Semi-Automatic Wrapper Generation for Web Information Extraction

Liu Zehua

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Doctor of Philosophy

2007

Acknowledgments

I would like to express my sincere thanks and gratitude to the following organizations and people:

Nanyang Technological University for offering me the opportunity to pursue both my bachelor and PhD degrees in Singapore, and **Ministry of Education, Singapore** for offering me the scholarship that supported my bachelor degree study at NTU.

Associate Professor Ng Wee Keong, my thesis advisor, for his guidance and thoughtful opinions that help me throughout these years, from encouraging me to take up the Accelerated Honours Programme, to challenging me with an interesting Honours Year Project, to guiding me during the course of this PhD programme. Being his student, I am fortunate to enjoy the freedom of generating and working on my own ideas as well as to receive timely motivations and enlightening suggestions. He has taught me the way of systematically approaching a problem (not just a research one).

Associate Professor Lim Ee Peng, Head of the Information Systems Division, for the many helpful discussions on the G-Portal project and on my own research topics. It was a very fruitful experience working with him on the G-Portal project (during the first two years of my candidature) which has helped me a lot in preparing for my PhD research, especially for writing papers.

Associate Professor Hsu Wen Jing, for bringing me into his team for the CrayQuest 2000 competition when I was just a second year undergraduate student and for the trust and encouragement that he gave me. Winning the Grand Champion in the competition has

greatly burst my confidence on my programming and research abilities. It is this confidence that supports me throughout these four years of PhD study.

Li Jianzhong, then programme manager at the Institute of Systems Science, for the sharing of his rich industry experiences and the many thoughtful discussions during my work at ISS.

Li Feifei and **Huang Yangfeng**, for working together with me on the first prototype of the WICCAP project.

Ong Kok Leong, **Woon Yew Kwong**, **Sun Aixin**, **Myo Myo Naing**, **Khin Myo Win**, **Yin Ming**, **Li Zhao**, **Li Wenyuan**, **Chen Ling**, and **Yu Hai**, postgraduates in the Centre for Advanced Information Systems for the discussions and idea sharing and for their presentations during the weekly data management seminar in CAIS.

Lai Chee Keong and **Tan Lay Choo**, for their assistance in providing the administration and technical support during my stay at CAIS.

My wife, for all the cares and encouragement that she has given me. It is her patient support and wordless sacrifice that allow me to fully utilize my spare time to complete this part-time PhD study.

My brother, for his support of the family while I am away from home. Without him, I wouldn't be able to fully concentrate on my research here in Singapore.

Last but not least, **my parents**, for all the love, support and sacrifice that they have given me to pursue my interest in research. For, without the both of you, all this would not be possible.

Thank you!

Contents

List of Figures

List of Tables

Abstract

The Web has so far been incredibly successful at delivering information to various groups of people. Information sources over the WWW contain a large amount of data organized according to different interests and values. It is important to enable people to obtain their interested information in a simple and effective manner. Meanwhile, there is also an urgent need to make information from the WWW accessible to applications, in order to offer automation, inter-operation and Web-awareness among services. To achieve these goals, information from Web sources need to be extracted automatically according to users' interests. Such automatic extraction is usually performed by specialized programs, called *wrappers*. However, the creation of wrappers often requires users to have in-depth knowledge of relevant technologies and is difficult, slow and tedious for ordinary users.

In this thesis, we consider a semi-automatic approach to creating wrappers that are capable of extracting data from Web sites. Firstly, we propose a data model to describe implicit logical structures of Web sites and to define the extraction rules required to reconstruct such structure. Given an instance of the data model for some Web sites, the data can then be extracted automatically according to the defined extraction rules. We design a software tool that provides several automatic tools to help users interactively build data models of Web sites.

To further accelerate the generation of data models, we propose an algorithm to automatically derive the skeleton of a given Web site, in order to help users jump start the process of building data models. The algorithm works in a top-down manner, starting from the home page of the site. Given a page, the algorithm examines hyperlinks in groups and identifies the navigation links that point to pages at the next level in the Web site structure. The entire skeleton is then constructed by recursively fetching pages pointed by the discovered links and analyzing these pages using the same process.

