Wrapping Web Pages into XML Documents

By

Tao Fu

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Science
Information and Systems Science

School of Computer Science
Carleton University
Ottawa, Ontario

March 18, 2004

© Copyright
2004, Tao Fu

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services

Acquisitions et
services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-93980-4

Our file Notre référence

ISBN: 0-612-93980-4

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this dissertation.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de ce manuscrit.

While these forms may be included in the document page count, their removal does not represent any loss of content from the dissertation.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

PAGINATION ERROR.

TEXT COMPLETE.

ERREUR DE PAGINATION.

LE TEXTE EST COMPLET.

Abstract

It has frequently been observed that most of the world's data lies outside database systems. The reason for this is that database systems focus on structured data, leaving the unstructured realm to other formats, such as HTML, which is designed specifically for human browsing rather than for applications to automatically access. While XML is gaining in popularity as an industrial standard for presenting and exchanging structured information on the Web, we argue that in order to broaden the use of the tremendous amount of information on the Web and afford automation, inter-operation and intelligent services, we need a concerted effort to cross this structural chasm. To reach this goal, some Information Extraction programs, called *wrappers*, have been developed to extract web data from HTML documents into a structured, feature-rich XML format. A conventional method would have been to either write extraction rules manually, known as *knowledge engineering*, or use human-annotated training examples for automatic *wrapper induction*. Both approaches require intensive human input and are prone to failure when the web source changes, as frequently happens in practice.

In this thesis, we present a novel approach to separate the data layer from its presentation in HTML and extract the pure data as well as its hierarchical structure into XML. This approach aims to offer a general purpose methodology that can adaptively build the wrappers without any fine-tuning for the particular domain. The intuition behind the strategy described here is that the task of extracting information from semi-structured web documents should take advantage of structural clues that can guide the process to the right extraction

points. For this purpose, we introduce the *Transitional Document Object Model* to facilitate the conversion from HTML to XML. It works like a well-organized bookshelf, where the HTML nodes are re-arranged so that the wrapper programs can pick them up more easily. Benefiting from this middle-tier model, a highly accurate and robust web information extraction system is developed and a web-based application called *XML Gateway*, which is able to do the conversion from HTML to XML on-the-fly, is implemented.

Acknowledgements

First of all I would like to express my sincere thanks to my supervisor Dr. Mengchi Liu, for being such a wonderful advisor. Through his support, care, and patience, he has transformed a struggling graduate student into an experienced researcher. His insight and ideas formed the foundation of this thesis as much as mine did, and his guidance and care helped me get over various hurdles during my graduate years.

I would like to express my gratitude to my wife, Wenjun Ge, for her love and support. Frankly, it is quite tough to be the wife of a graduate student, but she went through the last 4 years without many complaints. I don't think I can ever make up for her lost sleep because of my typing at 4 in the morning, but I will be able to give her as much support as I got for the rest of my life.

Special thanks to Xiaofei Jia, Zhen Yuan, Zhihong Li and Shan Wei for being the readers of my thesis. After reading a draft, they provided helpful comments on various aspects of my thesis. These comments made my thesis much better, and I learned a lot from their comments. I would also like to thank other members of the SCS database group who give me help during my graduate studies.

Contents

Abstract	iii
Acknowledgements	v
List of Tables	x
List of Figures	xii
1 Introduction	1
1.1 Thesis Goals	3
1.2 Thesis Outline	4
2 Background	6
2.1 Brief History of Information Extraction	6
2.2 Web IE Related Concepts	7
2.2.1 Semi-structured Documents	8
2.2.2 Difference between IE and IR	8
2.2.3 HTML and XML	9
2.3 A Brief Survey of Web IE Techniques	11
2.3.1 Challenges for Web IE Technology	11
2.3.2 Evaluation Metrics	12

3	Related Works	14
3.1	Knowledge Engineering and Machine Learning	14
3.2	Domain-Specific and Domain-Independent	18
4	Models and System Overview	22
4.1	W3C Document Object Model	22
4.2	Transitional Document Object Model	27
4.2.1	Mathematical Formalization of the DOM	28
4.2.1.1	Functions for Nodes	29
4.2.1.2	Functions for Relations	30
4.2.2	Atomic Units of Web Information	32
4.2.3	Hierarchical Relations of Web Data	34
4.3	System Overview	36
5	Content Partition	41
5.1	High-level Content Blocks	42
5.1.1	Find the Main Body Blocks	43
5.1.1.1	Node Density Formula (NDF)	44
5.1.2	Find the Side Bars	46
5.2	Detect the Block Boundaries	48
5.2.1	Explicit Boundaries	48
5.2.2	Implicit Boundaries	50
6	Topic Aggregation	53
6.1	Preliminaries	54
6.2	Extracting the Hierarchies in One-Dimensional Space	58
6.2.1	Static Clustering	58
6.2.1.1	Heading	58

6.2.1.2	Indention	60
6.2.2	Dynamic Clustering	62
6.2.3	Overall Algorithm for One-dimensional Hierarchy	65
6.3	Extracting the Hierarchies in Two-Dimensional Space	67
7	Adaptive Web Information Extraction	70
7.1	Semantic Annotation	70
7.1.1	Annotation by Topic nodes	71
7.1.2	Annotation By Regular Expressions	75
7.2	Schema Discovery	77
7.2.1	Instance-level Approaches	77
7.2.2	Schema in Data-rich Section	79
7.3	Wrapper Validating and Maintenance	84
8	Design and Implementation of XML Gateway	87
8.1	Requirements of the IE System	88
8.2	XML Gateway System Architecture	89
8.3	Retrieving the Web Documents	90
8.4	Extracting Language	92
8.5	XSL Templates Factory	93
8.5.1	Content Partition Filters	94
8.5.2	Topic Aggregation Filters	95
8.5.3	Pattern Recognition Filters	97
8.6	Experiments and Evaluation	98
8.6.1	Experiment Setup	99
8.6.2	Results and Analysis	99

9 Conclusion and Future Work **103**

9.1 Conclusion 103

9.2 Future Work 106

Bibliography **108**

List of Tables

5.1	Distributions of records in high-level content blocks.	44
8.1	Recall and Precision results for pages from non-commercial sites	100
8.2	Recall and Precision results for pages from commercial sites	100

List of Figures

2.1	Precision and Recall	12
4.1	A partial view of Yahoo's homepage	24
4.2	A partial view of HTML DOM tree for example 4.1.1.	25
4.3	A partial view of XML DOM tree for example 4.1.1	26
4.4	The consistent records are grouped into content blocks	35
4.5	Transforming functions for building the Transitional DOM	36
4.6	Mapping the Transitional DOM to the XML DOM.	38
4.7	The abstract model of the automatic Web IE system	39
4.8	The transition from HTML to XML	40
5.1	Content Blocks Identification	42
5.2	Node Density, calculation of hop detection for Carleton's homepage	46
5.3	The high-level content blocks in Carleton's homepage	47
5.4	Explicit boundaries detected by block-level tags	49
5.5	Detecting implicit boundaries by projection algorithm	51
6.1	Topic nodes are on the same level in the DOM tree for the Example 5.2.1. .	55
6.2	A $m \times n$ table showed on Yahoo's homepage.	56
6.3	Topic nodes in the two-dimensional space for the Example 6.1.2.	57
6.4	HTML Headings for Describing the Course Syllabus	59

6.5	Indentions created with HTML list and quotation tags	60
6.6	Indention created with HTML definition list tags	61
6.7	The same hierarchical structure represented by different HTML tags	63
6.8	Using the <i>class</i> attributes to indicate the hierarchical relations	64
6.9	Detect the topic nodes via calculating the similarity distance.	65
6.10	The process of table normalization	69
6.11	Reducing the two-dimensional table to a set of one-dimensional spaces . . .	69
7.1	An old version of the web page from SCS Carleton U.	74
7.2	The current version of the web page from SCS Carleton U.	75
7.3	Documents in Example 7.1.2 after content partition and topic aggregation .	76
7.4	Data-rich section in web pages from Amazon site	82
7.5	The traditional wrapper and the multi-functional wrapper	85
8.1	System architecture of the XML Gateway wrapper system.	90
8.2	A basic XSL filter unit on a XTF pipeline which transforms DOMs in re- spect of the templates.	93
8.3	XTF filters for content partition pipeline.	94
8.4	XSL templates for topic-level hierarchical reconstruction.	96
8.5	XSL templates for absolute and relative patterns extraction.	97

Chapter 1

Introduction

HTML pages, where the data intermingles with representational aspects, are designed specifically for display purposes and users access the web data mainly by browsing. To browse this enormous collection of on-line documents and gain the desired information can be a time-consuming and tedious process for users. Although the Web, in some ways, looks like a global database, querying the Web and getting the result directly is difficult, and in many cases not yet feasible due to the heterogeneity and the lack of structure of the web sources.

With the help of some Information Retrieval (IR) tools, such as web search engines, people can target the relevant web pages more quickly, but still need further processes to extract the pertinent data from them. There is an urgent need to make web information accessible to applications so as to afford automation, inter-operation and intelligent services. This mission will be easier if we can access the web information sources in a structured way.

To address these problems, W3C has released a new industrial standard called eXtensible Markup Language (XML), which is a pure data representation language with semantic

markup rather than formatting markup. By separating the data from its representation, XML has a profound impact on the way data is exchanged on the Internet. It is gaining in popularity as an application-independent way of presenting and exchanging structured data on the web.

However, the majority of web pages are still marked up with HTML and will not be replaced by XML in the near future. As a result, companies are busily converting HTML sources so these can be placed into Knowledge Management systems or incorporated within business-to-business applications. In the past few years, many Information Extraction (IE) systems [30, 22, 8, 13, 21] have been developed to extract web data and populate it into traditional databases for further handling [17]. As a key enabling technology for these strategies, these systems aim to pull information from documents in heterogeneous formats, such as PDF files, emails, and web pages, and convert it to a structured database format [15]. In the context of web extraction, the conversion is usually carried out from HTML to XML. Once the web information is extracted and mapped into XML, we can either populate the web data into a traditional database, or utilize the abundant XML integration techniques to manipulate the data efficiently.

A conventional approach for building a web information extraction system is to write programs, called *wrappers*. Researchers have proposed several distinct techniques to address the issue of generating wrappers for Web data extraction [23, 29, 25, 24, 1, 38, 11, 28, 5]. Such systems either rely on extraction rules tailored to particular document collections or, using machine training algorithms for rapid wrapper induction, observe the extraction rules in person and write programs for the specific domain. In other words, the wrapper for the pages of one web site will be different from the wrapper for the pages of a second web site. The disadvantages of such approaches are:

1. They require intensive human input either in the form of hand-coded rules or human-annotated training examples;

2. Wrappers are prone to failure when the layout of the web page changes, as frequently happens in practice.

For overcoming these problems, a new technique has been introduced in [19]. By utilizing the structural information of an input HTML document, the technique can extract the data information from the document and convert it into a corresponding XML document in a fully automated manner. In this thesis, we develop a web information extraction system which is built upon this technique.

1.1 Thesis Goals

The overall goal of the work presented in this thesis is to improve knowledge management and information integration techniques with an adaptive information extraction system, more specifically in the context of intelligent automated processing of information provided by World Wide Web services. The following list describes the goals in more detail:

- Propose an approach for an adaptive information extraction system in semi-structural environments;
- Build an abstract representation of the information encoded in HTML documents;
- Suggest a set of suitable algorithms to handle the semi-structural information;
- Find a general purpose methodology to organize and transform the structured data;
- Implement a prototype system capable of intelligent service;
- Indicate future topics of research.

1.2 Thesis Outline

This thesis has three parts. The first part introduces background knowledge regarding the web IE and different dimensions of related work. The second part discusses strategies for eventually separating the data from its presentation layer and putting the highly structured data into XML. The third part describes the implementation of the prototype system and the evaluation of our IE approach. The thesis ends with a summary and conclusion, and by posing future research questions. The content of the chapters, is described in more detail, is as follows:

- Chapter 2 presents some background knowledge that is important to understanding this thesis. The introduction focuses on the differences between concepts such as XML and HTML, IR and IE, precision and recall. It also concludes by noting some of the challenges and benefits of web IE research.
- Chapter 3 is an overview of related work. Based on the main technique they employ, we briefly group the strategies into three classes: Natural Language Processing (NLP) based, HTML structure-aware, and wrapper induction approaches based on machine learning. By analyzing their capabilities and features, we identify an important trade-off between the degree of automation of a tool and the flexibility of the wrappers generated by it.
- Chapter 4 gives a formal definition of a transitional object model that extends from HTML specifications. After that, we provide a system abstract model for the first look of the whole web IE system.
- Chapter 5 separates the single content block of a web page into high-level content blocks, then iteratively splits each content block into smaller ones using boundary detection algorithms.

- Chapter 6 introduces the topic aggregation process to explore the hierarchical structure hidden in the presentational tags. We separate the process into different dimensional spaces and develop the algorithms to solve the two-dimensional aggregation by reducing it to one-dimensional space.
- Chapter 7 accomplishes the data extraction process using semantic annotation, schema discovery and template building.
- Chapter 8 focuses on the implementation of a prototype system called “XML Gateway” which can process the HTML to XML conversion on-the-fly. We also evaluate our system sitting on practical experiments.
- Chapter 9 presents the conclusions of this thesis and describes directions for future research.

Chapter 2

Background

This chapter provides background concepts that are important to understanding the motivation of the thesis. We start, in Section 2.1, with a brief history of IE research. Section 2.2 gives some IE-related concepts and focuses on identifying the differences between them. Section 2.3 provides a brief survey of the challenges and evaluation metrics of Web-focused IE techniques.

2.1 Brief History of Information Extraction

The field of automatic information extraction is a new area, and has only been driven forward in the last decade or so. A precursor of IE is the Artificial Intelligence (AI) field of text comprehension. Researchers have been building various systems whose goal is to obtain an accurate representation of the contents of an entire textual article. These systems typically operate in very small domains only, and they are usually not very portable to new domains [48]. There are two factors which have been important stimulations for the development of these systems: the exponential growth in the amount of both on-line and off-line

textual documents, and the focus on the field through Message Understanding Conferences (MUCs)¹.

Since the late 1980s, the US government has been sponsoring MUCs to evaluate and advance state-of-the-art processes in information extraction. MUCs involve a set of participants from different sites, usually a mix of academic and industrial research labs. Each participating site builds an IE system for a predetermined domain. The IE systems are all evaluated on the same domain and text collection, and the results are scored using an official scoring program. The objective of the conferences has been to perform a quantitative evaluation of IE systems, which prior to these conferences had been performed only sporadically and often on the same data on which the systems had trained. The conferences provided the first large-scale effort to evaluate natural language processing systems. The methods of system evaluation have been a non-trivial issue, and through the conferences standard scoring criteria have been developed. The focus of the various MUCs has been on tasks, and the latest task of MUC-7 is to develop text understanding systems that annotate the text without pre-processing or human intervention².

While the MUCs have encouraged the development of IE for the natural language documents community, the explosive growth of the Web is responsible for the increasing popularity of wrappers. The need for tools that can extract and integrate data from heterogeneous web sources led to the development of the web IE field.

2.2 Web IE Related Concepts

Before we address Web IE issues, the following concepts are used frequently and need to be understood clearly.

¹MUC is a conference series focusing on information extraction research and is sponsored by the Defense Advanced Research Projects Agency (DARPA).

²<http://www.cogsci.ed.ac.uk/poesio/MATE/muc7.html>

2.2.1 Semi-structured Documents

A problem with the IE techniques of today is that they are mostly targeted for natural language text, i.e. unstructured text. If more structural information such as tables or lists is present in the document, the system will often fail to employ the linguistic and statistical methods. The term “semi-structured” refers to information that contains more structural information than natural language text, but not as structured as in database systems.

A typical semi-structured document is a World Wide Web hypertext document in HTML specification. Some HTML documents may have an irregular structure that does not adhere to any predefined schema, but does have an implicit underlying structure, such as online catalogs, that can be exploited to extract data. The information contained within these documents is not ordered as rigidly as database entries, but does contain some consistent formatting.

Consider a source such as the CIA World Fact Book.³ This source provides information on each of the world’s 269 countries, with information for each country presented on a separate page. The information on each page is presented in a semi-structured manner, since each page can be clearly sub-divided into distinct sections with headings denoting the beginning of each section. Furthermore, the information on each page is presented in exactly the same format and there are clearly identifiable sections such as “Geography”, “Area” and “Land boundaries”.

2.2.2 Difference between IE and IR

Information Extraction (IE) must not be confused with Information Retrieval (IR), the technology usually adopted by search engines. Whereas the goal of IR is the retrieval of relevant documents which match an end-user’s query, IE transforms a collection of documents,

³<http://www.cia.gov/cia/publications/factbook/>

usually with the help of IR systems, into information that is more readily digested and analyzed. For instance, by typing the words “weather of Ottawa” in a search engine, the user will obtain a list of all documents which contain these words. On the other hand, by issuing the same query to an IE system, the answer obtained will be the current temperature and weather forecast for the city of Ottawa.

IR tools, like web search engines, provide a quick way of targeting the relevant documents, but require further processes to extract data from these documents. A key element of IE systems is a set of text extraction rules or extraction patterns that identify the relevant information to be extracted [27]. Information extraction software requires that end-users specify in advance the categories of information they want to capture from a text. For instance, a system devoted to scanning financial news stories could extract all company names, interest rate changes, Stock Exchange Corporation announcements, or stock market quotes from texts. Because the parameters that define a particular topic are determined *a priori*, IE systems are fully customizable.

