.
3. At the NetBeans IDE Installation page, we need to do the following:
    1. Specify the directory for the NetBeans IDE installation.
    2. Accept the default JDK installation to use with the IDE or specify another JDK location.
4. Review the Installation Summary page to ensure the software installation locations are correct.

5. Click Install to begin the installation. When the installation is complete, you can view the log file, which resides in the following directory:

   `~/.nbi/log.`

Note: If we choose to install this bundle into a system-wide location such as `/usr/local`, we must first login as root to gain the necessary permissions.

5.2. Load and Run "tagSoup" software:

We have to load entire "tagSoup" software in the package sub-directory under the directory NetBeansProjects. We named out package as "Thesis" and so the "tagSoup" will need to be loaded at following path

"//NetBeansProjects/Thesis/taguoup"

This folder path included an "index.html" file that describes details about tagSoup releases, what it does, source path and how to run as a stand-alone program. The main Jar file "tagsoup-1.2.jar" which is distributable to any other application is located in the following folder:

"//NetBeansProjects/Thesis/taguoup/dist/lib/"

Create and update batch file:

A batchfile named "test.bat" needs to be created for DOS from administrative privilege in Windsows environment. For Unix environment, we need to change the access privilege (e.g., read-write-execute) for the file folder to run the batch file. The details to create and run the batch file are discussed in the following link:

"http://linux.about.com/library/cmd/blcmdl1_batch.htm"

The access privilege for Unix environment is discussed in the following link:

"http://www.zzee.com/solutions/unix-permissions.shtml"

In case of converting DOS batch files from Windows to Unix Shell script is described in details in the following link:

"http://tldp.org/LDP/abs/html/dosbatch.html"

The command string inside the batch file "test.bat" needs to be change automatically from the source program every time before running the batch file. The command script format into the batch file is given below:

"java -jar tagsoup/dist/lib/tagsoup-1.2.jar --html --method=html --files ***filename.html*** *src:* ***filename.html*** *dst:* ***filename_.html****"*

Here, filename is the name of the URL. For example, in case of "futureshop.ca" the source (e.g., src:) filename should be futureshop.html and destination (e.g., dst:) filename will be futureshop_html.

What happens while run the application?

After execution of the crawler module with URL for example "*compUSA.com*", a local html file named "*compUSA.html*" will be created in the following path:

"//NetBeansProjects/Thesis/"

All the images of this web document will be stored in the "photo" sub-directory under the URL source in the following path:

"//NetBeansProjects/Thesis/photo/"

When we run the batch file "test.bat", the "*compUSA.html"* file will be the input file for tagSoup and will generate destination file *"compUSA_.html"*. This "*compUSA_.html"* file will work as input file for extractor module of our system.

5.3. Installation of the Oracle software:

We used lightweight Oracle XE 10g edition for Windows environment as database storage for our system. The software is available to free download from:

" *www.oracle.com"*

To install Oracle XE database 10g edition, download and install Oracle Database 10g Express Edition on Windows environment using the following step:

1. Go to the Oracle Express Edition page.
2. Download the Oracle Database 10g Express Edition (Universal) - Multi-byte Unicode database for all language deployment, with the Database Homepage user interface available in the following languages: Brazilian Portuguese, Chinese

(Simplified and Traditional), English, French, German, Italian, Japanese, Korean and Spanish.

3. Save the download file, OracleXEUniv.exe (216,933,372 bytes), to a temporary directory.

4. Double click on OracleXEUniv.exe. Follow the installation wizard to finish the installation. Remember to change the destination directory, set the SYTEM password and take some notes like:

```
SYSTEM password: "your password"
Destination Folder: C:\local\oraclexe\
Port for 'Oracle Database Listener': 1521
Port for 'Oracle Services for Microsoft Transaction Server': 2030
Port for HTTP Listener: 8080
```

To start Oracle XE server, Click on Start > All Programs > Oracle Database 10g Express Edition > Start Database.

To stop Oracle XE server, Click on Start > All Programs > Oracle Database 10g Express Edition > Stop Database.

Oracle Driver plug-in: We used Oracle driver ojdbc14.jar which need to be plug-in into lib folder of the package folder.

5.4. Installation of JDeveloper IDE:

Oracle JDeveloper IDE is available to free download from www.oracle.com and the installation details are available in the oracle web site. We used this IDE to develop the data tables and database schema for the oracle database.