Finally, as Web data is exhibiting more and more dynamic characteristics, we focus our attention on a new class of Web pages that are updated frequently but with small changes at each update. We study how to extract streams of structured records from such pages. We propose a framework to solve this problem in a divide-and-conquer manner. We focus on two of the main components in this framework, namely finding recording independent changes and finding record list boundary, and propose a sampling technique to retrieve page instances to generate training examples and algorithms to find out changes not related to the core content and to discover the boundary of the list of records in the page.

Chapter 1

Introduction

1.1 Motivation

In the past ten years, the World Wide Web has completely reshaped the Internet. It has brought the Internet from the realm of academic and computer technologists into public consciousness. It has been so successful in delivering and sharing information that people are getting used to searching the Web as the first resort of obtaining information.

As the amount of information accessible via the Web continues to grow rapidly, the weaknesses of manual browsing and keyword searching, the two conventional means of obtaining information from the Web, become more and more apparent [45]. Browsing requires users to follow links and to read (usually) long Web pages; thus, making it tedious and difficult to find a particular piece of information. Keyword searching usually returns massive irrelevant information along with some useful ones hidden in the long list of search results. Even with improved search engines such as Google that returns accurate and relevant results, a large number of Web pages are not indexable by these engines[1]. Therefore, users surfing the Web with these two conventional facilities are often inundated with too much (irrelevant) information. This gives rise to the *information overloading* problem.

As HTML Web pages are designed to be viewed by humans, most of the HTML syntax are for presentation purpose and does not contain much semantic meaning; this makes automatic access by software applications difficult. However, there is an increasing demand to turn Web data into structured and machine readable format so that further processing, such as integration, filtering and customized visualization, can take place.

One solution to the problems mentioned above is to have some software program to automatically extract target information from the Web and convert the extracted data into

[1]The size of the Hidden Web is estimated to be 500 times larger than the surface Web, as of year 2000 [9].

structured format. Some Web information extraction (IE) systems have been developed for this purpose over the past few years. These systems are usually called *wrappers* because they are actually wrapping the original Web pages to provide a different view (a more structured view).

Although wrappers seem to be an appropriate solution, generating wrapper programs itself has created another big problem. The extraction of information from Web sites requires a process of specifying *what* is the information that users are interested in, *how* this information is to be extracted, and *how* the extracted information should be organized. This process requires users to have *in-depth knowledge* about relevant technologies, such as HTML and JavaScript, and even programming skills, which ordinary users do not possess. Even for expert users, the manual process of creating a wrapper is tedious and error-prone.

In addition, since Web sites are frequently undergoing various changes, ranging from minor presentation layout modification to large scale site-level restructuring. Wrappers for these sites need to be maintained to ensure that they are functioning properly and repaired or recreated when they fail. Given the large number of Web sites for which wrappers have to be created and maintained, it is obvious that manual wrapper generation is too slow to cope with that.

To cope with the difficulty in manual wrapper generation and the large number of Web sites to be wrapped, it is tempting to have tools that can automate the process of generating wrappers. However, it is very difficult, if not possible, to make the whole process fully automated because the wrapper has to understand what the users want to extract, which would inevitably require human intervention. Therefore, it is more practical to have a solution that lies in-between the manual and automatic approaches, which is called *semi-automatic wrapper generation.*

Semi-automatic wrapper generation systems attempt to provide tools and algorithms to accelerate the wrapper generation process as much as possible. One important goal of such systems is to provide *high degree of automation.* This is related to the amount of work that the user needs to perform during the process of creating a wrapper. Since the majority of users who are suffering the information overloading problem are those who do not have much technical knowledge, making the systems *easy-to-use* is another important issue. This would typically involve the use of graphical user interface (GUI) as a mean to shield the users from the technical details. Meanwhile, Web data is semi-structured in nature and is

usually organized in hierarchical structure; related information also exist across multiple Web pages. Thus, a flexible wrapper generation system should be able to produce wrappers that can extract data at *different granularity* from a Web site as a whole, a set of inter-linked Web pages, or a small portion within a Web page.