The difference between IR and IE is one of granularity regarding information access. IR is document retrieval and IE is fact retrieval. IR and IE are complementary. Together they create powerful new tools for accessing and organizing information stored on Web sites.

2.2.3 HTML and XML

Most of the documents on the Web are written in HyperText Markup Language (HTML). HTML was first introduced to mark up information only according to its meaning, such as <title> and <address>. Soon after it was opened to the public as a standard language for describing the contents and appearance of web pages. As a matter of convenience for HTML authors, browser vendors have introduced their own elements and attributes whose

only purpose is to specify the layout, such as `<font>`, `<center>` and `bgcolor`. They have even created HTML editors in which the markup is presentational rather than semantic. For instance, it uses `<ul>` to produce indentation, not just for lists, and uses `<table>` to arrange contents, not just relational data. Finally, HTML has become a layout-oriented language as a result of user requirements.

In the summer of 1996 W3C set up a working group that is now creating this new standard called XML (eXtensible Markup Language), which is designed for presenting and exchanging structured information on the Web. It allows a user to define the hierarchical relations between data as well as semantic descriptions of the data itself. With the separation of underlying data from the representation layer, XML is more powerful than HTML in making web information accessible to applications.

Experience has shown that separating the structure of a document from its representational aspects reduces the cost of serving a wide range of platforms and media, and facilitates document revision. XML is an important step towards offering efficient resource discovery on the Web, although it does not completely solve all problems. XML is important because it facilitates increased access to and description of the content contained within documents. The technology separates the intellectual content of a text from its surrounding structure, meaning that information can be converted into a uniform structure.

Despite the advances of XML, the majority of documents on the Web are still marked up with HTML, and companies are busily converting HTML to XML so documents can be placed in content-management systems and enterprise applications, published on portals, or incorporated into applications.

2.3 A Brief Survey of Web IE Techniques

2.3.1 Challenges for Web IE Technology

Although web documents, which are the target of web IE systems, exhibit syntactic regularities, information extraction from these sources is not trivial. Scalability is for instance an important challenge, as there is both a large number of sites and a large variation in formatting styles. Flexibility is another significant challenge due to the dynamic characteristics of the Web, where both the content and format of documents frequently change. Overall, to meet all the challenges raised in the web IE area, an effective IE system should satisfy the following conditions:

1. The extraction results are highly accurate and robust, while the extraction process is highly automatic and quick.
2. The wrapper generation should enable the user to create complex, structurally organized extraction patterns, which support expression of user needs correctly and comprehensively.
3. The extraction patterns should be flexible enough to cover a large class of similar web pages while also being applicable to pages after their structure or contents have been altered.
4. Constructing wrappers should be straightforward for most people, even for those with no knowledge of programming or of a particular programming language.

Scalability is the key challenge to automatic information extraction, because each source might format its content differently, and therefore could require a customized set of extraction patterns.

2.3.2 Evaluation Metrics

IE standards of measurement grew out of the IR metrics of “Recall” and “Precision”, but the definitions of these measurements were altered. In IR systems, recall is the ratio of the number of relevant documents retrieved to the total number of relevant documents, and precision is the ratio of the number of relevant documents retrieved to the total number of documents retrieved. In the IE task, recall can be interpreted as a measure of the percentage of records that have been extracted, and precision as a measure of the percentage of extracted records that are correct. Recall and precision are defined as follows:

$$Precision = \frac{|\{Correct\} \cap \{Extracted\}|}{|\{Extracted\}|}$$

$$Recall = \frac{|\{Correct\} \cap \{Extracted\}|}{|\{Correct\}|}$$

Figure 2.1 shows the spectrum of “Correct Records”, “Extracted Records” and their com-

Figure 2.1: Precision and Recall

bination “Correctly Extracted Records”. Note that the concept of a correct record does not mean we have “incorrect” records on web pages. A correct record implies a meaningful information fraction that can be part of a specific topic. Both recall and precision are always on the interval $[0, 1]$, their optimum being at 1.0.

When comparing the performance of different systems, both recall and precision must be considered. However, as it is not straightforward to compare the two parameters at the same time, a combined measure called “F-measure” has been proposed. F-measure combines precision P and recall R in a single measurement as follows:

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

The parameter β determines how much to favor recall over precision. Researchers in information extraction frequently report the F score of a system with $\beta = 1$, weighting precision and recall equally. Using the *F-measure*, the relative performance of systems reporting different values for recall and precision can easily be compared.

Chapter 3

Related Works

Previously, researchers proposed several web IE systems for the rapid generation of wrappers. Approaches for generating wrappers can be differentiated in a number of ways. For instance, they can be categorized by their development languages, output results, algorithm designs, use of a Graphical User Interface (GUI) or several other characteristics. We compare our work with other related systems in two respects: degree of automation and domain of patterns. Section 3.1 analyses web IE systems in terms of their automation features. Section 3.2 evaluates the web IE systems by the domain of the extraction patterns.

3.1 Knowledge Engineering and Machine Learning

A very important feature of any information extraction system is its degree of automation. This is related to the amount of work left to the users during the process of generating a wrapper for extracting web data.

Before Nickolas Kushmerick, the founder of wrapper induction [15], wrappers were typically hand-coded, a method known as “knowledge engineering”. Such systems as

RAPIER [9], SRV [18] and WHISK [39] are based on natural language processing techniques, and the hand-coded rules to be applied by the system are constructed manually by a syntactic analyzer and semantic tagger. This strategy is applied to the construction of the grammar rules as well as to the discovery and formulation of the domain patterns. Of course, effective rules can be constructed only iteratively, by starting with simple rule sets, evaluating the results and refining them stepwise. The users still must examine the document and find the HTML tags that separate the objects of interest, and then write a program to detect the object regions. In other words, the process of discovering object boundaries is also carried out manually. Thus the knowledge engineering approach is rather labourious.

To improve on this, Kushmerick introduces wrapper induction IE system [26], which is a method for automatic wrapper generation that uses machine learning techniques. Machine learning is a technique to create adaptive information extraction systems that automatically learn extraction rules. A pioneer wrapper induction system is Head-Left-Right-Tail (HLRT), which defines the wrapper induction problem as finding a wrapper of a given class that maps the given set of sample pages and associated labels, thus using automated training for the construction of the wrappers. Different classes of wrapper are presented and an evaluation per correctly wrapped page is proposed. There seems to be little more than pattern matching rules induced by the learning algorithms presented. The general algorithm is to find a delimiter, extract until another delimiter and then find the next delimiter and so on. HLRT identifies the delimiters for the head and tail of the page in addition to the left and right for each attribute, thus providing support for more sophisticated pages. However, the labelling of the example pages used for the supervised training is often performed manually.

STALKER [11] is another wrapper induction system that builds wrappers by learning from samples. It takes advantage of web pages that have a high degree of regular structure. By analyzing the regular structure of sample pages, its wrapper induction process can detect

landmarks and extract desired fields. For example, a set of street addresses “12 Bank St.”, “512 Oak Blvd.”, “416 Main St.” and “97 Adams Blvd”, all start with a pattern number, followed by a capitalized name and ending with “Blvd.” or “St.” STALKER refers to the starting and ending patterns together as the data prototype of the field. The advancement in this approach is the bootstrapping characteristic: given the initial examples provided by the user, it first learns a wrapper and then uses this wrapper to obtain many more examples that can be analyzed in much greater depth. Thus, by leveraging a few human-provided examples, STALKER ends up with a highly scalable system for wrapper creation and maintenance.

Both HLRT and STALKER systems involve the use of manually indexed documents to “train” the system as to what attributes make up desired content, also called supervised training, and improves its performance based on experience. To apply this approach, we need annotated corpora. For example, a name recognizer would be trained by annotating a corpus of texts with the domain-relevant proper names. Alternatively, training data can be obtained by close interaction with the user, where the system proposes new rules, which are either confirmed or rejected by the user. Moreover, even though corroboration is introduced, it remains a tedious task to tag by hand all the instances contained in every sample document. From that point of view, systems based on NLP and wrapper induction are said to be semi-automatic, because the user has only to provide examples that guide the generation of the wrapper. Furthermore, if the user fails to provide a good set of samples, the proposed evaluation gives poor results to wrappers favoring robustness over perfection.

A recently proposed method that further explores the inherent features of HTML documents to automatically generate wrappers is ROADRUNNER [12]. As a fully automatic approach, its model of page creation using a template is very similar to ours. ROADRUNNER starts by comparing the HTML structure of two (or more) given sample pages belonging to the same “class,” generating a schema for the data contained in the pages.

From this schema, a template is inferred which is capable of recognizing instances of the attributes identified for this schema in the sample pages (or in pages of the same class). To accurately capture all possible structural variations occurring on pages of the same class, it is necessary to provide more sample pages. The entire extraction process is based on an algorithm that compares the tag structure of the sample pages and generates regular expressions to handle structural mismatches found between the two structures. In this way, the algorithm discovers structural features such as tuples, lists and variations. It should be noted that the process is fully automatic and no user intervention is requested.

Comparing to our approach, there are several limitations to the ROADRUNNER system:

1. ROADRUNNER assumes that every HTML tag in the input pages is generated by the template. This assumption is crucial in ROADRUNNER to check whether an input page can be generated by the current template. This assumption is clearly invalid for pages in many web-sites, since HTML tags can also occur within data values. For example, a book review in an Amazon¹ site could be in several paragraphs, in which case it contains `<p>` tags, or some words in the review could be highlighted using `<i>` tags. When the input pages contain such data values, ROADRUNNER will either fail to discover any template, or produce an incorrect template.
2. ROADRUNNER assumes that the grammar of the template used to generate the pages is union-free. This is equivalent to the assumption that there are no disjunctions in the input schema. For example, the address information in a web page could be in one of two formats, based on whether the address is a US address or not, in which case the schema of the address is a disjunction of the schema for US addresses and the schema for non-US addresses. The authors of ROADRUNNER themselves have pointed out in [12] that this assumption does not hold for many collections of

¹<http://www.amazon.com>

pages. Moreover, as the experimental results in [12] suggest, ROADRUNNER may fail to produce any output if there are disjunctions in the input schema.

3. When ROADRUNNER discovers that the current template does not generate an input page, it performs a complicated heuristic search involving backtracking for a new template. This search is exponential in the size of the schema of the pages. It is, therefore, not clear how ROADRUNNER would scale to web page collections with a large and complex schema.

3.2 Domain-Specific and Domain-Independent

To improve accuracy and ease development, IE software is usually domain or topic specific. An IE system designed to monitor technical articles about Information Science, for example, could pull out the names of professors, research studies, topics of interest, conferences or forthcoming publications from press releases, news stories, or emails, and encode this information in a database. End-users could then search across this database by textual attribute or feature. A typical search could be for all forthcoming publications about information retrieval or to locate all conference presentations on a specific information science topic.

With respect to extraction domain, the web IE systems can be classified into domain-specific systems and domain-independent systems [15]. A wrapper by its nature is highly web source or domain specific, since it is a program specific to a web source, which extracts the requested information and returns the result. As the structural and presentation features of web pages are prone to frequent changes, a crucial property of wrappers is “resilience”, i.e., the capacity to continue to work properly despite changes in targeted pages. It is also desirable that a wrapper built of pages of a specific web source on a given application domain work properly with pages from another source in the same application domain.

Such a property is called “adaptiveness”. If a wrapper can be constructed to capture enough domain-independent features of the data to be extracted, the wrapper generated by the web IE system will be inherently resilient and adaptive.

Most IE approaches based on NLP techniques are this kind of domain-specific systems. NLP establishes patterns that are valid for a specific domain and for a particular task only. As soon as the topic changes, entirely new patterns need to be established. For instance, the verb “to place” within the domain of terrorist activities is always linked with bombs. Making this assumption outside this topic would lead to trouble. To place the ball on the ground, the vase on the table, or the person in the job is unrelated to terrorism.

WHISK[38] is a rather typical NLP based extraction system using regular expressions to extract structured information from free text and fill manual-designed slots. It can also perform multi-slot extraction, transforming a semi-structured text into a table of relational database tuples. Like all systems that learn extraction rules for free text, WHISK must operate in conjunction with a syntactic analyzer and a semantic tagger. For the IE problem that WHISK addresses, it is not possible to tabulate how often a term is associated with correct extractions because of a lack of path expressions. Note that path expressions are important in extracting data from semi-structured document because hierarchical navigation between nested HTML elements is frequently needed.

No NLP software system can claim to tackle general language in an open-ended task. This is the chief problem of all practical natural language processing systems. NLP is effective only in a narrowly restricted domain. Unrestricted natural language processing is still a long way from being implemented, yet IE methods are effective because they rely on topical restrictions.

Although wrapper induction is an important and ever-popular methodology in the IE area, there are some reasons that prevent us from whole-heartedly embracing it. In a wrapper induction system, training documents build patterns that describe relevant information,

and patterns are run through these systems to extract information from other similar documents. We use the term “similar” in a very empirical sense, meaning pages provided by a same site or web application. In this context, wrapper induction systems are domain-specific. Even the founder of wrapper induction, Kushmerick, has accepted that the wrapper induction task has been constructed rather narrowly. Therefore, scalability in terms of web sites or domains is a major concern in wrapper induction systems and any change in web source can invalidate a working wrapper.

In this thesis, we are interested in some domain-independent methodology that can lead to some general-purpose IE system for arbitrary internet sites. In contrast to NLP and wrapper induction, HTML-aware modelling systems operate independently of specific domain knowledge. Instead of analyzing the meaning of discourse at the sentence level, this approach identifies relevant content based on the structural properties that surround desired data. Many information resources on the Web do not exhibit the rich grammatical structure that NLP or wrapper induction was designed to exploit. Instead of isolating the words, this technology looks for patterns that exist in the hierarchical formats of a document and uses this information to determine the meaning of a text.

HTML-CM[29] belongs to the domain-independent wrapper category that builds on the specific HTML tags (e.g. headings and tables) and how they are used in data formats. Heuristics are used to determine the parent-child relationships between data items, for instance table names, field names, and values. The resulting wrappers depend on the nesting and orientation of the table and other elements, which works well with tabular web sites but not with sites that have less structure. Rules in HTML-CM are static; it is not suited for dynamic web pages where layout styles are defined on creation.

The WYSIWYG Wrapper Factory (W4F) [4] uses an SQL-like query language called HEL, but provides little ease-of-handling. The user must be expert in the use of HEL and HTML to develop a wrapper. The toolkit transforms the HTML pages into a parse tree

according to the HTML specification. Instead of considering the HTML page as a string; it is viewed as a tree where each node corresponds to an HTML element. The advantage is that the structural information in the document can be used in the extraction rules, instead of just character-level patterns. But this approach relies heavily on the structural information in the web pages and on some pattern matching. It requires the use of absolute HTML paths that point to the data item to be extracted. For instance, an absolute path to the third table, first row, and second column in an HTML document could be expressed in path express as “/html/body/table3/tr1/td2”. The absolute path approach is likely to fail when the target HTML page changes. The most common change in HTML design is to alter the position of items on the page. For this reason, it is important to establish the location of data items independently of their absolute paths. Our approach involves finding semantic logic within the page itself that serves as a starting point for data extraction. Ideally, data are extracted based on the hierarchical layout of data items, not on the absolute HTML path. For instance, a page that contains the price of a book probably has the layout logic “Price:” ahead the price value. By looking for the layout structure “Price:” we can establish a general template for the price value extraction and be independent of its absolute location.

Chapter 4

Models and System Overview

Our approach for web information extraction takes full advantage of the structural clues indicated by authors in HTML markup tags. To achieve this, we employ the W3C's Document Object Model (DOM), which provides a uniform way of accessing and manipulating both plain text and markup tags in an HTML document. This Chapter is organized in the following sections: Section 4.1 introduces the W3C's DOM Core, and its relationship with the HTML DOM and XML DOM. Section 4.2 provides the necessary definitions and functions to build a Transitional DOM that can significantly simplify the conversion from HTML to XML. Section 4.3 outlines our web information extraction system that benefits from this Transitional DOM.

4.1 W3C Document Object Model

Today web pages are created either in web applications or in stand alone text editors. Browsers render this information but do not encourage users to edit what they see. This is where the DOM specification is useful. Developed at W3C by the Document Object Model

Working Group¹ (DOM WG), DOM defines a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents².

The DOM WG released its work in several specifications³: Core, HTML and XML. The Core DOM [41] provides a low-level interface for accessing and manipulating the content of documents. Depending on how it is used, the Core DOM interface may not be optimal for all documents. The HTML DOM [40] and XML DOM [45] specifications provide additional, higher-level interfaces that offer more convenient operations for HTML/XML documents.

According to the DOM specifications, the HTML/XML document is defined as an ordered tree of element nodes, to which attribute and text nodes are attached. While element nodes are the structure of the document, their attributes and texts store the actual data in the form of character strings. This tree can be traversed through its nodes and edges using an object interface. A fundamental characteristic of DOM is that it only defines object interfaces, not the actual implementation. This makes the API very language-neutral. It is up to third-party software developers to implement the API in the environment of their choice.