6.0 Data Base Schema and format

Our intention is to develop database schema automatically from generated NFA. Right now we did not achieve this goal and developed our database schema and tables manually using the Oracle JDeveloper IDE. We used visual observation of incoming data from the web page for schema development. We observed that from our problem domain, maximum six types of data tuples may generate as discussed in section 3.2 of the thesis. These are Product data tuple, List data tuple, Text data tuple, Noise data tuple, Form data

tuple and Singleton data tuple. In case of our example test web pages, only four data tuples (Product, List, Text and Noise) exist and so we only generated four tables, one for each data tuple type. That means the content data from one specific data tuple type will be inserted into a specific data table. For example, all contents from any Product data tuple will be inserted into "PRODUCT" table and so on. Two more data tables are also generated; these are "COMPANY" and "LIST_TYPE". We described below in brief about these database tables:

COMPANY TABLE:

This table keeps track of web site ID from which data are coming and generates ID for each web site to keep track the relationship between different tables by using Primary key and Foreign key. Database schema for COMPANY table is given below:

Company(company_Name string, company_id int),

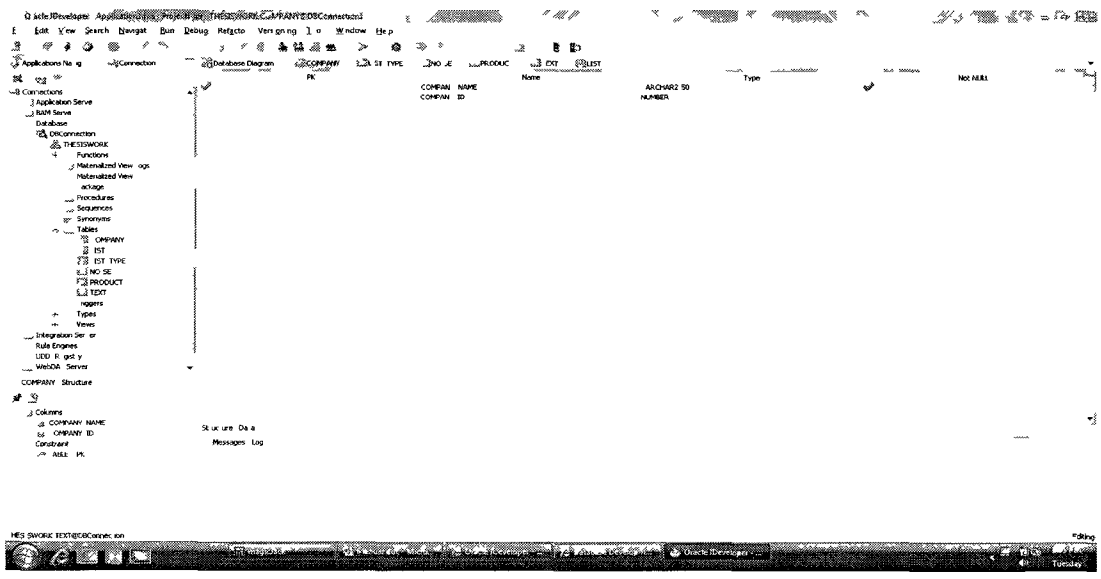


Figure-3: JDeveloper IDE screen shot for COMPANY table

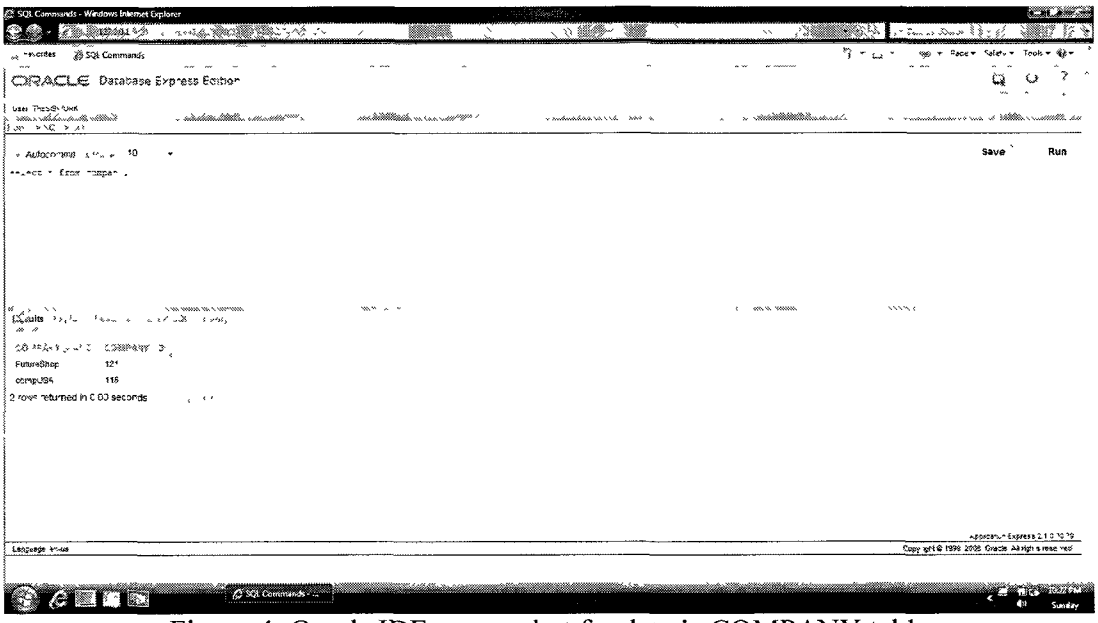


Figure-4: Oracle IDE screen shot for data in COMPANY table

PRODUCT TABLE:

This table holds the contents from the Product data tuples. Database schemas are given as follows:

Product (title:string, image:image-file, prodNum:string, brand:string, price:long);

Product (title:string, image:image-file, prodNum:string, price:long);

Product (title:string, image:image-file, brand:string, prodNum:string, price:long);

Product (title:string, image:image-file, brand:string, price:long);

Product (title:string, image:image-file, price:long);

Product (image:image-file, title:string, prodNum:string, brand:string, price:long);

Product (image:image-file, title:string, prodNum:string, price:long);

Product (image:image-file, title:string, brand:string, prodNum:string, price:long);

Product (image:image-file, title:string, brand:string, price:long);

Product (image:image-file, title:string, price:long);