1.2 Research Challenges

Motivated by the above, in this thesis, we investigate the issues and research challenges in using semi-automatic wrapper generation as a way to perform information extraction from the Web. In order to achieve the goals of semi-automatic wrapper generation mentioned earlier, there are many research issues to be addressed. In this thesis, we limit our study to the following research challenges. In Chapter 6, we present other challenges that we would like to address in future work.

- **What constitutes a good data model for wrappers**

  Underlying any wrapper systems is a data model that describes what data to extract and how to extract and organize it. Firstly, the data model should represent the target Web pages/sites' logical structure and what users would like to extract. Such logical structure should model users' understanding of the Web sites and be distinguished from the physical directory structure in the Web sites. Secondly, the data model should be able to express some extraction rules that describe how to extract data from the pages. Finally, the *granularity* of the targets that a data model can handle is important, because users might want to extract data only from a small portion of a large page, from a set of homogeneous pages, or from a selected substructure of a Web site.

- **How to make a wrapper generation system easy to use**

  As mentioned earlier, fully automated systems are difficult to achieve. The key point in making these systems easy to use is to automate as much as possible and to leave only those parts that really require human intervention to the users. This requires careful design in system architecture to separate and divide the relevant issues. Providing graphical user interface with a good interactivity is also important for hiding the technical details from users. When building the data model for a given Web site,

the construction of logical structure and the derivation of extraction rules are usually quite distinct tasks and should be handled separately.

- **How to accelerate model construction for large Web sites**

  Typical commercial Web sites contain a large amount of information spread over thousands of Web pages. Creating logical models for these sites is a formidable task. It is important to study what wrapper generation systems could do to accelerate this task. One possible way is to study how to automatically extract a base structure that the user could then continue to work on. This would involve determining what are available on the Web site, what the users would possibly like to extract and how to organize them.

- **How to extract information from continuously changing Web pages**

  The Web is dynamic and the information presented in a Web page is often not static. Many Web sites are powered by backend servers and databases and are subjected to changes when new content is added into/deleted from the databases. A simple repetitive extraction approach is not satisfactory because the changes in content are not detected and heavy users attention would be required after each extraction regardless of whether there are any interesting changes. One issue here is to distinguish the changes in the content from those that are not content related, e.g., advertisements. We would prefer to know the what content is added or deleted, instead of merely extracting all content each time.

1.3 Contributions

The major contributions of this thesis are summarized as follows:

- We propose the WICCAP Data Model (WDM), a data model that maps Web sites from their physical structure into commonly perceived logical views. This data model is capable of describing the hidden logical structure of multiple inter-linked Web pages and achieves a complete decoupling of the logical view from the physical structure of Web sites. To enable easy and rapid creation of such data models, we have implemented a visual tool, called the Mapping Wizard, to facilitate and automate the process of producing WICCAP Data Models. Using the tool, the time required to construct a logical data model for a given Web site is significantly reduced.

- We define and study the Web site skeleton extraction problem, which is an important step in fast construction of logical data models for Web information extraction. We propose the SEW algorithm for automatically discovering the skeleton of a Web site. Based on a recursive model, the algorithm applies a two-step process to each page that involves generating candidate link sets and selecting the best candidates as the navigation link sets. All the navigation links discovered and the pages retrieved form the skeleton of the Web site. Being able to discover Web site skeletons automatically greatly simplifies the process of creating WICCAP Data Models.

- We define and study the problem of extracting streams of structured data from Web pages that are frequently updated. We propose a framework for solving this problem that permits a divide-and-conquer methodology to be adopted. The framework consists of five steps with each step being an almost independent problem to be studied. Our main contributions lie on the first two steps where we propose a sampling technique to retrieve page instances to generate training examples and algorithms to find out changes not related to the core content and to discover the boundary of the list of records in the page.

1.4 Organization of the Thesis

The rest of the thesis is organized as follows:

- In Chapter 2, we introduce the background of Web information extraction and survey the existing works related to this research.

- In Chapter 3, we describe the WICCAP Data Model and the Mapping Wizard for creating logical data models of Web sites.

- In Chapter 4, we present the SEW algorithm for Web site skeleton extraction.