Example 4.1.1. The following is the HTML source from the web page Yahoo⁴ displayed in Figure 4.1.

```
...
<table cellpadding="2" cellspacing="0" border="0" width="95%" >
  <tr>
    <td nowrap="nowrap" valign="top" width="48%" >
      <b>
        <a href="http://www.yahoo.com/r/bu" >Business & Economy</a>
      </b>
    </td>
  </tr>
</table>
```

¹<http://www.w3.org/DOM/Group/>

²W3C's Recommendation for DOM, <http://www.w3.org/DOM/>

³<http://www.w3.org/DOM/Activity.html>

⁴<http://www.yahoo.com>

Figure 4.1: A partial view of Yahoo's homepage

```

<br />
<a href="http://www.yahoo.com/r/bb" >B2B</a>,
<a href="http://www.yahoo.com/r/fi" >Finance</a>,
<a href="http://www.yahoo.com/r/bs" >Shopping</a>,
<a href="http://www.yahoo.com/r/jo" >Jobs</a>...
<br />
<br />
<b>
  <a href="http://www.yahoo.com/r/ci" >Computers & Internet</a>
</b>
<br />
<a href="http://www.yahoo.com/r/in" >Internet</a>,
<a href="http://www.yahoo.com/r/ww" >WWW</a>,
<a href="http://www.yahoo.com/r/sf" >Software</a>,
<a href="http://www.yahoo.com/r/ga" >Games</a>...
<br />
<br />
...

```

From the HTML source, we can see that the HTML document consists of plain text (such as *Business&Economy* and *Software*), markup tags (such as `<table>`, `<a>` and `<br/>`) and attributes (such as *valign*, *width* and *href*). An HTML DOM tree for the

Figure 4.2: A partial view of HTML DOM tree for example 4.1.1.

above web document can be represented partially as shown in Figure 4.2. We can see that an HTML document has a fixed root element `<html>` with two child nodes `<head>` and `<body>`. In HTML DOM tree, each node is labelled with either an HTML tag name or a string of text in the content of the web page. The plain text of the HTML source is represented by text nodes, the markup tags are represented by element nodes and their attributes are represented by attribute nodes⁵. All the text nodes and attribute nodes are on the bottom of the tree and are called “leaf nodes”, while the element nodes can be either “leaf nodes” or “structural nodes”.

Since the tree is ordered, we can also mark the position of the nodes, such as *table4* and *a1*. This position information is useful for the orderly traversal of all nodes in the document, and for constructing the key components in the *path* expression we will address in the following section.

⁵In order to make the DOM tree look uncluttered, we hide all the attribute nodes in the tree.

Figure 4.3: A partial view of XML DOM tree for example 4.1.1

Example 4.1.2. If we convert the above HTML source in Example 4.1.1 to an XML document, we have the partial source as follows:

```

...
<Business_Economy url="http://www.yahoo.com/r/bu" >
  <B2B url="http://www.yahoo.com/r/bb" >
  <Finance url="http://www.yahoo.com/r/fi" >
  <Shopping url="http://www.yahoo.com/r/bs" >
  <Jobs url="http://www.yahoo.com/r/jo" >
</Business_Economy>
<Computers_Internet url="http://www.yahoo.com/r/ci" >
  <Internet url="http://www.yahoo.com/r/in"/>
  <WWW url="http://www.yahoo.com/r/ww"/>
  <Software url="http://www.yahoo.com/r/sf"/>
  <Games url="http://www.yahoo.com/r/ga"/>
</Computers_Internet>

```

W3C's DOM tree applies to HTML as well as to XML documents. Figure 4.3 displays an example of an XML DOM tree. Compared with the DOM trees in Figure 4.2 and Figure 4.3, we can see that the XML DOM describes the structured data itself while the HTML DOM focuses on the representational logic and leaves all the actual data lying on the bottom of the tree.

Despite the fact that one is focused on data structure while the other is focused on data representation, XML and HTML have one common DOM core. From this point of view,

the conversion from HTML to XML is intuitive and as simple as node-to-node matching. However, the fact is that, due to the diversity of HTML representation and constantly changing Web sources, this matching process can be extremely complicated and difficult. For example, if we want to build an extraction pattern from an HTML node *td* in Figure 4.2 to an XML node *Computers_Internet* in Figure 4.3, we have to go through the following steps:

1. Select all the child nodes of *td* and store them into a NodeSet *ns*;
2. Skip the first *i* nodes of *ns* and select the node *b2*;
3. Select the child node of *b2* which is *a*;
4. Map the child text node of *a* to XML node *Computers_Internet*;
5. Skip the next node of *b2* and map the node *a5* to the child of XML node *Computers_Internet*;
6. Repeat step 5 until reaching the *j*-th node in *ns* which is "...".

Obviously, mapping the HTML nodes directly to XML nodes is difficult and inefficient. To ease this complicated conversion process, we build a middle-tier model, called Transitional DOM (T-DOM). It works like a well-organized bookshelf, where the HTML nodes are re-arranged so that the wrapper program can select and map them to XML nodes more easily.

4.2 Transitional Document Object Model

Since HTML DOM and XML DOM have the same core, our Transitional DOM is also built upon this core to meet the following objectives:

1. HTML DOM conforms with representational purposes and allows web information to remain unstructured. The Transitional DOM must extend the HTML DOM interface to reflect the actual data and simplify access.
2. The structural information that the HTML tags represent can be ambiguous in the absence of context. The Transitional DOM must enhance the hierarchical structure by exploiting their explicit or implicit relations.
3. The purpose of building Transitional DOM is to simplify the transformation of HTML DOM nodes into XML DOM nodes, so the model definitions must be kept as simple as possible while conforming with multi-specifications.

To address the Transitional DOM more specifically, we provide a compact mathematical formalization of the DOM with respect to the DOM specification in [41] and the DOM transform specification in [43].

4.2.1 Mathematical Formalization of the DOM

A document DOM is defined as an ordered tree $(\mathcal{V}, \mathcal{R})$ where $\mathcal{V}$ is a set of nodes corresponding with the basic components of the tree, and $\mathcal{R}$ is a set of edges represented by a binary relation $\mathcal{R}$ on $\mathcal{V}$, thus $\mathcal{R} \subseteq \mathcal{V} \times \mathcal{V}$. Usually, we denote the DOM tree of the document by $\mathcal{T}$, but when we talk about the document source, we use the notion $\mathcal{D}$.

The basic component of a DOM tree is a node $Node \in \mathcal{V}$ or a set of nodes $NodeSet \subseteq \mathcal{V}$. The total number of nodes in a set of nodes can be denoted as $|NodeSet|$, such that $|NodeSet| := 1$ means it is a singular set, and $|NodeSet| := 0$ means it is an empty set. Given a node $x \in \mathcal{V}$, if x is a vertex of a tree, then we denote this tree as $T(x)$.

4.2.1.1 Functions for Nodes

There are three basic functions to identify the basic properties of a node:

1. $nodetype : Node \rightarrow Type$

Each node is one of six types, *root*, *element*, *attribute*, *text*, *comment* and *pi*⁶.

This function indicates the type of a node in the DOM core.

2. $label : Node \rightarrow String$

Each node has a name label beside it in a DOM tree. This function returns a string for the *qualified name* of a node. If the node is an element, it returns the tag name; if the node is an attribute node, it returns the attribute name; if the node is a text node, it returns an empty string.

3. $value : Node \rightarrow String$

Each node has a string value. This function returns the text value of a node. If the node is a text node, this text value is its content; if the node is an attribute node, it returns the attribute value; if the node is an element, the text value is a concatenation operation of the value of all the text nodes in a tree.

The root node is the vertex of the DOM tree for the entire document, and the HTML DOM tree has a concrete root labelled as *html*. In this thesis, we use node labels to represent the nodes in a DOM tree. For instance, in Figure 4.2, we have nodes labelled with *table*, *b* and *Jobs*. This representation differs from the document source representation, such as `<table>` and `<b>`. In case the labels of nodes are the same string in a DOM tree, we use a parentheses including an integer representing its position so as to distinguish the different nodes. To represent a value of a node, we use a quote like “Jobs”. Note that the element node has the concatenated text value, which can be a very long string; for example,

⁶In DOM specification, “pi” is the abbreviation of “processing instruction”.

$value(td)$ in Figure 4.2 has the value “Business_Economy B2B Finance Shopping ...”. On the other hand, the concatenated text value can also be empty, such as the node *br*.

For node types, we will make extensive use of the notations for the element nodes $\mathcal{E} \subseteq \mathcal{V}$, attribute nodes $\mathcal{A} \subseteq \mathcal{V}$ and text nodes $\mathcal{S} \subseteq \mathcal{V}$ to express our approach to web information extraction. Both the *comment* and *processing instruction* nodes are ignored in the Transitional DOM because they have no meaning relevant to the IE task.

There is much uncertainty in DOM specifications about when it is appropriate to store information in attributes. Every XML textbook explains somewhere within the first twenty pages that $\langle a \ b="dog" \ c="NN"/ \rangle$ and $\langle a \rangle \langle b \rangle \text{ dog } \langle /b \rangle \langle c \rangle \text{ NN } \langle /c \rangle \langle /a \rangle$ are (almost) equivalent representations of the same data. The accompanying examples often mix attributes and text nodes in a highly unsystematic way.

Since the HTML specification employs most of the attributes for storing the layout or event information, it is more appropriate to treat these as properties of elements rather than as attribute nodes in our Transitional DOM. For this reason, we define an extension function $attributes : \mathcal{E} \rightarrow \mathcal{A}$ to gather the attribute information of an element node. The result of the function is a sequence of attribute nodes in a tuple $\langle \alpha_1, \alpha_2, \dots, \alpha_k \rangle$ where $\alpha_1, \alpha_2, \dots, \alpha_k \in \mathcal{A}$.

For example, in Example 4.1.1, the attributes of the element node *table* are $attributes(table) = \langle cellpadding, cellspacing, border, width \rangle$. In this thesis, the symbols $\{ \}$ and $\langle \rangle$ are subscripted to help us refer to the corresponding *sets* and *tuples*.

4.2.1.2 Functions for Relations

There are two basic relations that are instances of DOM relations $\mathcal{R}$. They are dominant relations with respect to a direct edge from *parent* node to *child* node in a tree graph. We define these relations in two functions:

1. $children : Node \rightarrow NodeSet$
2. $parent : Node \rightarrow NodeSet$.

In Figure 4.3, we find some instances for these basic relations such as $parent(Jobs) = \{Business_Economy\}$ and $children(Business_Economy) = \{B2B, Finance, Shopping, Jobs\}$. Note that a function may return a *NodeSet* with *null* value, for example, $parent(html) = null$ and $children(Jobs) = null$. First result means the node *html* is the root node, and second result means the node *Jobs* is a leaf node. In fact, none of the attribute nodes and text nodes in a DOM tree have *child* nodes since they are always on the bottom of a tree.

From these basic relations, we have extended functions as follows:

1. $sibling : Node \rightarrow NodeSet$

This extension function is a peer-to-peer relation with respect to nodes that have one common *parent* in a tree graph. For a node x , the sibling nodes are $sibling(x) = \{z | y \in parent(x), z \in children(y)\}$.

2. $path : Node \rightarrow String$

This extension function of the dominant relations is to accurately represent the position of a node in the DOM tree. From the DOM root, following the edges down to the current node, we gain a set of connected nodes with dominant relations. The result of the path function is a string concatenated with “/” and the labels of these nodes. The *path* of a node x_k can be obtained inductively:

$$\begin{aligned}
 path(X_1) &= " X_1'', & nodetype(X_1) &= root \\
 path(X_2) &= " path(X_1)/X_2'', & X_1 &= parent(X_2) \\
 &\dots \\
 path(x_k) &= " path(X_{k-1})/X_k'', & X_{k-1} &= parent(X_k)
 \end{aligned}$$

The number k is also denoted as the *depth* of a path.

3. $level : Node \rightarrow Integer$

Another peer-to-peer relation is with respect to nodes which may not have one common *parent* node, but are on the same level in a tree graph; in other words, they have the same depth in the path. We define the root of the DOM tree as level 0, and use *level* function to return an integer indicating the level of a node. If node x has a parent node y , then the un-equation $level(x) > level(y)$ is true. The height of a DOM tree is the maximum *level* value of the nodes in the tree.

To select a node in the DOM tree, we usually process the depth first, navigating breadth left to right. In Figure 4.3, if we want to select the node *Jobs*, we may follow the path from the root *doc* and reach the node *B2B* first in a path *doc/directory/Business_Economy/B2B*, then navigate in a set of siblings $sibling(B2B) = \{Finance, Shopping, Jobs\}$ from left to right and finally reach the node *Jobs*. The nodes *Jobs* and *Games* have different parent nodes, but they are on the same level, namely $level(Jobs) = level(Games) = 4$.

The extended functions we give are, unsurprisingly, quite similar to XSLT [43], a standard pattern language for transforming DOM trees. However, the XSLT specification is considerably more complex. The semantics given here seem more appropriate for the purpose of building the Transitional DOM.

4.2.2 Atomic Units of Web Information

The task of information extraction is the focused searching for *meaning* in the target documents, where meaning is understood in terms of *facts*, formally described as a fixed set of lexical objects [2]. In the context of HTML documents, these lexical objects are atomic units of web information appearing in a specific region of the browser's display window. In order to detect and access these information units in our Transitional DOM, we introduce the definition of *record* and *delimiter* which are both extended from the HTML nodes.

Definition 4.2.1. A *record* is an HTML node which has a meaningful text value. All the records in an HTML DOM are denoted as $\mathbb{R}$. A *delimiter* is an HTML node which does not have a meaningful text value. All the delimiters in an HTML DOM are denoted as $\mathbb{D}$.

Remark 4.2.1. We give the following common notes for the Definition 4.2.1:

1. For languages that are based on ASCII, ISO 8859 or EUC standards, like English, the meaningful text value is a string containing any characters from $[0\dots9][a\dots z][A\dots Z]$. For other human languages, the definition of a meaningful text value may vary, but the basic criteria is that every record must carry a fact of web information.
2. The definition of record is upward transitive in a dominant relation, such that, if a node is a record, then its parent node is also a record. In contrast, the definition of delimiter is downward transitive in a dominance relation, such that if a node is a delimiter, all its child nodes are delimiters. In other words, if all the child nodes are delimiters, then this node is also a delimiter.
3. Most attributes in HTML, such as *border*, *width*, *onClick* and *onLoad*, are layout or event properties that contain no facts of web information to extract. So, we assert that attribute nodes, except *href* and *alt*, are not considered as records even if they have a meaningful text value.
4. Both records and delimiters are extended from the interface of the HTML DOM and inherit all the properties of DOM nodes, such as *label*, *value* and *level*.

As shown in Figure 4.2, element nodes *b1*, *a1* and *a4* are records, since they have text values “Business & Economy”, “B2B” and “Jobs” respectively. Aside from text nodes, a text value could also be obtained from an attribute node *alt* or *href*. An *img* node, with HTML source `<img alt=“Yahoo! Travel” src=“ytrv_prp.gif” />`, is a record because *alt* has a meaningful text value “Yahoo! Travel”. The element node *br* and the text nodes “,” and “...” are delimiters since they have no meaningful text value.

4.2.3 Hierarchical Relations of Web Data

A single record in the HTML DOM, just like a single word in an article, still lacks meaning without context. To explore the hierarchical relations between records, we need to group the consistent records into content blocks first, and then find the topics in each content block.

Definition 4.2.2. The notions of *content blocks* and *topics* are defined as follows:

1. A *content block*, denoted as ψ , is a set of adjacent records on the same level in the DOM tree.
2. A *topic node* (or *topic*), denoted as τ , is a record which has the property of being the subject of other records in the same content block.

Remark 4.2.2. We give the following common notes for the Definition 4.2.2:

1. The *topic* is extended from the interface of *record* and inherits all the properties of DOM nodes, such as *label*, *value* and *level*.
2. A content block should not be confused with the basic component *NodeSet* in a DOM tree. There are three essential differences. First, the *NodeSet* is a set of nodes of all types, including records and delimiters, while in a content block, all nodes are records. Second, nodes in a *NodeSet* can be on different levels in the tree, and they may not be adjacent, while in a content block, all nodes must be adjacent and on the same level. Last, nodes in a *NodeSet* can have different parent nodes in the HTML DOM tree, while all of records in a content block must have the same parent. For this property, we can denote it by a sub-tree in Transitional DOM tree, such as $T(x)$, where x is the common parent of all records in the content block.

Figure 4.4: The consistent records are grouped into content blocks

3. If $x_1, \dots, x_n$ are adjacent records on the same level in the DOM tree, a set $\{x_1, \dots, x_n\}$ is a content block constructed from records $x_1, \dots, x_n$ using a set constructor of order n .
4. If a content block ψ is $\{x_1, \dots, x_n\}$, a tuple $\langle \tau_1, \dots, \tau_k \rangle$, where $(\tau_1, \dots, \tau_k) \in \psi$, is a list of *topics* constructed from records $x_1, \dots, x_n$ using a tuple constructor of order k .
5. All topics come from the records of content blocks. We give a heuristic that every content block in the HTML DOM contains a *default* topic. In case no explicit topics can be found, we can add a certain record into the content block to be its default topic.

As shown in Figure 4.4, a set of consistent records $\{b1, a1, a2, a3, a4, b2, a5, \dots\}$ is grouped together as a content block and all delimiters are eliminated. By analyzing the hierarchical relations among these records, we obtain a tuple $\langle b1, b2, \dots \rangle$ where each record in this tuple is a topic with text values of “Business & Economy”, “Computers & Internet” and so on.

Topics are obvious on a browser’s window as shown in Figure 4.1. This is because the author of the web document uses many specific HTML tags and structures to *guide* a

Figure 4.5: Transforming functions for building the Transitional DOM

browser in emphasizing these topics so that they look different from other representations. A wrapper program can use a similar method to abstract the subject of other records in a content block.

4.3 System Overview

Our web information extraction system is designed to convert HTML documents to XML documents automatically. This task is accomplished in two stages. In the first stage, the HTML DOM is transformed into a Transitional DOM. In the second stage, extraction patterns are explored to map nodes from the Transitional DOM to the XML DOM.