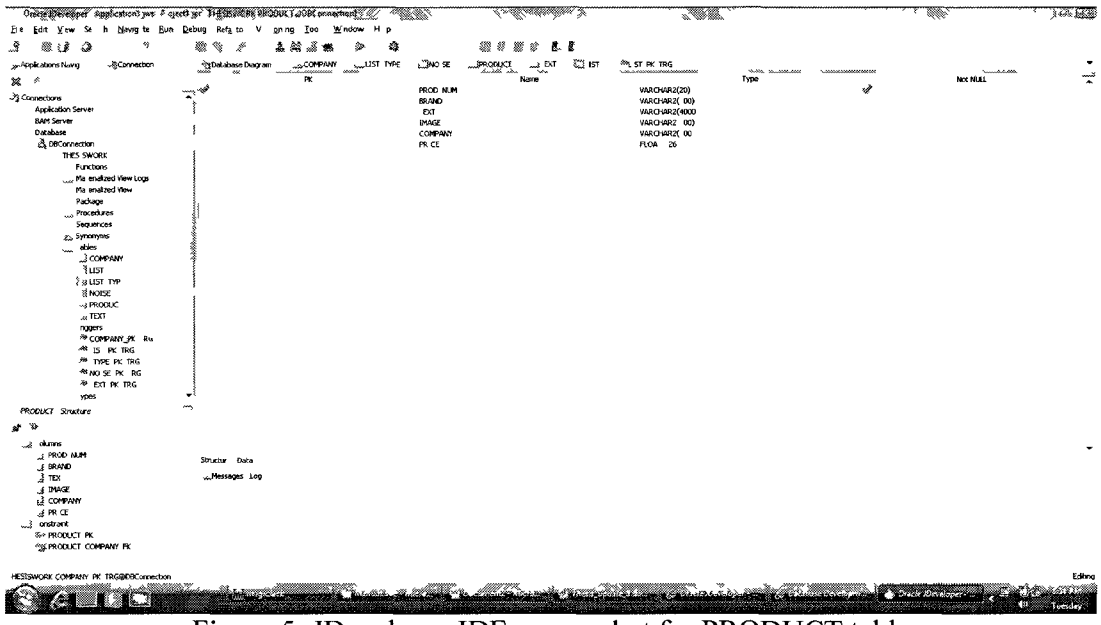


Figure-5: JDeveloper IDE screen shot for PRODUCT table

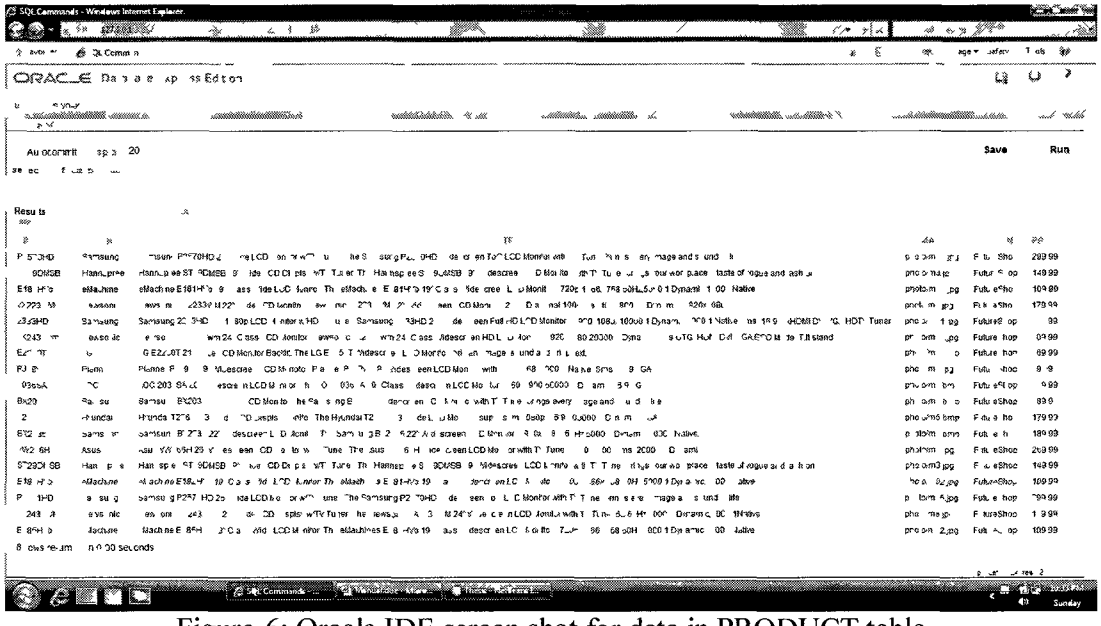


Figure-6: Oracle IDE screen shot for data in PRODUCT table

LIST_TYPE TABLE

This table keeps track among the contents of multiple lists from each web page as well as company (e.g., web page ID) from which those lists inserted from. The database schema for this table is: *List_Type (list_type string, id. integer);*

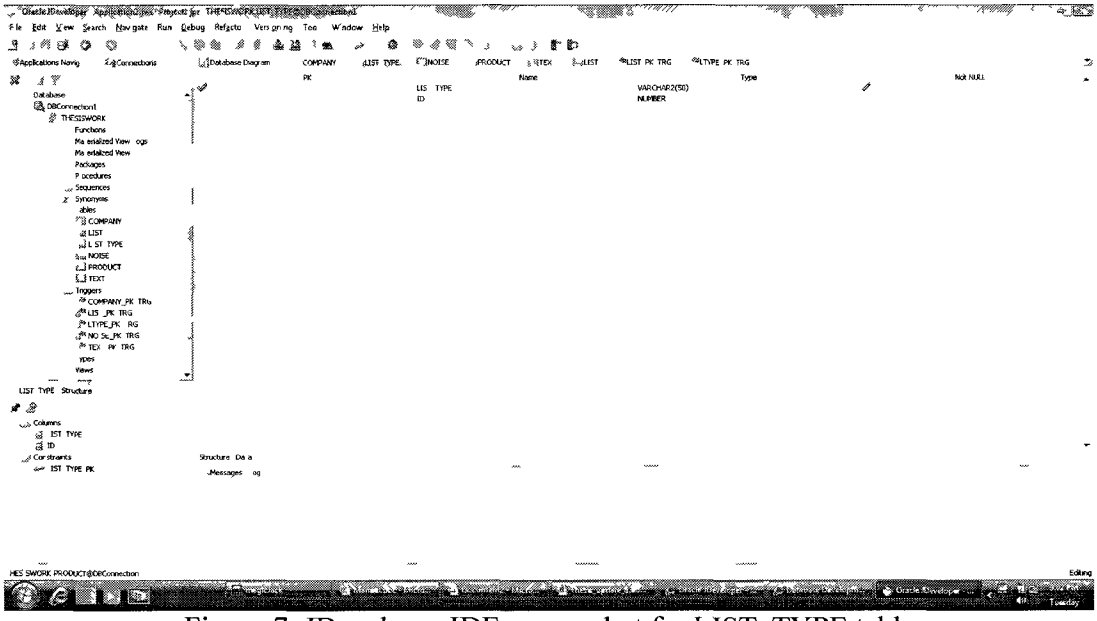


Figure-7: JDeveloper IDE screen shot for LIST_TYPE table

Figure-8: Oracle IDE screen shot for data in LIST_TYPE table

LIST TABLE:

This table stores content information of multiple lists from multiple web pages. The data base schema is given below:

List (list_id:int, link: string, text string, company: string, list_type:string),

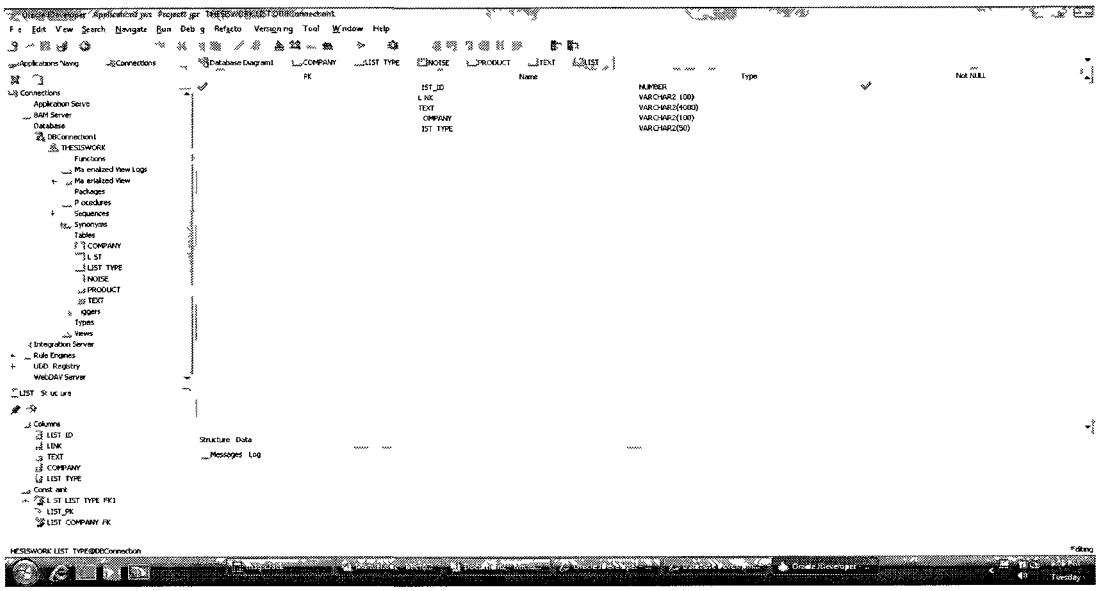


Figure-9: JDeveloper IDE screen shot for LIST table

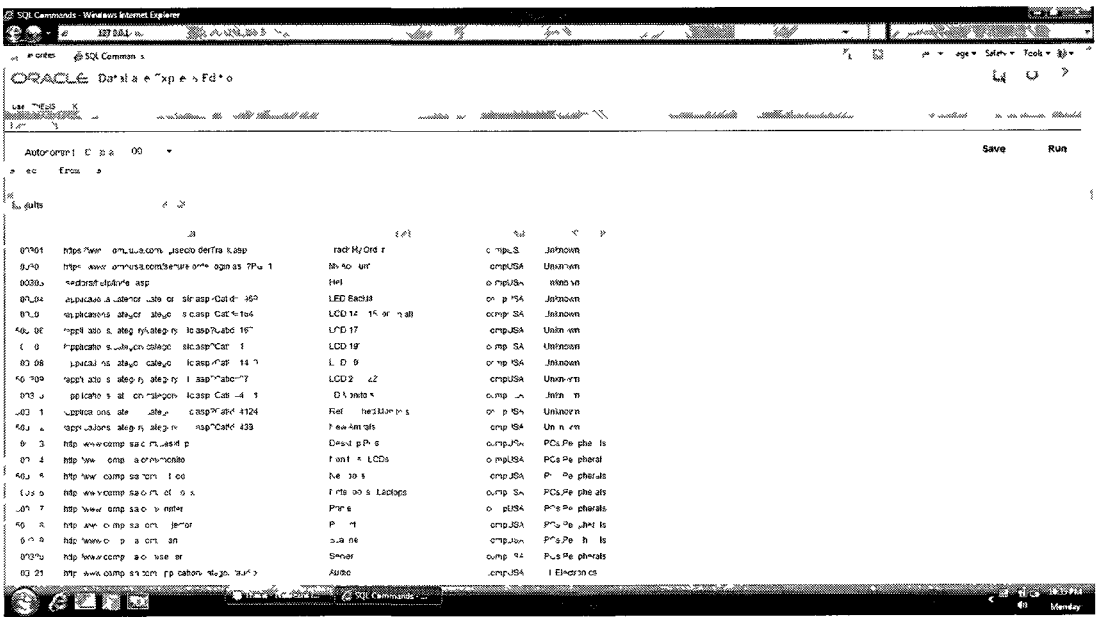


Figure-8: Oracle IDE screen shot for data in LIST table

TEXT TABLE: This table stores Text contents from the web pages. The database schema is given as: *Text(text_id: int, text: string, company: string);*