- In Chapter 5, we study the problem of extracting structured record stream and propose a framework to address this problem.

- In Chapter 6, we conclude this research and list out the future works.

Chapter 2

Literature Survey

This chapter presents the background information on Web information extraction and surveys existing work in this area. We begin this chapter with an overview of information extraction in general and extend it to the context of the World Wide Web context (Section 2.1). We then describe the techniques for Web information extraction that have been used in existing systems (Section 2.2). Finally, we end this chapter by proposing a multi-dimensional classification scheme for existing systems (Section 2.3) and summarizing the features that are necessary for developing Web information extraction systems (Section 2.4).

2.1 Web Information Extraction

Information extraction (IE) has been one of the active research areas in the past two decades. Traditionally, it has been focused on Natural Language Processing (NLP). With the expansion of the World Wide Web, the amount of information available on the Web has been growing in an incredible speed. This huge amount of online information sources has, therefore, led to an increasing research interest in the IE filed in the last few years.

2.1.1 IE Background

Information extraction is concerned with extracting relevant data from a collection of documents. It is originally the task of locating specific information from a natural language document, and is a useful sub-area of natural language processing [23]. The most important element of an IE system is a set of text extraction rules that specify the information to be extracted [71].

Unlike traditional information retrieval (IR), which merely indicates which documents need to be read by a user, information extraction systems extract pieces of information that

are salient to the user's needs. In addition, links between the extracted information and the original documents are usually maintained to allow the user to refer back to the context.

The kinds of information that systems extract vary in detail and reliability. For example, named entities such as persons and organizations can be extracted with reliability of up to 90% accuracy, but do not provide attributes, facts, or events that those entities have or participate in[1].

Extracting information manually could be a very difficult and tedious process, even for experienced users. A simple example demonstrating why manual information extraction is difficult can be found at the GATE Web site[2]. Thus, research in this field has mainly focused on building systems that perform automatic extraction of information, with as little human intervention as possible. The development of the field of information extraction has been largely motivated by the increasing amount of available online and offline textual documents. The focus on the field in the series of Message Understanding Conferences (MUC)[3] during the past ten years also helped to draw attention to information extraction research [19].

IE and IR

Information extraction is different from its more frequently mentioned counterpart, information retrieval (IR). In fact, the aims of these two technologies are rather different. While IR focuses on identifying a relevant subset of the documents from a larger collection given a user query, IE is concerned with the extraction of relevant information from within one or more documents without any keyword query. The approaches taken and technologies developed by researchers in both fields also differ greatly. IE tends to make use of rule-based system while IR has been influenced mostly by probability and information theory [29].

Nevertheless, IE and IR are often considered complementary to each other. Combining the technologies together can provide powerful tools for text processing [29]. Typically, information extraction from a large set of documents would require identifying a relevant subset first before the techniques for extracting data from each document can be applied. In this case, IR serves as a pre-processing module that helps to filter irrelevant documents for IE. At the same time, the data extracted by IE could be used by IR to enhance its keyword

[1] http://www.itl.nist.gov/iaui/894.02/related_projects/muc/info/whats_ie.html
[2] http://gate.ac.uk/ie/ie_example.html
[3] http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/proceedings_index.html

indexing and other aspects so as to improve the retrieval accuracy. Thus the distinction between the two seem to be more and more vague as the two fields are taking advantages of technologies from each other and as they extend their scopes.

2.1.2 Information Extraction from the Web

Information exists in the World Wide Web usually comes in the form of HTML Web pages. HTML syntax gives Web pages some loose structure; but such structure is mainly used for presentation purpose and sometimes may not directly provide much help for information extraction. Therefore, Web pages are often considered as *semi-structured* text, which in terms of structuredness, falls between free text (natural language documents) and structured text. Unlike free text and structured text, semi-structured text has been mostly ignored in the early years of IE research. Thus, extracting information from the Web has created a new challenge to researchers.

It should be pointed out that although web pages are usually perceived as semi-structured documents, the structuredness of a web page in fact depends largely on how the pages are created and what information the users want to extract. There are quite a number of web pages that are generated by machines and are relatively well structured; while some fancy hand-coded pages are wildly unstructured. Web pages may be perceived as well-structured, if what users want to extract is from the regular portion of the pages, such as the META tags, as done by early search engine robots. On the other hand, if an arbitrary portion is to be extracted, the pages may look rather unstructured.