Figure 4.5 demonstrates the first stage, which consists of a set of dynamic and incremental transformation processes that eventually abstract the hierarchical relations of data in the HTML DOM and build the Transitional DOM. These processes are denoted by *Node Classification*, *Content Partition* and *Topic Aggregation* and are described as follows:

1. In the process of *Node Classification*, we classify the HTML nodes by records and delimiters. This process is intuitive, and just follows the Definition 4.2.1. Each node in the HTML DOM tree either has a meaningful text value or does not, so the HTML DOM can be considered as containing only two types of nodes, where the records represent the actual data and the delimiters form their boundaries.

To classify the HTML nodes by *means* is helpful in addressing the diversity problem of HTML mentioned above. According to the definition of records, both `<b><a> ... </a></b>` and `<a><b> ... </b></a>` are treated as the same record in our Transitional DOM, so that the conversion rule from the Transitional DOM to the XML DOM is identical.

2. In the process of *Content Partition*, we group the consistent records into content blocks. This process takes advantage of the delimiters, which can indicate explicit or implicit boundaries for content blocks. In Chapter 5, we discuss in detail the methods and algorithms for content partition. As shown in Figure 4.4, when we eliminate the delimiters, the HTML DOM is simplified and this new DOM is ready for the next process.
3. In the process of *Topic Aggregation*, the topics are abstracted from these content blocks and the Transitional DOM is built. We analyse the properties of each record in the content block so as to find the subjects of other records, which are topics. This process aims to simplify the HTML DOM tree by bringing the hierarchical relations from the node level to the record level. More details on these strategies are given in Chapter 6.

Since the XML DOM organizes the hierarchical data in the dominant relations of the tree, every node is a topic of its child nodes. In the HTML DOM, this rule is exactly inverse, such that none of the dominance relations hold the hierarchies of records, while all the topics in a content block are lying on the same level with other records. We need a final transformation process for accomplishing the task. Figure 4.6 shows the second stage of web information extraction that builds the mapping patterns from the topics of the Transitional DOM to the topics of the XML DOM. This process is called *Semantic Annotation* which is discussed in detail in Chapter 7 Section 7.1.

Figure 4.6: Mapping the Transitional DOM to the XML DOM.

Recall the same task we discussed in the above section, the direct mapping of the HTML DOM text node *Computers&Internet* to the XML DOM node *Computers_Internet*. When mapping from the Transitional DOM to the XML DOM, we can easily match the content block $T(td2)$ containing the topic node $b1$ of the Transitional DOM to the node *Computers_Internet* of the XML DOM.

Our system abstract model is shown in Figure 4.7. The core of the system consists of three modules, *Separator*, *Conductor* and *Extractor*, which correspond with the different extraction steps. It also shows that the output of one process is the input of the next process; thus, an atomic information unit from the bottom of the HTML DOM tree is eventually mapped to the structured XML DOM.

Figure 4.8 gives an overall demonstration of how the system works for an original web document as shown in 4.8 (a). Figure 4.8 (b) is the result after content partition by separator, where the separator detects the content blocks with explicit or implicit boundaries. Figure

Figure 4.7: The abstract model of the automatic Web IE system

4.8 (c) is the result after topic aggregation, where the hierarchical relations are conducted by topics. Finally, the extractor builds the XML document as shown in Figure 4.8 (d).

(a) Original document

(b) After content partition

(c) After topic aggregation

(d) Converted XML document

Figure 4.8: The transition from HTML to XML

Chapter 5

Content Partition

An IE system must contend with co-reference recognition. Co-reference recognition determines when an expression, such as the pronouns “he”, “she”, or “it” and noun phrases like “the company”, refer to the same thing in a sentence. For IE to work correctly, various entities within documents (locations, people, places, events) must be identified within the boundary of a content block.

Observation 5.0.1. Whether it is manually composed or generated automatically by an application, a web page commonly keeps consistent information as discerned in the same content block, and different topics are separated by apparent visual boundaries.

Observation 5.0.1 explains what the *layout* of a web page means. The purpose of content partition is to recover these information blocks based on the clues that authors embed in the layout information. Figure 5.1 shows how the process of identifying these visual cues is performed in an iterative way. It starts with the whole web page as a single content block (made up of smaller content blocks), and then iteratively partitions each content block into smaller ones until the process reaches the final solution. At each iteration, the page analysis algorithm finds the best way to partition a content block. Note that instead of mining data

records directly, which is difficult, our method mines data regions first and then finds the data records within them.

Figure 5.1: Content Blocks Identification

We separate the process of content partition into two stages. Section 5.1 presents the first stage of this process using a novel method to identify the *high-level* content blocks in a page. Section 5.2 describes the second stage, which uses explicit and implicit boundaries to refine high-level content blocks into more logically associated content blocks.

5.1 High-level Content Blocks

When people create a web page, they in general either use a template or have a layout in mind to guide the arrangement of their information. For example, they may consider whether to put a header, footer, or side bar in a page, and decide how many distinct topics will exist in the main body block. We call these headers, footers, side bars and main content bodies the *high-level content blocks*. We suppose that every page can be divided into five high-level content blocks denoted as $(\psi_{head}, \psi_{body}, \psi_{foot}, \psi_{leftbar}, \psi_{rightbar})$. Sometimes we talk about the main body block, which refers to a group of three high-level content blocks $(\psi_{leftbar}, \psi_{body}, \psi_{rightbar})$. Note that not every web page has all the above high-level content blocks, but it will have at least one body block ψ_{body} .

Observation 5.1.1. Web pages from the same web site usually use the same headers, footers and side bars for stylization and convenience.

There are significant redundancies in web page contents. For instance, almost all dot-com web sites present their company's icons, service channels, navigation panels, copyright and privacy announcements, and advertisements in every page for business purposes with easy and user-friendly access. It is convenient for users to easily navigate related services but it is a significant challenge for web miners, since they must process the entire contents of a page.

According to our Observation 5.1.1, it is more effective for web information extraction systems if, after we have parsed the first page of the site and identified its high-level content blocks, we can focus our extraction tasks only on the main body blocks of the rest of the pages in the same site. To identify these high-level content sub-trees is crucial to the control of the whole extraction procedure.

5.1.1 Find the Main Body Blocks

Finding the body block in a web page is the key step in the high-level content partition process. Suppose we write a paper or thesis. Usually, we use more chapters and sections to address the details of our solution. As for the introduction or conclusion sections, there are obviously fewer chapters and sections. Similarly, web-page authors put most important information in the body block, since it is the centre of the page and is more attractive to readers.

Heuristic 5.1.1. Based on Observation 5.1.1, if a web page has a header or footer, the distribution of records will create a big *hop* between the header and the main body, or between the footer and the main body. This hop can be used to locate the main body block.

Web Page	Header	Body	Footer
www.yahoo.com	13	196	87
www.microsoft.com	7	106	7
www.scs.carleton.ca/people/faculty	8	395	2
www.carleton.ca	9	58	2
www.harvard.edu/academics/	16	65	9
www.hpl.hp.com/personal/Zhichen_Xu/index.html	15	210	4
jupiter.scs.carleton.ca	2	19	1
www.amazon.com	6	200	61
www.apple.com/software/	21	143	13
www.gsfc.nasa.gov	1	48	15
Average:	9.8	144	20.1

Table 5.1: Distributions of records in high-level content blocks.

Once the main body is detected, the content part of forward hop is the header, and the content part of the backward hop is the footer.

Table 5.1 lists the record distribution in the headers, bodies and footers of 5 web pages. The average number of records in the three high-level content blocks are 9.8 (header), 144 (body) and 20.1 (footer). It shows that the percentage of records in the body block is 82.8%, much higher than those in the header and footer blocks, which are 5.6% and 11.6% respectively. Significant distribution hops exist between the header and body, as well as between the body and footer blocks, which prove the Heuristics 5.1.1.

5.1.1.1 Node Density Formula (NDF)

The purpose of NDF is to calculate the record density in the tree of the first-level element nodes (direct children of *body* node in HTML DOM) so as to detect the position of the hop as indicated in the Heuristics 5.1.1. For the first-level element nodes $\{\varepsilon_i | \varepsilon_i \in \mathcal{E}, 1 \leq i \leq k, \text{level}(\varepsilon_i) = 1\}$, there are a set of trees $\{T(\varepsilon_1), \dots, T(\varepsilon_k)\}$ with the first-level element nodes as vertices. Number k is the total number of the first-level elements in the HTML

DOM. The density of the records in $T(\varepsilon_i)$ can be calculated using Formula 5.1.

$$D_i = \frac{n_i}{\sum_{i=1}^k n_i} \quad (5.1)$$

where n_i is the number of records distributed in $T(\varepsilon_i)$.

In order to detect the hops, we calculate the *Harmonic Mean* [36] $\bar{H}$ of the density changes between these first-level elements in Formula 5.2.

$$\bar{H} = \frac{k-1}{\sum_{i=2}^k \left| \frac{1}{\Delta D_i} \right|} \quad (k > 2) \quad (5.2)$$

When the two adjacent first-level elements ε_i and ε_j have the records density change $\Delta D_{ij} = |D_j - D_i|$ over the threshold $\bar{H} + D_1$, then there is a distribution hop between element ε_i and ε_j . We consider that there are more than two first-level elements, since otherwise the entire HTML contents will be treated as one body block. The default sense of $\bar{H}$ is to minimize the threshold value if the change rate of record densities is low.

Example 5.1.1. Figure 5.2 presents an example of node density and hop calculation for a web page from Carleton University ¹. Both the record number and the node density are counted for every first-level element.

Using the Formula 5.2, we discover that the harmonic mean of the record density changes is 0.241 and the threshold value is 0.255. Based on the threshold value, we find that a hop occurs between the second `<table>` and the third `<table>`. According to the Heuristic 5.1.1, the third `<table>` can be distinguished as main body block ψ_{body} . Then, all of *table* nodes before this one belong to the header block ψ_{head} and the last *table* node is treated as the footer block ψ_{foot} , as shown in Figure 5.3.

¹<http://www.carleton.ca/>

```

<html>
+ <head>
  <body>
    +<table records.number="1" density="0.014" />
    +<table records.number="8" density="0.116" />
    +<table records.number="58" density="0.841" />
    +<table records.number="2" density="0.029" />
  </body>
</html>

```

Figure 5.2: Node Density, calculation of hop detection for Carleton's homepage

5.1.2 Find the Side Bars

In most cases, the main body block is a *table* node. According to the HTML specification, `<table>` tags can be used to optimize the displaying space of data on a web page, to make the alignment of consistent information tidier, and also to make it easy to differentiate them from each other.

Observation 5.1.2. The display region of a side bar in a web page will for the most part not exceed the 25% width of the total display area of the main body.

Heuristic 5.1.2. If there are side bars in a main body block, there must be a *table* node that has only one child *tr*, while *tr* has at least two child nodes *td*. The side bars can then be identified by the following characteristics:

- The left most record is the left-bar block if its width is less than 25% of the table width. Otherwise, it is part of the body block.
- The right most record is the right-bar block if its width is less than 25% of the table width. Otherwise, it is part of the body block.

Normally, there are *width* attributes for the nodes *table* and *td*. The sum of the width of the *td* nodes must not be greater than the *width* of the *table* node. The value of *width* is either an absolute value, such as 770, or a relative value, such as 30%. If no *width* attribute is specified for the *table*, then a default width value 800 or 100% is generated. By default, all *td* nodes have the same width. Not all *td* nodes are records, so the heuristic works for the *td* nodes that are records.

Figure 5.3: The high-level content blocks in Carleton's homepage

Based on the Heuristics 5.1.2, we can distinguish the side bars from the main body block of a web page as shown in Figure 5.3. The left-bar block $\psi_{leftbar}$ is the left most record object which has the *width* attribute with the specific value of 163, less then the 25% of the total table width 770.

5.2 Detect the Block Boundaries

Web information in an HTML document is well displayed; relevant data are gathered into a distinguishable region separated by boundaries, such as frames and white space. There are two kinds of boundaries: *explicit boundaries* and *implicit boundaries*.

5.2.1 Explicit Boundaries

In HTML, tags are either *inline* or *block-level* elements. Inline tags (elements) become a part of the *flow* of text in which they are inserted, have no *box* around them and do not have the carriage return either. Inline tags simply embed something on the spot, such as an image tag `<IMG/>` or horizontal line `<HR/>`. Block-level elements exist in their own virtual *box* and are always followed by a carriage return (like hitting the *enter* key after typing in some text).

Most HTML tags are block-level tags, meaning they can enclose other HTML tags or text, or in the view of the DOM, they can have child nodes. They are good indicators for detecting the boundaries of content blocks.

Example 5.2.1. Figure 5.4 displays some explicit boundaries which are a list of `<tr>` tags. The HTML source could be:

```
<tr>
  <td>
    <b>Shop</b>
  </td>
  <td colspan="2">
    <a href="http://www.yahoo.com/r/a2">Auctions</a>
    <a href="http://www.yahoo.com/r/cr">Autos</a>
    <a href="http://www.yahoo.com/r/cf">Classifies</a>
    <a href="http://www.yahoo.com/r/r1">Real Estate</a>
    <a href="http://www.yahoo.com/r/sh">Shopping</a>
    <a href="http://www.yahoo.com/r/ta">Travel</a>
```

```

</td>
</tr>
<tr>
  <td>
    <b>Find</b>
  </td>
  <td colspan="2" >
    <a href="http://www.yahoo.com/r/jb" >HotJobs</a>
    <a href="http://www.yahoo.com/r/mp" >Maps</a>
    <a href="http://www.yahoo.com/r/ps" >People Search</a>
    <a href="http://www.yahoo.com/r/pr" >
      <b>Personals</b>
    </a>
    <a href="http://www.yahoo.com/r/yp" >Yellow Pages</a>
  </td>
</tr>
...

```

Figure 5.4: Explicit boundaries detected by block-level tags

Figure 5.4 displays some explicit boundaries which are a list of `<tr>` tags. Every `<tr>`

record has a set of child nodes *td1* and *td2*. In a DOM structure, selecting the explicit boundary node will select the entire set of list records as a content block.

5.2.2 Implicit Boundaries

Unlike explicit boundaries, which are indicated by specific HTML tags, implicit boundaries are identified by the delimiters. In a set of HTML nodes, delimiters separate both the records and their blocks. Implicit boundaries are those delimiters that separate the content blocks.

Definition 5.2.1. In a set of nodes $\{x_1, \dots, x_n\}$ with the same level, an implicit boundary is a *maximal* set of adjacent delimiters $\{x_i, x_{i+1}, \dots, x_{i+j}\}$, that $x_i \in \mathbb{D}, x_j \in \mathbb{D}, 1 < i < j < n$ with $|\{x_i\}| = \max(j - i)$.

As defined in Definition 5.2.1, implicit boundaries are on the same level as the records in the HTML DOM tree, and separate the records into content blocks by using the maximum number of delimiters.

Detecting implicit boundaries is not as direct as with explicit boundaries, because delimiters can be used to separate two records or two content blocks. If we treat every delimiter as a *gap* between the records, then the implicit boundary is the biggest gap between two groups of records.

This gap is relative: as shown in Figure 5.5, one `<br/>` tag creates a smaller space, but double `<br/>` tags create a bigger space between content blocks. The differences in gap spaces between blocks make the boundaries obvious to human eyes in separating content blocks, but implicit for applications because they cannot be directly detected by just looking for the HTML tags.

Figure 5.5: Detecting implicit boundaries by projection algorithm

Algorithm 1 Axis Projection

Input: A block ψ has a set of nodes with m records and n delimiters, where $m + n = k$.

Output: A set of sub-blocks separated by implicit boundaries.

- 1: Find all records $\lambda_1, \dots, \lambda_k$ and delimiters $\nu_1, \dots, \nu_n$.
- 2: Projecting records set $\lambda_1, \dots, \lambda_k$ and delimiters set $\nu_1, \dots, \nu_n$ into block ψ , so that the new block is $\lambda_1, \nu_1, \dots, \lambda_i, \dots, \nu_j, \dots, \nu_n, \lambda_k$
- 3: **while** every λ_i has an adjacent ν_{i+1} with $1 \leq i \leq k$ **do**
- 4: reduce the size of block set ψ by eliminate object ν_{i+1}
- 5: let $k \leftarrow k - x$ where x is the number of eliminated delimiters
- 6: **end while**
- 7: **while** λ_i has an adjacent λ_{i+1} with $1 \leq i \leq k$ **do**
- 8: extract the adjacent objects $\lambda_i, \lambda_{i+1}, \dots, \lambda_{i+x}$ and construct a new block set ψ_i
- 9: let $k \leftarrow k - x$ where x is the number of extracted records
- 10: **end while**

To solve this implicit boundary detection problem, we develop an *axis projection* algorithm that can discern the record delimiters from the block delimiters. The axis projection algorithm can be performed in $O(n + m)$, where n is the number of delimiters and m is the number of records. So, the total time complexity of axis projection is $O(n)$.

In Example 4.1.1, we give a sample block that contains both records and delimiters. If we project all the delimiters into the block, we have the form $\{b1, \nu, a1, \nu, a2, \nu, a3, \nu, a4, \nu, \nu, \nu, b2, \dots\}$ where ν is a delimiter. After reducing the size of the block by eliminating delimiters ν_{i+1} , we gain a new block $\{b1, a1, a2, a3, a4, \nu, \nu, b2, a5, \dots\}$. Splitting the block by the boundary ν, ν , we have sub-blocks $\{b1, a1, a2, a3, a4\}$ and $\{b2, a5, a6, a7, a8\}$ and so on.