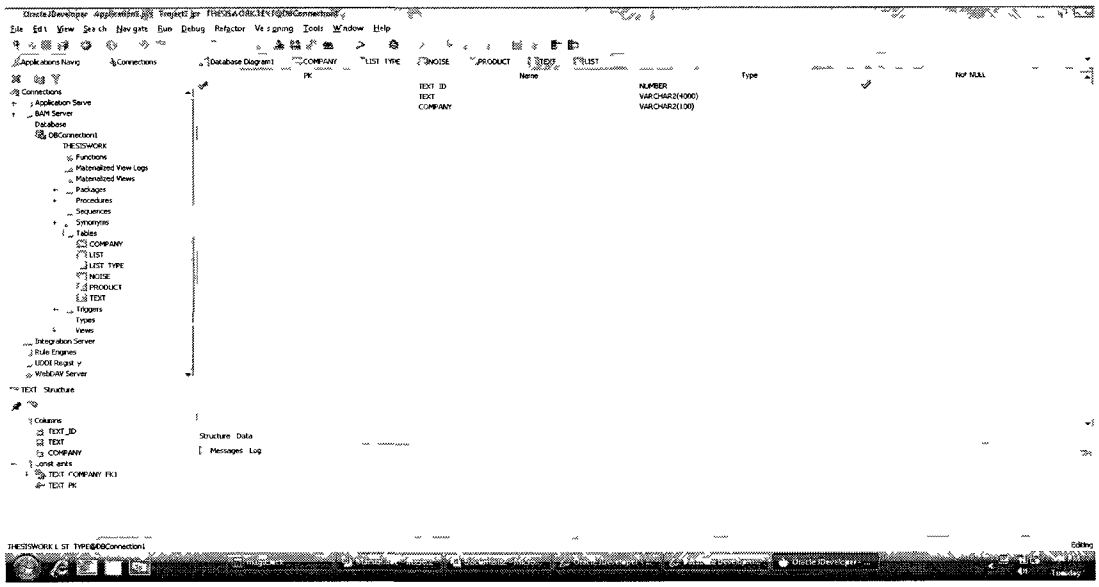


Figure-10: JDeveloper IDE screen shot for TEXT table

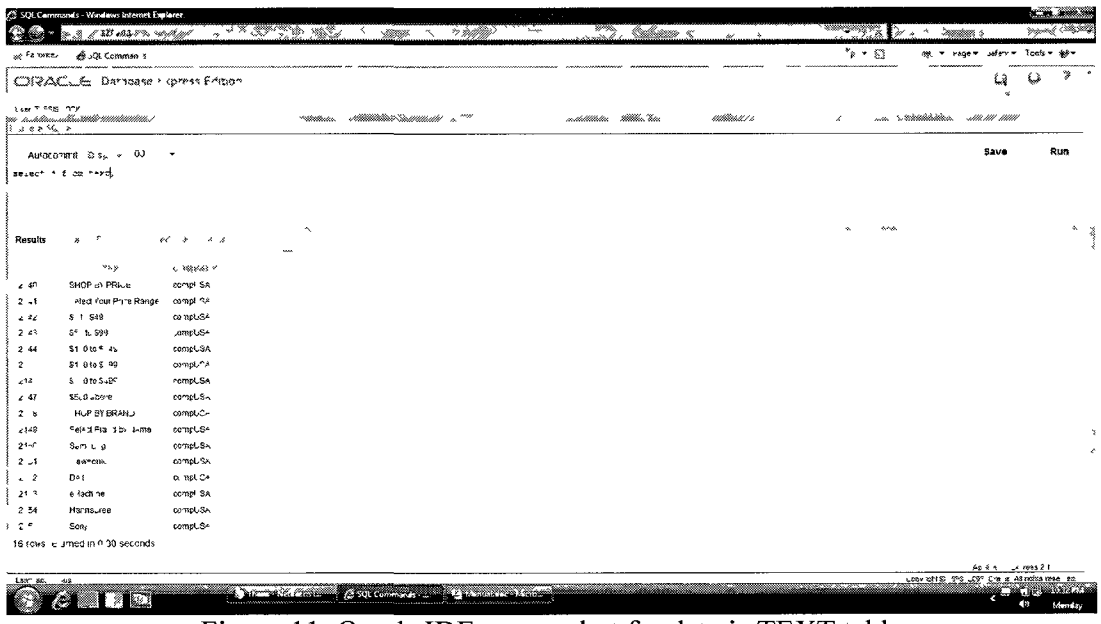


Figure-11: Oracle IDE screen shot for data in TEXT table

NOISE TABLE:

Noise table stores the noise contents from the web page. The database schema for noise table is: *Noise(id: int, link: string, image: string; company: string);*

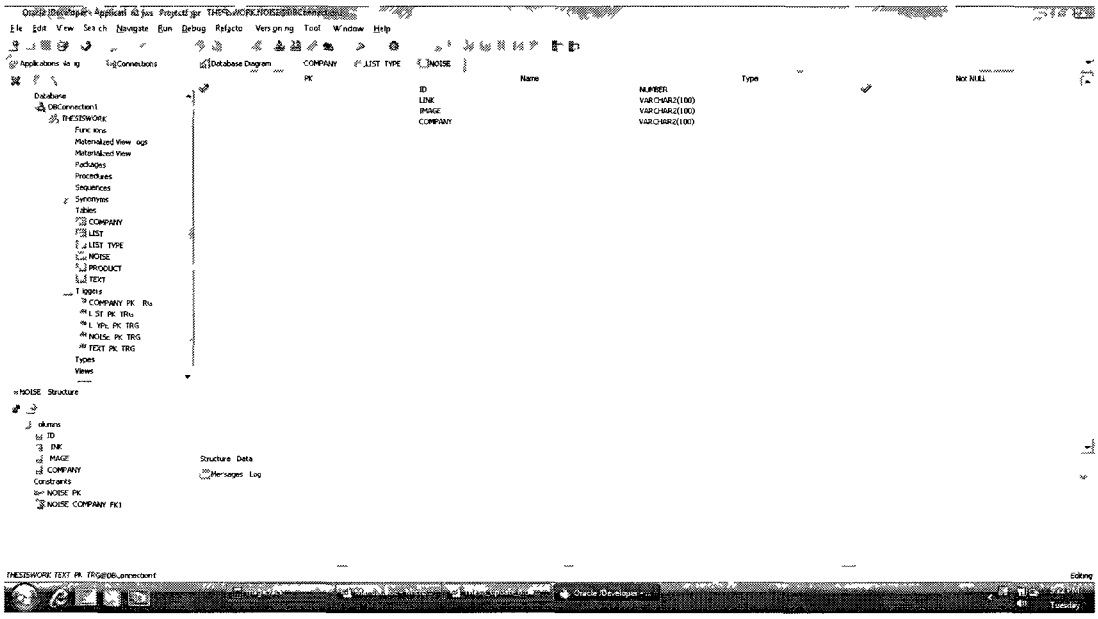


Figure-12: JDeveloper IDE screen shot for NOISE table

7.0 Limitations of the software

This is the very first effort for mining web contents using object oriented model. There is plenty of room for improvement of our system in future. Limitations of the current system and future work as identified are stated as below:

Crawler module: Current crawler module can take one URL string at a time for extraction and mining of web contents. Further improvement is required in future for automatic identification of positive web pages from the web. Present implementation of this module is designed aiming to work for basic functionalities as crawler with the functionality to download data stream from the targeted web pages into the local computer and cleaning or the web page comments from it. For robustness and scalability, we need to improve the current crawler module to handle all kinds of situations from the web.

Cleaner module: Currently we are using open source software "tagSoup" for cleaning the web pages. Development of an independent cleaner module may improve the systems performance and usability in future.