Information over the Web is *dynamic* in nature. The content in a Web page might change frequently. The portion that changes within a Web page is usually the information that users are interested in and, therefore, is the information to be extracted. Another important characteristic of Web pages is the *interlinking* between documents. Hyperlinks have been explicitly incorporated into HTML syntax as an important concept to promote easy browsing of information. Such interlinking may help information extraction systems to locating relevant documents, but also poses challenges to distinguishing important and relevant hyperlinks from those unimportant and irrelevant ones.

As all these characteristics of Web documents are quite unique and have never been dealt with by traditional information extraction systems, they have posed new challenges to IE research.

2.1.3 Wrappers for Web Information Extraction

Wrappers

Information extraction from the Web is often performed using *wrappers*. One possible definition of wrapper is: "a procedure designed for extracting content of a particular information source and delivering the content of interest in a self-describing representation" [23]. In [45], wrappers refer to specialized programs that "identify data of interest and map them to some suitable format". In particular, in the context of the World Wide Web, the goal of a wrapper is to convert information implicitly stored in HTML documents into some explicit data structure for further processing. Note that the granularity of the target is not limited to a single HTML Web page; it could be just a segment of a Web page or a set of inter-linked Web pages (typically a whole Web site or a self-contained portion of a Web site).

A wrapper consists of two main components – a *data model* and an *execution engine*. The data model consists of a *logical structure* that describes what information to extract and how to organize the extracted information and a set of *extraction rules* that specify how to extract the information. The engine is a software program that is capable of performing several sub-tasks of information extraction, including fetching the target document, performing extraction according the extraction rules and converting the extracted information into the logical structure defined in the data model. In some wrapper systems [6, 50, 68], these two components are combined together by embedding the logical structure and extraction rules into the engine. The resulting wrapper for each information source is a Java class or C++ class which contains both the extraction rules and the codes. However, when the extraction rules are well defined in formal syntax, the codes for executing the extraction rules are often generic and can be generalized and separated. Wrapper systems, such as [34, 42, 62], adopt this approach by having a generic extraction module that can execute any rule of the defined extraction grammar. In such cases, the data models alone are usually considered as the wrapper.

Wrapper Generation

Although the execution engine can be generalized and separated from a wrapper, data models, especially the extraction rules, are specific to each individual information source.

Thus, for each new source (e.g. a new Web site), a logical structure and a new set of extraction rules have to be derived. This gives rise to the question of how a wrapper should be created, which is often referred to as the *wrapper generation problem.*

The definition of the wrapper generation problem varies depending on the scope of the source. In [45], the source containing the information to be extracted was restricted to a single Web page. The problem of generating a wrapper for information extraction from a Web page was then stated as follows.

Definition 2.1 *Given a Web page S containing a set of implicit objects, determine a mapping W that populates a data repository R with the objects in S. The mapping W must also be capable of recognizing and extracting data from any other page S' similar to S.* □

The term *similar* here is used in a very empirical sense, meaning pages provided by the same site [45]. The generated mapping W contains a set of extraction rules that some engine can use to extract data.

However, this definition is too restrictive, since information usually exists across multiple Web pages due to the inter-linking nature of Web documents. The definition also does not mention the logical structure of the source, which is one of the two components of the data model. Therefore, we extend the problem definition to include multiple Web pages as the target, as stated in the following.

Definition 2.2 *Given a Web information source S (a Web page or a set of inter-linked Web pages) containing a set of implicit objects organized in certain implicit structure, determine a mapping W that populates a data repository R with the objects in S and organizes the objects into some explicit structure. The mapping W must also be capable of recognizing and extracting data from S when the object instances in S change and from any other source S' having similar object structure as that of S.* □

With this new definition, the source of information now can be any set of Web pages (usually a Web site), not necessarily a single Web page. It also requires that the structure in which the objects are organized be recognized. This is important because the connections among objects in multiple pages are likely to form a hierarchical structure. Such implicit structure may be lost if the objects extracted are merely stored in a flat relation. The mapping W should also work with new sources similar to S. However, as it is not often to

find Web sites that are *similar* to each other, the definition also emphasizes on continuously extracting different data values from the same source as its content changes. This is often the case, as Web sites are dynamic in nature and are often updated regularly by changing the data content but maintaining the layout and structure of the content. In this thesis, we focus only on wrapper generation for extracting information from the same source.