Chapter 6

Topic Aggregation

Content partition is a top-down procedure, starting with the identification of the high-level content blocks, and ending with the separation of each set of consistent records (content block). After content partition, the data (record) and its layout (delimiter) are separated but the hierarchical structure of the data is still ambiguous. We develop a *Topic Aggregation* process which is a bottom-up procedure to cluster consistent records or blocks into specific topics so as to explore the hierarchical relations of the data in the HTML DOM.

In Chapter 4 Section 4.2, we defined the topics and showed in Figure 4.6 how the topic nodes match with an XML hierarchical structure. In this chapter, we propose a set of methodologies for clustering topics from the content blocks. In Section 6.1 we introduce the important characteristics of a topic node and its distribution space. Section 6.2 solves the topic aggregation problem in one-dimensional space via topic clustering strategies. Section 6.3 provides a novel algorithm for reducing the complex two-dimensional space to a set of one-dimensional spaces.

6.1 Preliminaries

Observation 6.1.1. Normally, a topic node conforms to the following two characteristics that allow it to compare its peer records in the same content block.

1. A topic node is a record with emphasis at a preceding position compared with its sibling records in the same block. The main strategies for emphatic purpose are
 - Using a distinguishing style, such as bold fonts.
 - Placing an indication symbol, like a colon “:”, to infer dominance.
 - Creating an indentation between primary and secondary.
2. Topic nodes have the most similar properties and a maximal frequency of occurrence in the same content block.

Example 6.1.1. An HTML block displayed as “**Tel:**, 2280256, **Fax:**, 2280255, ...” has the following HTML codes:

```
<p>
  <b>Tel: </b> 2280256 <b>Fax: </b> 2280255 ...
</p>
```

The bold style nodes (**Tel:** and **Fax:**) are topic nodes of the block $\{b, 2280256\}$ and block $\{b, 2280255\}$ respectively. Compared with its peer records in the same block, a topic node is a leading record with significant text value, and for emphatic purposes, they are commonly in bold style. Moreover, the repetition of a similar structure $\{b, s\}$ has a maximal frequency of occurrence in the content block, where $s \in \mathcal{S}$.

Theorem 6.1. *All the topic nodes in a content block are HTML DOM nodes on the same level of the tree.*

PROOF: According to the Definition 4.2.1, all the structured objects in a block are HTML

Figure 6.1: Topic nodes are on the same level in the DOM tree for the Example 5.2.1.

DOM nodes on the same level of the tree. In the Definition 4.2.2, a topic node is one of the records in the content block, so all the topic nodes in the block are HTML DOM nodes on the same level of the tree.

Detecting the topic nodes among a set of records could be very simple, as in Example 6.1.1, or more complex, as in Example 5.2.1. In that example, finding topic nodes like “Shop”, “Find” and “Connect” is not as simple as seeking the emphasized texts. For instance, records such as “Personals” and “Movies” in the content block are not topic nodes even though they are emphasized. The figure shows that, although the “Personals” record has the same emphasis as the record “Shop”, they are on different levels in the HTML DOM tree.

Hierarchical relations hidden inside HTML sources have multiple dimensions. The most common situation is one-dimensional space, which means all topic nodes of the same class have the same parent. For instance, in the Example 6.1.1, topic nodes `<b>Tel: </b>` and `<b>Fax: </b>` are the same class `<b>` and have the same parent `<p>`.

In the HTML specification, the description of one topic may intertwine with the descriptions of some other topics. For example, the descriptions of two topics in the HTML source may follow the sequence part1 of topic1, part1 of topic2, part2 of topic1, part2 of topic2. Thus, the descriptions of both topic1 and topic2 are not contiguous. However, when they are displayed on a Web browser, they appear contiguous to human viewers. This leads to another more complicated hierarchical relation hidden inside HTML DOM: two-dimensional space.

Example 6.1.2. Figure 6.2 shows a two-dimensional space represented by an HTML table.

Figure 6.2: A $m \times n$ table showed on Yahoo's homepage.

```
<table>
  <tr>
    <td bgcolor="efefff" colspan="2" >
      <b>Europe</b>
    </td>
    <td bgcolor="efefff" colspan="2" >
      <b>Asia Pacific</b>
    </td>
    <td bgcolor="efefff" colspan="2" >
      <b>Americas</b>
    </td>
```

Figure 6.3: Topic nodes in the two-dimensional space for the Example 6.1.2.

```

</tr>
<tr>
  <td width="16%" >
    <a href="http://www.yahoo.com/r/nc" >Catalan</a>
    <a href="http://www.yahoo.com/r/dk" >Denmark</a>
    <a href="http://www.yahoo.com/r/fr" >France</a>
  ...
</tr>
<tr>
  <td width="16%" >
    <a href="http://www.yahoo.com/r/ai" >Asia</a>
    <a href="http://www.yahoo.com/r/an" >Australia & NZ</a>
    <a href="http://www.yahoo.com/r/cc" >China</a>
  ...
</tr>
...
</table>

```

If we examine the HTML DOM tree as shown in Figure 6.3, we find that the topic nodes are in the first `<tr>` and all other records are in the remaining `<tr>` tags. If one topic node conducts a set of records, these records will on the lower lever than the topic node.

6.2 Extracting the Hierarchies in One-Dimensional Space

In one-dimensional space, all record nodes appear at the same level in the DOM tree. The key step in solving this one-dimensional space problem is to detect topic nodes from a content block. We conclude with two primary cases for topic node detection: *static clustering* and *dynamic clustering*.

6.2.1 Static Clustering

Based on the characteristics of topic nodes, if an emphasized record is followed by un- or less-emphasized records, then it is a topic node of these subsequent records. HTML specification provides specific tags as static rules for building hierarchical relations within web data. We organize these rules into two groups: headings and indentions.

6.2.1.1 Heading

HTML specification provides six levels of headings from $\langle H1 \rangle$ (the most important) to $\langle H6 \rangle$ (the least important). A heading element briefly describes the topic of the section it introduces. Visual browsers usually render more important headings in larger fonts than they do less important ones.

Rule 6.2.1. In a set of records $\{\lambda_i | \lambda_i \in \mathbb{R}, 1 \leq i \leq n\}$, if $label(\lambda_i) := hx$ and $label(\lambda_j) := hy$ with $(1 \leq i < j \leq n, 1 \leq x < y \leq 6)$, we can build a hierarchical structure $\{\lambda_i \Rightarrow \{\lambda_{i+1}, \dots, \lambda_{j-1}\}\}$ where $j \geq i + 1$.

Example 6.2.1. Figure 6.4 shows using HTML headings for describing the course syllabus.

In this example, we construct the hierarchical relations via the heading levels. Using

Figure 6.4: HTML Headings for Describing the Course Syllabus

Rule 6.2.1 repeatedly by $\langle H1 \rangle$, $\langle H2 \rangle$ and $\langle H3 \rangle$ in the content block displayed in Figure 6.4, we gain the following hierarchical structures:

```

{ "Course Syllabus"
  ⇒ "...description of course..."
  ⇒ { "Attendance Policy" ⇒ "...description of policy..." }
  ⇒ { "Semester Schedule"
    ⇒ { "Readings" ⇒ "...schedule info..." }
    ⇒ { "Lectures" ⇒ "...schedule info..." }
  }
  ⇒ { "Office Hours" ⇒ "...contact info..." }
}

```


6.2.1.2 Indention

An indention, used to indicate that one section is a subsection of another, can create a one-level hierarchy between two records. These kinds of hierarchies are static and can be discovered by HTML-specific tags. Some examples are an order list `<ol>`, un-order list `<ul>`, definition list `<dl>` and quotations `<blockquote>`.

Rule 6.2.2. For two records $\{\lambda_1, \lambda_2\}$ such that $\lambda_1, \lambda_2 \in \mathbb{R}$, if $label(\lambda_2) \in \{ol, ul, blockquote\}$, we can build a hierarchical structure $\{\lambda_1 \Rightarrow \lambda_2\}$.

Figure 6.5: Indentions created with HTML list and quotation tags

Example 6.2.2. Figure 6.5 shows using the HTML indention for one-level hierarchy.

In this example, we construct the hierarchical relations via the indention. The content block displayed in Figure 6.5 can be represented obviously as:

```

{ "Order list"  $\Rightarrow$  "...work environment..."
   $\Rightarrow$  "...benefits package..."
}
{ "Unorder list"  $\Rightarrow$  "...work environment..."
   $\Rightarrow$  "...benefits package..."
}
{ "Quotation"  $\Rightarrow$  "...work environment..."
   $\Rightarrow$  "...benefits package..."
}

```

Definition lists `<dl>` vary only slightly from other types of lists in that list items consist of two parts: a term and a description. The term is given by the `<dt>` element and is restricted to text content. The description is given with a `<dd>` element that contains block-level content. This representation is much like HTML headings.

Rule 6.2.3. For two records $\{\lambda_1, \lambda_2\}$ such that $\lambda_1, \lambda_2 \in \mathbb{R}$, if $label(\lambda_1) := dt$ and $label(\lambda_2) := dd$, we can build a hierarchical structure $\{\lambda_1 \Rightarrow \lambda_2\}$.

Figure 6.6: Indention created with HTML definition list tags

Example 6.2.3. Figure 6.6 shows using the HTML definition list for one-level indention.

In this example, we use Rule 6.2.3 to construct the hierarchical relations. The content block displayed in Figure 6.6 can be represented as:

```
{ "Dweeb" ⇒ "young person who may mature into a Nerd" }
{ "Hacker" ⇒ "a clever programmer" }
{ "Nerd" ⇒ "technically bright but socially inept person" }
```

6.2.2 Dynamic Clustering

Another kind of hierarchical structure is implicit, which cannot be detected by static tags. These kinds of topic nodes either use boldface or specific attributes which are defined dynamically for individual pages. The bold style text normally uses HTML tags `<b>` (Example 6.1.1) or `<th>` (Example 6.2.4). The most commonly used attributes are *class* (Example 6.2.5) and *bcolor* (Example 6.2.4).

Example 6.2.4. This example shows a table structure that has three different HTML representations. Figure 6.2.4 (a) uses tag `<b>` to create boldface for indicating a topic node. Figure 6.2.4 (b) uses tag `<th>` to achieve the same effect as Figure 6.2.4 (a). Figure 6.2.4 (c) uses the *bcolor* attribute to emphasize the record, which is a topic node.

Observation 6.2.1. In practice, more and more web authors accept the Cascading Style Sheets (CSS) method to represent the layout structure of their web documents. This trend makes the hierarchical structures more obscure in that we cannot simply use the rules based on the meaning of tags. However, we can still find some regularities in building the hierarchical relation:

1. Compared with regular records, a topic node appears in the preceding positions in the content block.

Figure 6.7: The same hierarchical structure represented by different HTML tags

2. In a set of hierarchies, the number of topic nodes is less than the number of regular records.
3. The hierarchical structure building upon this topic node has the highest frequency of occurrence in the content block.

The common rules of this dynamic clustering are based on the Observation 6.2.1. First, we detect the types of the records based on their properties. If two records have the same properties, such as the same *label* and *attributes*, we mark them with the same type, such as type *A* or type *B*. Then, we employ a *Frequent Structure Mining* (FSM) formula for computing their frequency of appearance in a set of adjacent records. In the FSM formula, we introduce a *Similarity Distance* value to measure the frequency of appearance in a group of adjacent records.

$$\bar{D}_i = \frac{p}{k} \times \sum_{i=1}^n d_{ij} \quad (i \leq j \leq n) \quad (6.1)$$

Equation 6.1 is the formular translation of Observation 6.2.1. It calculates the similarity in the distance of a record λ_i in a block $\{\lambda_1, \dots, \lambda_n\}$. Factor $\frac{p}{k}$ is the appearance of λ_i in the block decided by the number k of records with the same type, and the position p beginning with n represents its first appearance in the block. d_{ij} is the distance between λ_i and λ_j , which represents how many records can be conducted in a hierarchy.

Example 6.2.5. In the web page of SCS Carleton, there is a side-bar as shown in Figure 6.8 that uses the *class* attributes to indicate hierarchical relations.

Figure 6.8: Using the *class* attributes to indicate the hierarchical relations

In this example, all tags are marked with `<a>` but have two different *class* attributes, *scstitle3* and *menuitem*, which are dynamically specified in the style sheet to create the bold style and indentation. To detect the topic nodes from this block, we first denote each node by *A* or *B* depending on the different *class* attributes; then we have a block like $\{A, A, B, B, B, B, B, A, B, \dots\}$. According to the FSM approach, we can calculate that the similarity distance of *A* is $\bar{D}_a = 91.2$, much higher than the similarity distance of *B* at $\bar{D}_b = 3.16$. This result means that type *A* records are topic nodes of a set of topics such as $\{A\}$, $\{A \Rightarrow \{B, B, B, B, B\}\}$, $\{A \Rightarrow \{B, \dots\}\}$ and so on.

Example 6.2.6. In this example, we suppose a more complicated block $\{A, B, A, A, C, B, D, D, F\}$ which has 5 different types of records.

Figure 6.9: Detect the topic nodes via calculating the similarity distance.

Figure 6.9 shows the result using the FSM approach to calculate the similarity distance of records *A*, *B* and *C*. We can find the topic node *B* since it has the maximum $\bar{D}_b$.

6.2.3 Overall Algorithm for One-dimensional Hierarchy

The FSM approach solves the problem of dynamic clustering and can be also extend to solve all one-dimensional space problems. For example, in Example 6.2.1, if we use *A*, *B*, *C* to denote the element nodes `<h1>`, `<h2>` and `<h3>`, we have a content block $\{A, B, B, C, C, B\}$. Using FSM, we can easily get the topic $\{A \Rightarrow \psi_a\}$ with $\psi_a :=$

Algorithm 2 One-Dimensional Aggregation

Input: a content block $\psi := \{\lambda_i | \lambda_i \in \mathbb{R}, 1 \leq i \leq n\}$ with a common parent node x in a tree $T(x)$ **Output:** an hierarchical tree $T'(x)$.

- 1: let $\bar{D}_a$ be the similarity distance of λ_1 , and $\bar{D}_b$ be the similarity distance of λ_2 ;
 - 2: **if** $n < 3$ **then**
 - 3: output the tree $T(x)$ and end the procedure;
 - 4: **else if** $\bar{D}_a < \bar{D}_b$ **then**
 - 5: separate the content block by $\psi_1 := \{\lambda_1\}$ and $\psi_2 := \{\lambda_i | \lambda_i \in \mathbb{R}, 2 \leq i \leq n\}$;
 - 6: call the One-Dimensional Aggregation algorithm with input ψ_2 ;
 - 7: **else**
 - 8: **for** every λ_i has the same type as λ_1 **do**
 - 9: create a virtual node v_1 , and let it be the child of the parent node x ;
 - 10: attach the node λ_i to v_1 as its first child;
 - 11: create a virtual node v_2 , and let it be the child of v_1 ;
 - 12: **while** has direct following sibling node λ_j not be the same type as λ_1 **do**
 - 13: attach the node λ_j to v_2 as its child node;
 - 14: **end while**
 - 15: call the One-Dimensional Aggregation algorithm with input $T(v_2)$;
 - 16: **end for**
 - 17: **end if**
-

$\{B, B, C, C, B\}$. For block ψ_a , we can continue to use FSM to eventually solve the hierarchical problem and obtain the same result as that using Rule 6.2.1. So, based on the FSM approach, we can give an overall algorithm to solve the one-dimensional hierarchical problem.

Algorithm 2 recursively detects the topic nodes in each content block. Since all topic nodes are on the same level in one-dimensional space, we introduce a virtual node for each topic that identifies a new block, and make this node a child of the topic that should be its immediate parent based on the FSM result.

6.3 Extracting the Hierarchies in Two-Dimensional Space

In two-dimensional space, topic nodes and records have different parent nodes and one the different level in the DOM tree. The only case of two-dimensional space appearing in HTML documents is $m \times n$ tables.

Table representation could be very simple (Figure 6.2) or more complicated (Figure 6.10). This is due to the two specific attributes of `<td>` nodes: 1) *rowspan* = i can span the `<td>` cell i rows; 2) *colspan* = j can span the `<td>` cell j columns. Since these variations break the convention of hierarchical representation when compared with a normal HTML table, we need a procedure to recover the normal hierarchies implied by these span attributes.

Algorithm 3 Table Normalization

Input: a $m \times n$ table $\mathbb{T}$ denoted as $\{C_{11}, C_{12}, \dots, C_{21}, \dots, C_{mn}\}$ with i th row and j th column cell $\{C_{ij} | 1 < i < m, 1 < j < n, \text{rowspan} = a, \text{colspan} = b\}$.

Output: a normalized table $\mathbb{T}'$

```

1: for each  $C_{ij}$  do
2:   copy the cell  $C_{ij} \rightarrow \{C_{i(j+1)}, \dots, C_{i(j+b-1)}\}$ ;
3:   copy the cell  $C_{ij} \rightarrow \{C_{(i+1)j}, \dots, C_{(i+a-1)j}\}$ ;
4:   if  $i = 1$  then
5:     for  $x = 1 \dots n$  and  $y = 1 \dots (a - 1)$  do
6:       merger the cells  $C_{yx}$  and  $C_{(y+1)x}$ ;
7:     end for
8:   end if
9:   if  $j = 1$  then
10:    for  $x = 1 \dots m$  and  $y = 1 \dots (b - 1)$  do
11:      merger the cells  $C_{xy}$  and  $C_{x(y+1)}$ ;
12:    end for
13:   end if
14: end for

```

Our table normalization algorithm is based on the intuitive structure indicated by the *colspan* and *rowspan* attributes, and translates their semantics, using the absence of heading cells in the same way browsers do. In conveying the logical relations among the cells,

Algorithm 4 Dimensions Reduction**Input:** a normalized $m \times n$ table $\mathbb{T}$ denoted as $\{C_{11}, C_{12}, \dots, C_{21}, \dots, C_{mn}\}$ **Output:** One-dimension tuples.