Extractor module: One major problem with the extractor module is its inability to handle long tag attribute values. A reasonable way to handle long HTML tag attribute value is needed that are currently blasting the DOM Tree creation. We need to find out a way to reduce the tag attribute value length without loss of resources from it. A reasonable solution by finding any alternative way to create DOM from other platforms may solve the problem. Another noticeable limitation of this system is its limited capacity to handle noise contents from the web page. More research is required to handle noise from data tuples.

Miner module: We introduced the idea of using NFA for mining web contents in this thesis. This NFA has two fold uses: Generation of extraction pattern for contents and generation of database schema, cardinalities to create tables and to store contents into relational database. In this thesis we generated extraction pattern but generation of database schema from the generated NFA is pending to develop. Another limitation is the use of pattern table for classification of tuples from the ContentObjectArray. Implementation of any automatic classification using co-sign or other similarity algorithm will eliminate the use of semi-automatic use of pattern table and squeezeTuple algorithm from the miner module.

8.0 Java Tools

We used a set of java tools for the development of WebOMiner system. Table - A list some important java tools we used for the development of the system and their reference URL's for future developer's reference.

Table – A

Java Tools	Reference URL
ArrayList	http://download.oracle.com/javase/1.4.2/docs/api/java/util/ArrayList.html
BufferedInputStream	http://download.oracle.com/javase/1.4.2/docs/api/java/io/BufferedInputStream.html
Class	http://download.oracle.com/javase/1.5.0/docs/api/java/lang/Class.html
File	http://download.oracle.com/javase/1.4.2/docs/api/java/io/File.html
Hashset	http://download.oracle.com/javase/1.4.2/docs/api/java/util/HashSet.html
HttpURLConnection	http://download.oracle.com/javase/1.4.2/docs/api/java/net/HttpURLConnection.html
InputStream	http://download.oracle.com/javase/1.4.2/docs/api/java/io/InputStream.html
InputStreamReader	http://download.oracle.com/javase/1.4.2/docs/api/java/io/InputStreamReader.html
javax.xml.parsers.DocumentBuilder	http://download.oracle.com/javase/1.4.2/docs/api/javax/xml/parsers/DocumentBuilder.html
javax.xml.parsers.DocumentBuilderFactory	http://download.oracle.com/javase/1.4.2/docs/api/javax/xml/parsers/DocumentBuilderFactory.html
NodeList	http://download.oracle.com/javase/1.4.2/docs/api/org/w3c/dom/NodeList.html
org.w3c.dom.traversal.DocumentTraversal	http://download.oracle.com/javase/1.5.0/docs/guide/plugin/dom/org/w3c/dom/traversal/package-tree.html
org.w3c.dom.traversal.NodeFilter	http://download.llnw.oracle.com/javase/1.5.0/docs/guide/plugin/dom/org/w3c/dom/traversal/class-use/NodeFilter.html
org.w3c.dom.traversal.NodeIterator	http://download.oracle.com/javase/1.5.0/docs/guide/plugin/dom/org/w3c/dom/traversal/class-use/NodeIterator.html
Process	http://download.oracle.com/javase/1.4.2/docs/api/java/lang/Process.html
Runtime	http://www.science.uva.nl/ict/ossdocs/java/jdk1.3/docs/api/java/lang/Runtime.html
String	http://download.oracle.com/javase/1.5.0/docs/api/java/lang/String.html
Sql	http://download.oracle.com/javase/6/docs/api/java/sql/package-summary.html
StringTokenizer	http://download.oracle.com/javase/1.4.2/docs/api/java/util/StringTokenizer.html
Thread	http://download.oracle.com/javase/1.4.2/docs/api/java/lang/Thread.html
URL	http://download.oracle.com/javase/6/docs/api/java/net/URL.html

VITA AUCTORIS

NAME:	Titas Mutsuddy
PLACE OF BIRTH:	Chittagong, Bangladesh.
YEAR OF BIRTH:	1968
EDUCATION:	Chittagong University of Engineering and Technology, Chittagong, Bangladesh. Bachelor of Science in Civil Engineering (1992)
	University of Windsor, Windsor, ON, Canada Bachelor of Computer Science (Honors) (2005)
	Wayne State University, Detroit, MI, USA Master of Science in Civil Engineering (2008)
	University of Windsor, Windsor, ON, Canada Master of Science in Computer Science (2010)