2.1.4 Semantic Web and Web Information Extraction

Before going into details of the existing Web information extraction systems, it is necessary to discuss the relationship between Web information extraction in general and the efforts in the current Semantic Web initiative [10]. Information extraction in general tries to address the information overloading problem from the information consumers' side. This is appropriate in short term, because currently the information providers, mainly the Web sites, are not providing much help. In long term, when information providers help advocate the Semantic Web and add semantics (e.g. RDF[4] and Ontology[5]) to the contents that they provide, the Semantic Web will be more promising, as the information consumers and providers will be able to cooperate seamlessly to make the best out of the information. However, if the approach advocated by Web information extraction systems, especially those that provide logical data models, is well adopted and information providers start to supply logical views for their Web sites, a similar end-effect to the Semantic Web may be achieved by these systems, although the capabilities of knowledge induction and evolution are still lacking, which may have to be built into the application layer.

2.2 Web Information Extraction Systems

As the result of research in Web information extraction in recent years, some wrapper generation systems have been developed. This section surveys existing Web information extraction systems and investigates their advantages and limitations. It should be noted that the focus of this survey is on academic research projects or some commercial products rooted from research projects where detailed literatures are available. Readers are referred to [41] for a brief survey of commercial Web IE systems.

[4] http://www.w3c.org/rdf/
[5] http://www.semanticWeb.org/knowmarkup.html

Traditionally, Web IE systems are classified into manual, semi-automatic, and automatic systems. As more systems are developed and new techniques are invented, this classic classification becomes inappropriate. In this section, the surveyed systems are roughly categorized into a taxonomy based on that given in [45]. The categories are: *Language-based Systems*, *NLP-based Systems*, *Wrapper Induction Systems*, *Supervised Wrapper Generation Systems*, *Automatic Wrapper Generation Systems*, *Ontology-based Systems*, and *Wrapper Maintenance Systems*. This simple classification is by no means comprehensive. In Section 2.3, we propose a multi-dimensional classification scheme that classifies existing systems from various aspects. For simplicity of discussion, we shall stick to this simple taxonomy in this section.

Meanwhile, since each of the subsequent chapters has its own discussion on related work, we will not repeat those discussions here. Therefore, we only present a brief survey that is sufficient for demonstrating the key points and limitations of each category of systems.

2.2.1 Language-based Systems

This group of systems refers to those that rely on some specialized language to perform extraction. The focus is more on the extraction rules, where the languages of the rules are typically designed to be declarative and rather expressive and have certain primitives that are specific to the Web environment, which traditional general purpose programming languages do not have. Extraction rules usually have to be manually written by expert users familiar with the language and with HTML syntax. For this reason, language-based systems are also called *manual wrapper generation* systems. Some of the first generation wrapper generation systems, such as TSIMMIS [33, 16], EDITOR [58], Minerva [20], and W4F [68], belong to this category. Other similar systems include Jedi [36] and WebOQL [4].

Wrappers are used in the TSIMMIS [16] project to encapsulate the capabilities of information sources. Wrappers for HTML sources in TSIMMIS extract data from Web pages based on some specification files [33]. A specification file defines the objects to be extracted using variable binding and the patterns for extracting these objects. The extracted objects are stored into an OEM [64] object, with a structure implicitly derived from the specification files and the extraction process. The Object Exchange Model (OEM) is a semi-structured schema-less model, where objects within the model form a graph with a unique root.

```
PAGE AllConferences
    $AllConferences :   *<hr> ( $ConfWithInitial )+ ;
    $ConfWithInitial :  <h3><a name=''*''>$Initial</a></h3>
                        [<ul> (
                                <li><a href=''$ConfURL''> $Acronym </a>
                                $TmpConfName $TP
                              )* </ul>] ;
    $TmpConfName :      - $ConfName ;
    EXCEPTION (*(?<li>|</ul>))
                        {
                          $ConfName.reset();
                          $Acronym.cutAll();
                          $ConfName.paste();
                          $TmpConfName.cutAll();
                          $ConfName.paste();
                        }
    $Initial :          *(?</a>) ;
    $ConfURL :          *(?">) ;
    $Acronym :          *(?</a>) ;
    $ConfName :         *(?<li> --- </ul>) ;
    $TP :               {
                          $Initial char(1);
                          $Acronym char(100);
                          $ConfName, $ConfURL char(255);
                        }
END
```