-
- ```

1: if the first row records $\{C_{11}, C_{12}, \dots, C_{1n}\}$ are topic nodes then
2: for each rows from the second do
3: construct the hierarchical relation as $\{C_{1y} \Rightarrow C_{xy} | 1 < x \leq m, 1 \leq y \leq n\}$
4: end for
5: else if the first column records $\{C_{11}, C_{21}, \dots, C_{m1}\}$ are topic nodes then
6: for each rows from the second do
7: construct the hierarchical relation as $\{C_{x1} \Rightarrow C_{xy} | 1 \leq x \leq m, 1 < y \leq n\}$
8: end for
9: end if

```
- 

the first row and the first column in this two-dimensional grid are semantically significant. Due to the display characteristics of HTML documents, the first row is more reasonable for the conduction of topics than is the first column, assuming both are topic nodes.

Figure 6.10 (a) is a two-dimensional space with *colspan* and *rowspan* at cells. Using the Algorithm 3, we expand the cells with *colspan* and *rowspan* in columns and rows firstly, as shown in Figure 6.10 (b), and then merge the first two rows since they contain the topic nodes, as shown in Figure 6.10 (c). Finally, we use Algorithm 4 to reduce the two-dimensional space to one-dimensional spaces as shown in Figure 6.11.

Figure 6.10: The process of table normalization

Figure 6.11: Reducing the two-dimensional table to a set of one-dimensional spaces

## Chapter 7

# Adaptive Web Information Extraction

In the previous chapters, we have discussed how to define and construct a *Transitional DOM* from the original HTML source. This chapter focuses on the use of this middle-tier model to build the adaptive web information extraction system. In the first Section 7.1, automatic annotation is provided to bring the semantic elements to markup the hierarchical information following the XML specification and create an instance indicated by an underlying schema. In Section 7.2, we provide instance-level schema discovery post-processing tailored for the extracted content. In Section 7.3, we discuss issues related to wrapper validation and maintenance that are overlooked by the other information extraction systems.

### 7.1 Semantic Annotation

In this section, we provide an automatic annotation algorithm which is motivated by the nature of our Transitional DOM. After the content partition and topic aggregation processes, we have transformed the original HTML DOM into the T-DOM tree where the document structure is enhanced using topics and blocks. The topic nodes in content blocks intuitively

---

**Algorithm 5** Semantic\_Annotation( $T, T'$ )

---

**Input:** A Transition DOM node  $T$  and An XML DOM node  $T'$ **Output:** An XML DOM node  $T'$  with child nodes

```

1: if The Transition DOM node T is a leaf, $children(T) = \phi$ then
2: Append The Transition DOM node T to the XML DOM node T' as a child;
3: else if The Transition DOM node T is a topic node, $T \in \tau$ then
4: Let set $x = \{y | y \in children(T)\}$;
5: Create an XML node t with $label(t) = value(x_1)$;
6: Append t to T' as a child;
7: for $i = 2, \dots, |x|$ do
8: Semantic_Annotation(x_i, t);
9: end for
10: else if The Transition DOM node T is a block, $T \in \psi$ then
11: Let set $x = \{y | y \in children(T)\}$;
12: for $i = 1, \dots, |x|$ do
13: Semantic_Annotation(x_i, T');
14: end for
15: end if

```

---

act as good annotators that can build the hierarchical relations. Given highly hierarchical input, most semantic markup can be done quickly using the topic nodes as the XML elements. For less-hierarchical pages, we can add some text mining approaches, such as regular expression, to improve annotation accuracy.

### 7.1.1 Annotation by Topic nodes

The semantic annotation algorithm we develop here is a process of building the XML DOM tree recursively. First, we give two trees represented by two roots, one for XML DOM the other the HTML source. In our system, this HTML source is actually the T-DOM we have obtained from the original HTML source. Then, we decide the types of the HTML root: if it is a leaf record without child records, we just append it to the XML root and end the process; if it is a T-DOM block that contains child records but without a specific subject, we keep the original XML root and send each of the child records iteratively to the

beginning of the algorithm and run the process recursively; if it is a T-DOM topic that contains a topic node as the first child node, we construct a new XML node according to this topic node and append it as the child node of the XML root, passing this new XML node as the new root of XML DOM, as well as each of the other child records of the T-DOM node to run the algorithm recursively. Obviously, the complexity of this recursive algorithm is  $O(\log n)$ , the same as a recursive binary search process. An example using Algorithm 5 for semantic annotation is given in Example 7.1.2.

**Example 7.1.1.** For the following part of Yahoo<sup>1</sup> page, after the topic aggregation we have the adapted HTML source:

```
<table>
 <tr>
 <th>

 Business & Economy

 </th>
 </tr>
 <tr>
 <td>
 B2B
 Finance
 Shopping
 Jobs
 </td>
 </tr>
</table>
```

Then we can use topic nodes (like *Business & Economy*) to conduct the hierarchical structure and gain the XML result:

```
...
<Business_Economy url="http://www.yahoo.com/r/bu">
```

---

¹<http://www.yahoo.com>

```

<B2B url="http://www.yahoo.com/r/bb">
<Finance url="http://www.yahoo.com/r/fi">
<Shopping url="http://www.yahoo.com/r/bs">
<Jobs url="http://www.yahoo.com/r/jo">
</Business.Economy>
...

```

As happens frequently in practice, when a web document layout structure changes, the wrapper generated based on absolute patterns fails to extract the data. This is the reason that traditional wrapper induction systems are domain specific and not for reuse. The advantage of our layout-based approach is the capability of creating the relative pattern: at the stage of topic aggregation, hierarchical relations were built on both structure and attributes levels, and the DOM tree was rebuilt according to topic nodes.

Our approach involves finding topic nodes within the page that serve as starting points for data extraction. Ideally, topic nodes are established based on the content of a data item, not on its HTML path. For instance, a page that contains the price of a book probably has the word “Price” somewhere near the price value. By looking for the word “Price” we can establish a topic node for the price value and be independent of its absolute location.

Example 7.1.2. Figure 7.1 and Figure 7.2 show the same web site with different layout pages from the faculty page of SCS Carleton University².

We can see that these two pages containing faculty information have similar layouts, but we also see the huge difference in the sources. For example, the topic node is `<b>Jemal Abawajy</b>` in the old page but `<div>Jemal Abawajy</div>` in the new page, and `<b>E-Mail:</b>` has a different path.

After the content partition and topic aggregation processes, both pages can yield almost identical T-DOMs. Figure 7.3 shows the output result of the web page in Example 7.1.2

---

²The old page was gained from Internet Archive WayBackMachine http://web.archive.org/web/*/www.scs.carleton.ca/people/faculty

Figure 7.1: An old version of the web page from SCS Carleton U.

after content partition and topic aggregation. Both the explicit and implicit boundaries are detected and enhanced to group the relevant records together into blocks. Using semantic annotation with topic nodes, the resulting two T-DOM trees have the same structure as follows:

```
<Jemal_Abawajy >
 <data>
 Grids Computing, Cluster Computing, Distributed ...
 </data>
 <email>
 ...
 </email>
 <phone>
 ...
 </phone>
...
```

Figure 7.2: The current version of the web page from SCS Carleton U.

</Jemal_Abawajy>

Note that not all the topic nodes in the topic are useful for building the XML element. For example, we have a topic such as {"We can cook the shrimp in 5 different ways" $\Rightarrow$ {fry, bake, boil, ... }}. Obviously, the topic node has too many words and is not suitable as the element name. In this case, we can build an XML element by a general term like "topic" and append an attribute like "about = ..." to the element.

7.1.2 Annotation By Regular Expressions

It is fairly easy to extract simple data items such as the stock quote shown in Example 7.1.2. The extracted data is simple in structure and its presentation on the HTML page maps

Figure 7.3: Documents in Example 7.1.2 after content partition and topic aggregation

directly to a corresponding XML structure (essentially a flat database record). Structural synthesis may be required in more complex situations. Consider the task of aggregating product catalogue data from several Web sites. Here, it is essential to represent catalogues and the products they contain in as much detail and fine granularity as possible so that integration of the catalogues can be successful. Some web pages don't support explicit structure and semantics within the documents; there are no topic nodes for this kind of information because it is considered *common sense*. For instance, on many online shopping sites, product features are embedded in plain text paragraphs, and some data may be omitted because it is implicitly understood by the user viewing the page or is available elsewhere (e.g. that a laptop computer has an LCD display as opposed to a CRT display).

Regular Expression (RE) is the simplest NLP (Natural Language Processing) technique

that can be used to extract structured data from snippets of unstructured text, such as time and phone number.

Example 7.1.3. A set of street addresses: “1020 Bank St., 512 Oak Blvd., 416 Main St. and 97 Adams Blvd.”, that all start with a pattern (Number + Capitalized Name) and end with (Blvd.) or (St.).

We refer to the starting and ending patterns together as the regular expression:

```
<address match="\d?\s[A-Z].*\s[(Blvd)|(St)].*" type="regular express"/>
```

Another common usage for RE is to construct hierarchies via free text mining. For instance, a web page concatenates several data items into a single plain text field (such as *Chapter 1*, §1 or *Part 1*) to represent the structure instead of using layout tags. This analysis is potentially less accurate than a deep linguistic analysis but is still very powerful and quick to master. Obviously, using RE can improve the quality of the extraction in cases where hierarchy structure is lacking.

7.2 Schema Discovery

Schema discovery technologies hold the promise of an intelligent information system which takes the existing data with a particular structure [16]. Our information extraction system also benefits from the schema of data in the stages of both data extraction and population.

7.2.1 Instance-level Approaches

Some Web documents are supported by a back-end database which has a rigid schema, but this schema resides only in the applications. After a Web page is generated, this structural

information is always mingled with diverse presentations and hidden inside the HTML format. Previous approaches that attempt to construct the schema information directly from the HTML source, such as DIPRE [6], ROADRUNNER [12] and EXALG [3], have suffered from such problems and have had to limit their achievements to a specific domain.

Instance-level data can give important insight into the contents and meaning of schema elements. This is especially true when useful schema information is limited, as is often the case for semi-structured data. The main benefit of evaluating instances is a precise characterization of the actual contents of schema elements. It is more intuitive that we construct the schema from its instances.

Inducing the schema of a given set of XML documents is a well studied subject and significant effort has been invested in this field, as discussed in a survey [14]. With the help of the semantic annotation process, we have created a set of XML documents for a particular site. To develop a schema discovery algorithm is not an objective of this thesis, but we can take advantage of existing instance-level schema discovery techniques to determine a more general structural schema from a large set of specific patterns.

Recently, several approaches to the discovery of schemas have been proposed in [35, 33, 20, 10, 34]. The approaches differ in the type of schema discovered and the assumptions made concerning the semantics of structures in the underlying XML documents. However, all of these approaches assume that the semi-structured data in the underlying XML documents are generated homogeneously and semantically. For XML documents that are converted from HTML pages, the schema may contain heterogeneous structures due to the diversity of HTML documents. This is especially true for the instances we created automatically via semantic annotation, which may also contain *un-certain* elements such as “topic”, “data” due to a lack of semantic indications. In this scenario, we choose to adopt some modifications to deal with these characteristics, compared with previous work.

7.2.2 Schema in Data-rich Section

Semi-structured data is characterized by the lack of any fixed and rigid schema, although typically the data has some implicit structure, especially in the so-called *data-rich* section, such as the genuine tables in a web page. For the discovery of common structures among a set of XML documents, we use an approach similar to [34] in which ordered trees representing XML documents are mapped to paths. Unlike [34], which tries to create one common schema for all inputs, we induce a set of *mini-schemas* that represent only the data-rich sections in the Web documents.

Recall that Web pages from the same web site often use the same HTML design template, so that their HTML structures are quite similar if not identical. Moreover, the same information appearing concurrently in those pages is often used for navigational purposes, advertisement or other nepotistic purposes; usually such information has little to do with the main content of the page and remains constant throughout the entire site. Two XML documents produced from the same schema can have very different sizes on account of optional and repeating nodes. Given two XML DOM trees representing two extracted pages from the same web site, the contents that change significantly should be the data-rich sections.

Edit distance (or Levenshtein distance) is a well known NLP method of comparing the difference between two strings by counting the minimum number of simple edit operations required to transform one string into the other [32]. We found this method is more indicative of the notion of structural similarity. The basic idea is to traverse these two trees using a depth-first and left-to-right order and compute the edit distance node-by-node from the root to the leaves so as to find the sub-trees that have the largest edit distance. However, before explaining our data-rich section extraction algorithm in more detail, we need to define several functions employed in the process.

In our case, we just need two edit operations in the construction of our algorithm. Given a tree T with $label(T) = l$ and first level subtrees $\{T_1, \dots, T_m\}$, the tree transformation operations are defined as follows:

1. **[Insert Tree]** Given a tree X , the function $insert(T, X, i)$ is an insert tree operation applied to T at i that yields the tree T' with $label(T') = l$ and its first-level subtrees $\{T_1, \dots, T_i, A, T_{i+1}, \dots, T_m\}$.
2. **[Delete Tree]** The function $delete(T_i)$ is a delete tree operation applied to T at i that yields the tree T' with $label(T') = l$ and its first-level subtrees $\{T_1, \dots, T_{i-1}, T_{i+1}, \dots, T_m\}$.

Definition 7.2.1. Given any trees A and B and the sequences of edit operations ED that when applied to A will yield a tree equal to B , then the edit distance is denoted by $\theta(A, B) := |ED|$ with each operation $op \in ED$ in sequence being allowable if it satisfies the following two conditions:

1. A tree x may be inserted only if x already occurs in the source tree A .
2. A tree x may be deleted only if x occurs in the destination tree B .

This restriction limits the use of the insert tree and delete tree operations to cases where the subtree that is being inserted (or deleted) is shared between the source and destination trees. We can only insert (delete) subtrees that are already *contained in* the source (destination) tree. A pattern tree x is said to be contained in tree T if all nodes of x occur in T . Without this restriction on allowable sequences of edit operations, one could delete the entire source tree in one step and insert the entire destination tree in a second step.

The algorithm works recursively. For two nodes, if they are not the same (the edit distance > 0), then the algorithm will go down one level to match their children from the leftmost to the rightmost one. If the nodes are leaf nodes or the same, the algorithm will

Algorithm 6 $DSE(A, B)$ **Input:** Two XML DOM trees A and B correspond to two XML documents.**Output:** Data-rich section T' of the subtree $T' \subseteq A, T' \subseteq B$.

```

1: Let $m = |children(A)|, n = |children(B)|$, m, n are the number of first level subtrees
 of A and B .
2: if $m = 0$ or $n = 0$ then
3: return the data-rich section T' .
4: else
5: Let each first level subtrees $T_x \in \{A_1, \dots, A_m\}$ and $T_y \in \{B_1, \dots, B_n\}$
6: Let data-rich section $T' = \phi$ and current edit distance $d = 0$.
7: for $i = 1, \dots, m$ do
8: for $j = 1, \dots, n$ do
9: Let edit distance $d' = \theta(T_i, T_j)$
10: if $d' > d$ then
11: Let data-rich section $T' = T_i$ and current edit distance $d = d'$.
12: $DSE(T_i, T_j)$
13: end if
14: end for
15: end for
16: end if

```

---

move to their siblings and continue to compute the edit distance, if any. If all of the children of two parent nodes have been compared, the subtree that represents the Data-rich section with a maximum edit distance will return.

To analyze the complexity we can see that the algorithm only compares nodes from the same level of two trees. If we let the height of source tree A be H , the time complexity of our algorithm for measuring the edit distance between a source tree A and a destination tree B will be $O(H \times |A| \times |B|)$. However, this is the worst case. In practice, we can avoid many comparisons since we do not need to compare those nodes whose semantic meanings are uncertain.

In [34], the schema is created from a search space S which is defined as a common schematic structure among a set of XML documents $\mathcal{D}_X$ that can be determined by the *path* express derived from the trees associated with the documents. Let T_D denote the tree

associated with a document $\mathcal{D}$. Then $s := \{p | p \in \text{path}(T_D) \wedge \mathcal{D} \in \mathcal{D}_\chi\}$. If we can find the data-rich section in $\mathcal{D}$, then replace T_D with this data-rich section T' , we will get a new search space S' . Finally, we can use the schema discovery approach introduced in [34] to extract the *mini-schemas* of T' in $\mathcal{D}$.

Figure 7.4: Data-rich section in web pages from Amazon site

Example 7.2.1. Web pages from Amazon³ site are generated by applications with an underlying database schema support.

Using DSE algorithm, the data-rich section can be detected as shown in Figure 7.4. The HTML source of this section is as follows:

Parts of HTML codes of page 1:

```
<b class="sans" path="//body[1]/table[2]/tr[1]/td[3]/form[1]/b[1]">
 Janice VanCleave's A+ Science Fair Projects

...
<table path="//body[1]/table[2]/tr[1]/td[3]/form[1]/table[3]">
```

³<http://www.amazon.com>

```

<tr>
 <td class="small">
 <b path="//body[1]/table[2]/tr[1]/td[3]/form[1]/table[3]/tr[1]/td[1]/b[1]">
 List Price:

 </td>
 <td path="//body[1]/table[2]/tr[1]/td[3]/form[1]/table[3]/tr[1]/td[2]">
 <td class="small"
 path="//body[1]/table[2]/tr[1]/td[3]/form[1]/table[3]/tr[1]/td[3]">

 $14.95

 </td>
</tr>
...
</table>

```

Parts of HTML codes of page 2:

```

<b class="sans" path="//body[1]/table[2]/tr[1]/td[3]/form[1]/b[1]">
 Chemistry (Cliffs Quick Review)

...
<table path="//body[1]/table[2]/tr[1]/td[3]/form[1]/table[3]">
 <tr>
 <td class="small">
 <b path="//body[1]/table[2]/tr[1]/td[3]/form[1]/table[3]/tr[1]/td[1]/b[1]">
 List Price:

 </td>
 <td path="//body[1]/table[2]/tr[1]/td[3]/form[1]/table[3]/tr[1]/td[2]">
 <td class="small"
 path="//body[1]/table[2]/tr[1]/td[3]/form[1]/table[3]/tr[1]/td[3]">
 $9.99
 </td>
 </tr>
 ...
</table>

```

With the help of *path* express, it is easy to find the patterns of matching the topic nodes

with values. For instance, a prevalent pattern of finding the value of the *List Price* topic node can be matched to the path express as:

```
<List_Price match="List Price:" type="topic node" >
 <value-of match="/body[1]/table[2]/tr[1]/td[3]/form[1]/table[3]/tr[1]/td[3]"
 type="absolute path"/>
</List_Price>
```

After all the prevalent patterns are discovered, schema discovery approach in [34] will be employed to determine a common schema of this data-rich section in the form of a DTD, which is like the follows:

```
<!ELEMENT book (Author+, List_Price, Price, You_Save?, Availability, Reading_level?,
 Edition) >
<!ATTLIST book title CDATA #REQUIRED >
<!ELEMENT Author #PCDATA >
<!ELEMENT List_Price #PCDATA >
<!ELEMENT Price #PCDATA >
<!ELEMENT You_Save #PCDATA >
<!ELEMENT Availability #PCDATA >
<!ELEMENT Reading_level #PCDATA >
<!ELEMENT Edition #PCDATA >
```

7.3 Wrapper Validating and Maintenance

Given the diversity of problem domains, Component-based Software Engineering holds considerable promise in terms of reusable parts, and for the maintenance and improvement of systems by means of component replacement and customization. Based on this motivation, we provide an essential solution that intends to ease the problems of wrapper validation and maintenance in the information extraction system. Before discussing our approach, we give a new definition of wrappers.

Definition 7.3.1. A wrapper is a program $\mathcal{W}$ that consists of a set of template functions $\mathcal{T}$. Each template is said to cover a subset of instances $\mathcal{I}$ belonging to a common schema $\mathcal{S}$ which appears in a set of HTML documents $\mathcal{D}$.

Based on this new definition, the wrappers we want to build are multi-function programs. A multi-function program can be represented as a set of functions $\mathcal{T} : \mathcal{D} \rightarrow \mathcal{I}^{\mathcal{D}}$, each extracting subsets of instances $\mathcal{I}^{\mathcal{D}}$ from the documents $\mathcal{D}$ sharing a similar structure of $\mathcal{S}$. The wrapper is complete when the generated instances cover all the instances of $\mathcal{S}$. It is said to be consistent when no more than the instances of $\mathcal{S}$ are covered. Recall and precision measure the completeness and consistency, respectively, of a multi-function wrapper.

Figure 7.5: The traditional wrapper and the multi-functional wrapper

Figure 7.5 (a) displays a traditional wrapper $\mathcal{W}$ that deals with the entire program as an integrated whole. Any patterns from the original HTML source to the target XML document break the program. Figure 7.5 (b) displays a multi-functional wrapper $\mathcal{W}(\mathcal{T})$ in which one program is separated into a set of templates $\{\mathcal{T}_1, \dots, \mathcal{T}_4\}$.

In considering wrapper maintenance, the advantage of splitting the extraction task into a set of subtasks and letting each template function handle one of them is obvious. One erroneous extraction in a subtask will not break the whole program. Moreover, the template can be decomposed and recomposed into a new wrapper program relevant to a new domain.

For instance, the template $\mathcal{T}_4$ could be a template that comes from another wrapper program generated for a different HTML source, which is now composed into this wrapper program for the extraction task. This process is usually needed to build two different wrappers for a traditional information extraction system.

With respect to wrapper validation, it is almost impossible for the traditional wrapper to indicate which point of the program is malfunctioning when we at last discover that the extraction result is a mess. Since it has only one operation, validating a template function is very easy. We can even build a *validator* template to detect changes in the web source.

Example 7.3.1. The following template is built from Example 7.1.2 and works like a *validator* for testing changes in the HTML source page.

```
<Jemal_Abawajy match="Jemal Abawajy"
 path="body[1]/table[1]/tr[1]/td[2]/table[1]/tr[1]/th[1]" >
 <email match="E-Mail:"
 path="body[1]/table[1]/tr[1]/td[2]/table[1]/tr[3]/td[1]/b[1]" >
 <value-of match="abawjem (@scs.carleton.ca)"
 path="body[1]/table[1]/tr[1]/td[2]/table[1]/tr[3]/td[1]/b[1]/text()[1]" >
 </email>
 </Jemal_Abawajy>
```

In this template, there are a total of three matches. The first two element nodes are topic nodes; if both are matched, the structure of source page must be unchanged. The last node matches a text node as the record in the source page. It is normal when this value is changed while its parents are matched. However, if all three of these matches are invalid, we can confirm that all the templates generated according to this source are invalid.

Chapter 8

Design and Implementation of XML Gateway

This chapter introduces the design and implementation of *XML Gateway*, an adaptive IE system designed with portability and maintainability, that serves as an HTML to XML converter on the web. Throughout the experimental evaluations, we show that our HTML-aware modelling method leads to a highly automatic and accurate transformation for extracting semi-structured documents on the web. Our purpose here is to demonstrate, using our fully automatic information extraction framework, the inter-operability between web sources and applications.

In Section 8.1, we give a set of requirements for the implementation of a web IE system. With these goals in mind, we design a system architecture in Section 8.2 which can work in two different modes. Section 8.4 introduces the extraction language and explains why we chose XSL in our IE system. Section 8.5 describes the details of the system's core

implementation called “XML Template Factory”, a pipeline structure to filter the input documents. Section 8.6 applies the implemented system to various web documents and evaluates the design and approaches provided by this thesis.

8.1 Requirements of the IE System

To support the design goals of automatic Web information extraction, our system is designed to meet the following requirements:

1. Functionality requirements.

- (a) System should work fully automatically without human intervention and keep the cost of developing and maintaining as low as possible.
- (b) The extracted output data should be highly structured and easily accessible for use on various applications.

2. Flexibility requirements.

- (a) Mobility is one of requirements for a flexible IE system to be able to migrate in a self-directed way from one host platform to another.
- (b) With respect to relatively complex document structures and frequently changing contents, this system should be both resilient and adaptive to be able to handle the dynamic nature of the Web.

3. Performance requirements.

- (a) Each module of the system should be layered with respect to the various extraction stages and keep the implementation independent, understandable, maintainable and reusable. Components should inter-operate with each other to achieve a common goal.
- (b) Given a specific domain, the extraction results must be highly accurate while keeping the extraction process fast.

8.2 XML Gateway System Architecture

XML Gateway has been written in J2EE and implemented on Apache Web Server. In order to make our extractor easy to use, we implemented it as a web proxy. This allows an administrator to set up the extractor and provide content extraction services for a group. The proxy is coupled with a graphical user interface (GUI) to customize its behaviour. The current implementation of the proxy is in Java for cross-platform support.

Figure 8.1 shows the system architecture of “XML Gateway”, a semi-structured information wrapper system. Our solution employs multiple extensible techniques that incorporate the advantages of previous work on Web information extraction. There are three sub-systems under the main-frame of the whole system: *Document Retrieving Module*, *Data Extraction Module* and *XSL Templates Factory Module*. These sub-systems work together in two extraction modes, (*Expressive* and *Enhanced* extraction). In expressive mode, the extraction patterns are retrieved directly from pattern repositories and used for mapping XML with the HTML DOM trees. If the extraction patterns are not available or the old patterns cannot extract the right data, the HTML DOM passes into an enhanced mode, where a set of structural and semantic syntheses are processed to find the extraction

Figure 8.1: System architecture of the XML Gateway wrapper system.

patterns.

8.3 Retrieving the Web Documents

The initial job of the Web wrapper is to retrieve the to-be-processed Web document. The retrieval module is in charge of issuing an HTTP request to a remote web server and fetching the corresponding HTML pages as any Web browser would do. For the wrapper, it is completely transparent, since the entire job is done in the background by the retrieval module: the creation of the HTTP request, management of connections, handling of redirections and authorizations, and so on.

The *URL* might contain some variables which need to be replaced by their string values in order to offer parameters. If the target resource is a service, the implementation of the module will stimulate the service according to the query. In the case of an access

to a static HTML page the parameter is composed of variables consisting of two strings (*key = value*) used to construct the query string or the body of the message depending upon the format of the request (HTTP Get or HTTP Post).

As mentioned earlier, much of the HTML content on the Web today is ill-formed because it does not conform to HTML specifications. Therefore, the first step taken in the “XML Gateway” framework is to pass the original HTML page through a filter that “repairs” the broken syntax and produces well-formed HTML, or what is today known as Extensible HTML (XHTML) [44]. Toolkits for this step exist already, including the Jtidy package [37].

In order to analyze a web page for content extraction, the page is first passed through an HTML parser that corrects the HTML and creates a Document Object Model tree representation of the web page. This process accomplishes the steps of structural analysis and structural decomposition. The DOM tree is hierarchically arranged and can be analyzed in blocks or as a whole, providing a wide range of flexibility for our extraction algorithm.

The extraction is not performed on the document itself (a document is just a big bag of strings), but on an abstract representation of it, better suited for high-level manipulation. Thus each document retrieved is parsed into a tree structure corresponding to its HTML hierarchy according to W3C’s DOM standards. Such a tree consists of a root, internal nodes and leaves. Each node corresponds to an HTML tag or text string (*PCDATA* nodes). Given this, it is important to note that there is 1-to-1 mapping between the contents of an HTML document and its DOM nodes.

8.4 Extracting Language

Unlike systems such as [4, 5] which use their own specific extraction languages, the extraction language of this system is *Extensible Stylesheet Language* (XSL)[42], which is being developed as part of W3C's *Style Sheets Activity*. The fact that W3C has started developing XSL in addition to CSS is that CSS can only be used to style HTML documents. XSL, on the other hand, is able to transform documents. An XSL Transform (XSLT) program (called stylesheet) is a set of template rules, each of which has two parts: a pattern that is matched against nodes in the source document and a template that can be instantiated to form the resulting document. For example, XSL can be used to transform XML data into HTML documents on a Web server. In practice, more and more commercial sites use XSL as the template language for page creation from a document database. The reasons behind our choice of XSL as our extraction language are as follows:

- The base of our auto-wrapper system is the HTML DOM tree. In order to navigate through the tree graph, the language needs a path tracing capacity. XSL syntax consists of XPath specifications, which have document manipulation capabilities beyond styling. XPath is a language for addressing parts of an XML document, essential for cases where you want to say exactly which of a set of documents are to be transformed by XSL. XPath allows you to say, for example, “select all paragraph belonging to the chapter element,” or “select the elements called special notes.” XSL is the most intuitive choice for DOM transformation.
- As an international standard, XSL has been accepted broadly and many applications and tools have been developed based on this standard. For the XSLT tools, which represent the key process in transforming HTML to the XML, we have many choices,

including 1) *Oracle XMLparser* 2) *Microsoft MSXML* 3) *IBM Xalan* 4) *Saxon*, *Jdom*

- The W3C XSL specification, as of July 1999, was split into two separate parts. The first part deals with the syntax and semantics, while the second part describes how the document is transformed into another document that uses the formatting vocabulary. The splitting of the XSL and the XSL transformation makes the process of extraction more flexible and program independent: we simply change the templates in the XSL files for different extraction tasks without rewriting the entire transformation code.

8.5 XSL Templates Factory

For the implementation of the core modules of the system, we use a structure called *XSL Templates Factory*. XTF inputs the original HTML DOM into a pipeline consisting of a series of filters that can modify specific nodes to restructure the content of the document. Figure 8.2 shows a basic filter unit with the XSL transformer which can take input from a DOM and XSL templates to output a new DOM.

Figure 8.2: A basic XSL filter unit on a XTF pipeline which transforms DOMs in respect of the templates.

The term *filter* is more appropriate than transformation for an XSL script whose output shares more similarities than differences with its input. A water filter removes impurities, but the result is still water; boiling transforms water into steam, a substance altogether

different. XSL filters have an equally broad capability. For example, XSL filters can strip away spurious differences to isolate those of semantic importance, allowing the IE system to detect them unencumbered. Filters may augment as well as subtract; just as a water filter might add flavour, an XSL filter might add markup to XML data. All filters work together on a pipeline for the HTML to XML transformation eventually, and each of them can be easily turned on and off and customized to a certain degree.

Figure 8.1 shows that the XSL transformation is divided into three stages corresponding with *content partition*, *topic aggregation* and *pattern recognition*. The output of one stage is the input of the next stage. Corresponding to each process stage, the whole XTF pipeline consists of three sets of filters with different levels of granularity.

8.5.1 Content Partition Filters

The first set of filters identify the structural objects from the input HTML DOM tree and group the relevant record objects in blocks with clear delimiters splitting them. The content partition process pipeline is shown on Figure 8.3.

Figure 8.3: XTF filters for content partition pipeline.

Normalize-space.xsl Templates in this filter function like a document cleaner. There is probably significant “noisy data” [47], such as active-X objects, scripts, instructions, and whitespace, which are helpful to user browsing but complicate the process of information

extraction. When dealing with web pages containing both data objects and “noisy data”, the “noisy data” could be wrongly matched as correct data, resulting in either inefficient or even incorrect wrappers. While the document is cleaned up after this filter, the structured objects are clear to the next operations.

ContentBlocks.xsl This filter accepts the input DOM with clear structured objects and detects them as a single content block. The NDF algorithm is used to identify the main-body blocks, so as to detect all the high-level content blocks. Then, each block is iteratively partitioned into smaller ones by analyzing all the explicit delimiters (nodes with child record objects) according to their properties. Changes in the properties of adjacent delimiters usually lead to boundaries which distinguish content blocks.

DelimiterProjecting.xsl This filter uses the projection algorithm to find the explicit boundaries and reconstruct the output DOM so as to simplify the operation of blocks. This is achieved by reducing the size of delimiter objects on the same level in the DOM tree.

8.5.2 Topic Aggregation Filters

The second set of filters is more complex and algorithmic, providing a higher level of extraction than offered by the content partition filters. This set, which can be extended, consists of the headings arranger, the definition list arranger, the complex table normalizer, and the virtual node reconstructor. The topic aggregation filters pipeline is shown in Figure 8.4.

Figure 8.4: XSL templates for topic-level hierarchical reconstruction.

Headings.xml This filter analyzes the HTML heading tags `<H1>` to `<H6>` and reconstructs the nested relations of heading levels. Since there is no inherent hierarchical structure between heading tags in the input DOM tree, this is a process that requires recursive transformation via template mode calling. We utilize the HTML table specification to revise the heading structure and construct the hierarchical relations according to their intuitive levels.

DefinitionList.xml This filter analyzes the HTML definition list `<dl>` and reconstructs the nested relations of one level hierarchy between `<dt>` and `<dd>`. Compared with the heading tags, the definition list can be easily reconstructed into an HTML table structure without recursive transformation.

TableNormalization.xml This filter is the implementation of the table normalization algorithm developed in Chapter 6 Section 6.3 (Algorithm 3) that analyzes the $M \times N$ tables with *colspan* or *rowspan* attributes and rebuilds the table with normalization.

VirtualNodes.xml This filter reconstructs all the hierarchical relations detected by conductor records with virtual nodes. Although virtual nodes have no display effects on the browser screen, they can be easily recognized by programs and help the wrapper generation

process that follows.

8.5.3 Pattern Recognition Filters

This third set of filters are the final stage for the adaptive information extraction system. Starting with the semantic annotation filter, all the presentation nodes in the HTML source are discarded, and only the data at leaf nodes are transformed into the XML document. Then, we operate the output in three directions for post-processing. First, the XML document can be delivered directly to another system or service for users. Second, input into a pattern recognition filter can be used to discover the intuitive structure and output of a schema. Finally, we build the templates for the cardinality of extraction patterns.

Figure 8.5: XSL templates for absolute and relative patterns extraction.

Annotation.xml This filter implements the semantic annotation algorithm as described in Section 7.1. It takes a depth-first, left-to-right traversal through the input Transition DOM tree and builds the XML DOM nodes according to the semantic rules.

SchemaComposing.xsl Schemas are valid XML files in their own right, and may differ from each other. This input-side XSL script facilitates schema comparisons. (The 1999 version assumes the XML Schema 1999 definition.) It is useful to track schema changes over time, for example, over the course of schema development. Schema composition can also be useful in consolidating schemas designed for similar purposes. While the XSL derivation of this filter is rather complex, its action is simple. The filter performs attribute insertion.

Document Type Definitions are an older alternative to schemas, but are not an XML format. Therefore, XTF cannot differentiate DTDs, whereas XTF can differentiate the XML files controlled by a DTD, or indeed any well-formed XML whatsoever, whether tied to a Schema or a DTD or free-standing.