Figure 2.1: A Minerva document with exception handling

Editor [58] and Minerva [20] are the formalisms used in the Araneus project [6] for the development of wrappers. Editor is a procedural language that borrows the ideas of "searching" and "cut and paste" from text editors. An Editor program consists of a sequence of instructions that could be search, replace, cut, copy, paste, and loop. In [58], it was proven that Editor programs are as expressive as *recursively enumerable languages* – even more expressive than *context-sensitive languages*. However, writing Editor programs is a non-trivial task, as it follows the procedural programming language flavor. In addition, there is no construct for organizing the extracted data, or restructured documents in this case.

Minerva was introduced to specify the data to be extracted in a more declarative manner by combining the grammar-based approach with Editor programs. It is specified in an EBNF style, as shown in Figure 2.1. To deal with potential irregularities of Web pages, each production is allowed to have an exception-handling clause that is essentially an Editor program. When the parser fails to parse a Web page with a production, the corresponding Editor program will be invoked to solve the irregularities in the page and resume the parsing. As all values are stored into relational database using a special $\$TP$ production, the logical structure of Minerva is in fact relational. To express more flexible structure, the Araneus

Data Model (ADM) [6] was proposed, where a set of heterogenous Web pages is represented by a *page scheme* that defines the structure of these pages. Minerva/Editor programs are associated with a page scheme to parse Web pages and to extract the values to construct instances of the page scheme.

The World Wide Web Wrapper Factory (W4F) [68] is a toolkit for the generation of wrappers for Web sources. The core of W4F is a language, called the HTML Extraction Language (HEL), that can be used to specify the extraction of complex structure from HTML pages. Unlike the pattern-based extraction languages used in TSIMMIS and Editor, HEL is a tree-based language that relies on the DOM tree model[6] of HTML pages. It allows two complementary types of navigation along the DOM tree: by document hierarchy for travelling along the tree structure using a path and by document flow for locating a node by its label using depth-first search. Complex structures are explicitly expressed using Nested String Lists (NSL), defined as follows:

$$NSL = null \mid NSL',$$
$$NSL' = String \mid list(NSL').$$

A suite of primitive visual tools are provided by W4F to assist users in writing HEL extraction rules. However, there are no tools or algorithms that help derive HEL rules by examples.

WebOQL [4] is a SQL-like query language based on a data model, called hypertree, that is a type-less semi-structured data model with added constructs for *ordering* and *references* (URLs). The query language is capable of locating data of interest in the hypertree and performing some transformation operations to convert the data into another format. Queries across multiple pages are possible due to the explicit inclusion of URLs and a browse(url) function.

Jedi [36] is a lightweight tool for the creation of wrappers and mediators to extract, combine and reconcile information from different sources. Wrappers in Jedi use context free grammars to describe the syntactic structure of text sources, including HTML Web pages. The extraction language also tolerates ambiguous grammars by introducing a production '.+' that automatically accepts any patterns not accepted by other rules in a disjunction.

All these systems share a similar characteristic that they provide extraction languages that are expressive enough and data structure models that are flexible enough to handle

[6] http://www.w3.org/DOM/

the various unique aspects of HTML Web pages. However, due to the over-expressiveness, they fail to provide easy-to-use tools that can significantly accelerate the process of writing extraction rules, i.e. the process of wrapper generation.