Templates.xsl A pattern is used to locate the right nodes with relevant information on the HTML tree, and extract them to create a structured format. This filter analyzes the extraction patterns and composes the templates to group the common patterns into an extraction function. The results are stored in a template file just as with XSL documents.

8.6 Experiments and Evaluation

The approach of the extraction task described in Chapter 7 was implemented in the XML Gateway system described in the above sections. This section describes how the experiment was setup and the results of the valuation.

8.6.1 Experiment Setup

A consequence of the regularity of structure in the result of IE is that it makes it possible to establish a notion of a “correct answer”. By comparing the system’s result against the correct answer, the performance of the system is easily evaluated. 20. IE standards of measurement grew out of the IR measurement metrics of “Recall” and “Precision”, where recall can be interpreted as a measure of the records that have been extracted and precision as a measure of the percentage of extracted records that are correct. Both recall and precision are always on the interval from 0 to 1 with their optimum value being at 1.

Since our purpose is to develop a general purpose extraction system that is domain independent, we selected different style and change period web pages to test our solution and give the results for both recall and precision. Based on this motivation, we build two groups of test sources. One group is consisted of 12 selected web pages from the educational organizations, the governments or personal sites. These pages have simpler styles and longer changing period. The other group is consisted of 8 selected web pages from commercial web sites which have more complicated styles and shorter changing period.

8.6.2 Results and Analysis

We run the *XML Gateway* application on Pentium IV 2.6 GHz CPU with 512M memory. The operation system is Windows XP and the average running time for a web page is 1.67 seconds. The testing result for the pages of the first group are listed in the Table 8.1, and the result for the pages of the second group are listed in the Table 8.2.

Web Page	Recall	Precision
www.scs.carleton.ca/people/faculty	0.95	1
www.informatik.uni-trier.de/ley/db/	1	1
www.harvard.edu/academics/	0.91	0.98
www.hpl.hp.com/personal/Zhichen_Xu/index.html	0.89	0.92
www.iiasa.ac.at/Research/LUC/ChinaFood/data/t_data	0.80	0.88
www.unescap.org/pop/data_sheet/2003/country.asp?	0.87	0.93
www.dbai.tuwien.ac.at/staff/	0.79	0.87
jupiter.scs.carleton.ca	1	1
www.cuhk.hk/en/research.htm	0.95	0.96
www.stanford.edu/home/faculty/	0.93	0.86
www.nyu.edu/academics/nyu	0.9	1
www.gsfc.nasa.gov	0.92	0.87
Average:	0.91	0.94

Table 8.1: Recall and Precision results for pages from non-commercial sites

Web Page	Recall	Precision
www.yahoo.com	0.96	1
209.47.1.197/content/aboutus.html	0.99	1
www.microsoft.com	0.9	0.84
www.amazon.com	0.7	1
www.go.com	0.89	0.65
news.bbc.co.uk	0.8	0.78
www.apple.com/software/	0.78	0.83
www.nhl.com	0.9	0.88
Average:	0.87	0.87

Table 8.2: Recall and Precision results for pages from commercial sites

To analyse the recalls and precisions, we can see that for some web pages, our implemented system has reached the optimum value of both recall and precision at 1. But for some web pages, the recall and precision are not satisfied (0.7 and 0.65). The reason for these lower recall and precision values is that we generate wrappers based on exploiting the HTML layout structures. But in practice, we commonly encounter high-hierarchical structures that lack semantic meanings. In this case, extraction can be accomplished by relying directly on the data. Regular expressions work well for many types of constants and can be fine-tuned to work even better. However, regular expressions are not likely to be powerful enough to recognize all constants. Names of people and places are particularly difficult, especially if parts of the names are to be inferred from the context (e.g., children have the same last name as their parents unless otherwise stated, country names of major cities are understood without comment). Dates are seemingly easier, but also present difficulties, such as month-day order depending on cultural context and partial dates whose remaining parts are understood from context.

Comparing with the results for the two groups of web pages, our implemented system works more effectively on the pages that have simpler styles and longer changing period, such as the pages from educational organizations or governments (with the average recall is 0.91 and average precision is 0.94). For the web pages from the commercial web sites, both the average recall and average precision are 0.87. The reason for this difference is that there are many advertising information in commercial sites. These so called “noisy data” are normally consisted of active-X objects, scripts, instructions, and whitespace, which are helpful for information displaying but can not be processed by information extraction systems. While such “noisy data” are cleaned up by filters, some information are also missed for extracting.

Since our goal is to develop a general purpose extraction system that is domain independent and works for a broad spectrum. With the overall experimental result for average recall is 0.9 and average precision is 0.92, it shows that our system reached a highly accurate data extraction for the semi-structured documents on the web while keep the conversion from HTML to XML fully automatic.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

The Web is the source of a huge number of online documents that conform to the HTML specification. Web documents are different from the documents traditionally targeted by information extraction, in that the volume is very large, new documents appear often while old document contents often change, a large portion of the documents contain structured or semi-structured texts, and the documents often contain hyper-linked information. Therefore, Web documents have provided new challenges for the IE field.

Web IE is an emerging web mining technology, whose function is to process semi-structured data from given web sites and transform them into a well-structured, feature-rich representation. Web IE systems rely on patterns that extract relevant information from a domain-specific collection of documents and fill the slots of a predefined template. These

patterns can be created either manually by knowledge engineers (knowledge-based) or automatically by exploiting machine-learning techniques. In both cases, a domain-specific collection of Web pages annotated by domain experts, with information relevant to the extraction task, is required.

The core IE engine uses a cascade of sets of patterns to match the linguistic complexity. Each pattern consists of an associated mapping from syntactic to semantic form. The pattern sets are customized for each new topic, as defined by the set of facts to be extracted. An effective pattern must be precise and have wide coverage. A common goal of all web IE systems is to generate wrappers that are highly accurate and robust, while demanding as little effort as possible from users. In practice, this imposes an important trade-off between the degree of automation of a tool and the flexibility of the wrappers generated by it.

This thesis leverages the problem and solves it in three stages. First, we introduce the structured objects (records and topics) as the facts of the extraction task. With respect to these structured objects, we use both the explicit and implicit features of the page's layout to split the whole page into specific content blocks according to visual boundaries. Then, we analyze the nested structures of all the records under the frame of these individual content blocks so as to explore the hierarchical relations between them. Meanwhile, we reconstruct the page's representation for the sake of reducing the dimension of complex spacings. Finally, in the process of semantic annotation, the extraction patterns are created to extract the data based on the intentions of the page's original author. The whole extraction process is automated without any human interventions.

Our approach is less labour-intensive than other approaches that manually or semi-automatically generate wrappers. Experimental results show that it achieves good recall

and precision ratios for these kinds of data-rich, multiple-record web pages (e.g. advertisements, movie reviews, weather reports, travel information, sports news and many others).

A problem that has been largely ignored by past similar work is that site contents change often. The advantage of our wrapping technique over previous work is that it is generally insensitive to the changes in web page format. Since our approach relies on the structure of the presentation features of the data within a web page to generate rules or patterns to perform the extraction task, the content change would not affect our correct extractions. This significant departure from past approaches is a key difference in what we present here. Generating the extraction template in a relative way makes our approach more resilient to dynamic on-line documents than are rigid, grammar-oriented wrappers. It also has the ability to verify the extraction rules to ensure that the correct data continues to be extracted, and to adapt automatically to minor changes of structure in the sites from which the data is being extracted.

Using the *XML Gateway* application, one could retrieve information much more accurately than today. This solution involves the creation of a sequence of relatively simple XSL style-sheets for the transformation from HTML to XML. It encapsulates the extraction logic in external XSL files that are easy to develop and maintain. The information in XML documents is so precisely described by the markup that one can search them much more effectively than when using the primitive text searches currently available from search engines like Google¹ and Alltheweb². One could envision things like a central search engine for chip vendors allowing for very precise searches for components by specification, almost as if they were in an ordinary relational database.

---

¹www.Google.com

²www.alltheweb.com

The approach provided in this thesis, however, does not constitute a Web query language like [31] and [29]. Instead, once we utilize the abundant XML integration techniques to populate the model instance into a database, or use XML Query Language (XQL) techniques to query the XML data directly, we can populate the data into the relational database so that we can query the information using standard query languages such as SQL.

9.2 Future Work

The system described in this thesis is still in the development cycle. Even though it can already handle simpler tasks, the goal is that it should be able to handle complex tasks and still maintain the automation and flexibility. We believe that the most promising avenue is a hybrid approach that combines domain-specific accuracy and domain-independent flexibility, that is, pursuing a domain specific methodology only after all generic, domain-independent features have been fully utilized. Through such a combination, a domain specific wrapper can be expected to be more expressive, the process of wrapper-induction is less complicated, and the cost of wrapper maintenance is reduced.

The following list presents some topics and problems that need further work to improve the extraction quality and performance of our web IE system.

- Additional experiments should be performed to evaluate the system further in different environments and levels of complexity.
- The problem of dynamic structures need further investigation and experimentation. For example, it is common to have multi-page tables, i.e. lists that are split across several pages since they will not fit on a single page, and special consideration should

be given to the structural information for these lists. The document retrieving module should be given the ability to handle various types of structures that needs special handling.

- Our approach for web information extraction relies on the structure of presentation features of the data within a document. However, generating rules or patterns to perform extraction can be accomplished by relying directly on the data. Given a specific domain application, an *ontology* can be used to locate contents present in the page and to construct semantic objects with them. Ontology is a branch of philosophy that attempts to model things as they exist in the world [7]; it is particularly appropriate for modelling objects, including their relationships and properties [46]. A conceptual-model-based ontology can provide a mechanism to represent knowledge, extract information and give symbols a specific meaning in a particular context. Future implementation of *XML Gateway* could automatically discover existing ontologies on the Web so as to make the retrieval process task-specific.
- Since different users might have diverse focuses over the data in an input HTML document, it is worth considering some additional ability to interact and give feedback to the system. This includes both human and machine users.

Bibliography

- [1] B. Adelberg. Nodose - a tool for semi-automatically extracting structured and semi-structured data from text documents. In *ACM SIGMOD international conference on Management of data*, pages 283–294. ACM Press, 1998.
- [2] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM International Conference on Digital Libraries*, 2000.
- [3] Arvind Arasu, Hector Garcia-Molina, and Stanford University. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data*, pages 337–348. ACM Press, 2003.
- [4] Fabien Azavant Arnaud Sahuguet. Web ecology: Recycling html pages as xml documents using w4f. In *The WebDB Journal*, pages 31–36, 1999.
- [5] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual web information extraction with lixto. In *The VLDB Journal*, pages 119–128, 2001.

- [6] Sergey Brin. Extracting patterns and relations from the world wide web. In *WebDB Workshop at 6th International Conference on Extending Database Technology, EDBT'98*, 1998.
- [7] M. Bunge. Ontology i: The furniture of the world. In *Treatise on Basic Philosophy: Vol. 3*. Reidel Boston, 1977.
- [8] J. Ambite N. Ashish C. Knoblock, S. Minton. Modeling web sources for information integration. *AAAI-98*, Madison, WI, 1998.
- [9] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, pages 6–11, Menlo Park, CA, 1998. AAAI Press.
- [10] Gao Cong, Lan Yi, Bing Liu, and Ke Wang. Discovering frequent substructures from hierarchical semi-structured data. In *SDM02 SIAM International Conference on Data Mining*, 2002.
- [11] Steven Minton Ion Muslea Craig A. Knoblock, Kristina Lerman. Accurately and reliably extracting data from theweb: A machine learning approach. Kluwer Academic Publishers, Boston, 1999.
- [12] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 109–118, 2001.
- [13] M. Erdmann D. Fensel, S. Decker and R. Studer. Ontobroker: Transforming the www into a knowledge base. In *Proceedings of Knowledge Acquisition Modeling and Management(KAW98)*, pages 18–23, April, 1998.

- [14] H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *Proceedings of the 2nd Int. Workshop on Web Databases (German Informatics Society)*, 2002.
- [15] Line Eikvil. Information extraction from world wide web - a survey. Technical Report 945, Norweigan Computing Center, 1999.
- [16] O. Etzioni, A. Halevy, A. Doan, Z. Ives, J. Madhavan, L. McDowell, and I. Tatarinov. Crossing the structure chasm. In *Conf. on Innovative Database Research (CIDR-2003)*, 2003.
- [17] D. Florescu, A. Levy, and A. O. Mendelzon. Database techniques for the world wide web: A survey. In *SIGMOD Record* 27, pages 59–74, 1998.
- [18] D. Freitag. Machine learning for information extraction from online documents. pages 55–60. *IEEE Intelligent Systems* 7(2), 1996.
- [19] Tao Fu and Mengchi Liu. A gateway from html to xml. In *Proceedings of the 8th International Database Engineering Applications Symposium (IDEAS'04)*, July, 7th–9th 2004.
- [20] Minos Garofalakis, Aristides Gionis, Rajeev Rastogi, S. Seshadri, and Kyuseok Shim. Xtract: a system for extracting document type descriptors from xml documents. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 165–176. ACM Press, 2000.
- [21] N. Guarino, C. Masolo, and G. Vetere. Ontoseek: Content-based access to the web. pages 70–80. *IEEE Intelligent Systems* 14(3), May/June, 1999.

- [22] K. Ireland H. Garcia-Molina, J. Hammer. Integrating and accessing heterogeneous information sources in tsimmi. pages 61–64. AAAI Symposium on Information Gathering, 1995.
- [23] Joachim Hammer, Hector Garcia-Molina, Junghoo Cho, Arturo Crespo, and Rohan Aranha. Extracting semi-structure information from the web. In *Proceedings of the Workshop on Management of Semistructured Data*, 1997.
- [24] Chun-Nan Hsu and Ming-Tzung Dung. Generating finite-state transducers for semi-structured data extraction from the web. In *Information Systems 23(8)*, pages 521–538, 1998.
- [25] Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. In *Intelligence, vol. 118*, pages 15–68, 2000.
- [26] Nickolas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper induction for information extraction. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.
- [27] A. Laender, B. Ribeiro-Neto, A. Silva, and J. Teixeira. A brief survey of web data extraction tools. In *SIGMOD Record*, volume 31, June 2002.
- [28] Ling Liu, Calton Pu, and Wei Han. XWRAP: An XML-enabled wrapper construction system for web information sources. In *ICDE*, pages 611–621, 2000.
- [29] Mengchi Liu and Tok Wang Ling. A conceptual model for the web. In *International Conference on Conceptual Modeling / the Entity Relationship Approach*, pages 225–238, 2000.

- [30] P. Schwarz M. Carey, L. Haas. Towards heterogeneous multimedia information systems: The garlic approach. In *Proceedings of the 5th Internat Workshop of Research Issues in Data Engineering: Distributed Object Management*, pages 124–131, 1995.
- [31] A. Mendelzon, G. Mihaila, and T. Milo. Querying the world wide web. In *International Journal on Digital Libraries*, pages 54–67, 1997.
- [32] John Nerbonne, Wilbert Heeringa, and Peter Kleiweg. Edit distance and dialect proximity. In David Sankoff and Joseph Kruskal, editors, *The Theory and Practice of Sequence Comparison, 2nd ed.*, pages v–xv. CSLI, Stanford, CA, 1999.
- [33] Svetlozar Nestorov, Serge Abiteboul, and Rajeev Motwani. Extracting schema from semistructured data. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 295–306. ACM Press, 1998.
- [34] Svetlozar Nestorov, Serge Abiteboul, and Rajeev Motwani. Reverse engineering for web data: From visual to semantic structures. In *18th International Conference on Data Engineering (ICDE'02)*. IEEE Computer Society, February 26 - March 01, 2002.
- [35] Svetlozar Nestorov, Jeffrey D. Ullman, Janet L. Wiener, and Sudarshan S. Chawathe. Representative objects: Concise representations of semistructured, hierarchial data. In *ICDE*, pages 79–90, 1997.
- [36] B. L. Pellom and J. H. Hansen. Spectral normalization employing hidden markov modeling of line spectrum pair frequencies. In *Proceedings of ICASSP '97*, pages 943–946, Munich, Germany, 1997.

- [37] Dave Raggett. Clean up your web pages with html tidy. <http://www.w3.org/People/Raggett/tidy/>.
- [38] Stephen Soderland. Learning information extraction rules for semi-structured and free text. In *VLDB*, pages 1–44. IEEE, 1999.
- [39] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.
- [40] W3C. Document object model (dom) level 2 html specification, Recommendation 09 January 2003. <http://www.w3.org/TR/DOM-Level-2-HTML/>.
- [41] W3C. Document object model (dom) level 2 core specification, Recommendation 13 November, 2000. <http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/>.
- [42] W3C. Extensible stylesheet language (xsl), version 1.0, Recommendation 15 October 2001. <http://www.w3.org/TR/xsl>.
- [43] W3C. Xsl transformations (xslt), version 1.0, Recommendation 16 November 1999. <http://www.w3.org/TR/xslt>.
- [44] W3C. Xhtml: The extensible hypertext markup language, Recommendation, January 2000. <http://www.w3.org/TR/xhtml1/>.
- [45] W3C. Extensible markup language (xml) 1.0 (third edition), W3C Recommendation 04 February 2004. <http://www.w3.org/TR/2004/REC-xml-20040204/>.
- [46] Y. Wand. A proposal for a formal model of objects. In W. Kim and F. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, pages 537–559. ACM Press, 1989.

- [47] Jiying Wang and Fred H. Lochovsky. Data extraction and label assignment for web databases. In *Proceedings of the twelfth international conference on World Wide Web*, pages 187–196. ACM Press, 2003.
- [48] K. Zechner. Ia literature survey on information extraction and text summarization. Technical report, Carnegie Mellon University, 1997.