2.2.2 NLP-based Systems

As manually writing extraction rules for a wrapper tends to be difficult and time-consuming, several research efforts have focused on applying natural language processing (NLP) techniques, especially machine learning, to learn the rules from training examples given by users. Systems based on NLP techniques typically make use of syntactic and semantic constraints to specify extraction rules that locate relevant data within a document.

Traditional information extraction systems for free text [37, 67, 70, 72] focus on the problem of automatically building dictionaries of single or multi-slot rules [60] for new domains to allow easy porting of IE systems across different domains. They rely on extraction rules that are based on syntactic and semantic constraints. The expressiveness of the extraction patterns vary depending on what constraints are provided and where they can be applied. Grammatical structure is frequently used to detect relevant relationship among the syntactic constituents. Although these systems develop algorithms for inducing the extraction rules that they propose, these algorithms all require a set of (manually) annotated training examples. The required size of training corpus depends on the specific algorithms and on the desired level of precision/recall; but it is usually not a small amount. Moreover, since these systems do not provide any logical structure to organize the extracted data, the resulting data exists in isolated form or in a simple one level structure. Most importantly, in the Web context, the assumption that information sources are grammatical texts is often not valid, making it very difficult to apply these systems to extraction information from Web documents.

To apply the NLP techniques resulted from the traditional IE systems to Web documents, one approach is to preprocess these documents so that they can be supplied to NLP-based systems. With such approach, the core NLP functions do not require too much changes. However, the way that information is presented in Web documents is rather different from that of free text. There is usually not many grammatical and linguistic clues in HTML Web pages, which would cause NLP techniques to result in a very low recall ratio. The following three systems, namely WHISK [71], SRV [28], and RAPIER [14], extend the

```
DOCUMENT:
  Capitol Hill- 1 br twnhme. fplc D/W W/D. Undrgrnd Pkg incl $675.
  3BR upper flr of turn of ctry HOME. incl gar. grt N. Hill loc $995.
  (206) 999-9999 <br>

WHISK Rule:
  Pattern:: * ( Digit ) 'BR' * '$' ( Number )
  Output:: Rental {Bedrooms $1} {Price $2}

Extracted Data:
  Rental:
    Bedrooms:  1
    Price:     675
  Rental:
    Bedrooms:  3
    Price:     995
```

Figure 2.2: An example of a WHISK rule for information extraction

traditional NLP techniques and learning algorithms to make them more appropriate for extracting data from Web pages.

WHISK [71] is a system for learning extraction rules that can handle text sources ranging from highly structured to semi-structured to free text. The extraction rules used in WHISK are a variant of regular expressions. As shown in the middle of Figure 2.2, a WHISK rule consists of two main components: an extraction rule (Pattern) that specifies how to extract the data and a data organization rule (Output) that groups the extracted data instances and assigns labels to them. A form of disjunction can be expressed by defining a *semantic class*, which is essentially a set of equivalent terms. Linguistic tokens such as `'@Passive'` are also supported in the rules. The organization of extracted data is always one-level, similar to a relation. To learn the extraction rules, WHISK uses a supervised learning algorithm, which starts from the most general rule and gradually specializes it to reduce the errors on the tagged examples.

RAPIER (Robust Automated Production of Information Extraction Rules) [14] learns rules that are capable of extracting data from documents and filling the data into slots in a template. Besides exact delimiters, the extraction rules make use of limited syntactic information, such as the output of a part-of-speech tagger, and semantic class information, such as WordNet [59]. To learn extraction rules, RAPIER takes as input pairs of documents and filled templates (i.e. data to be extracted) and applies a bottom-up, specific to general search to induce the correct patterns.

In SRV (Sequence Rules with Validation) [28], the information extraction problem is treated as a classification problem where all possible fragments of the text are fed into a classifier which gives a confidence measure (or sometimes simply 'yes' or 'no') that each of the fragments is an instance of the data to be extracted. The rules of SRV are conjunctions of a set of predicates, similar to first order logic sentences. There are some 28 pre-defined features that could be used in 5 types of predicates. The rules are learned in a similar way to that FOIL [66] from a set of marked examples.

All the three systems mentioned above extend the ability of traditional IE systems to handle semi-structured documents. The types of texts that these systems handle well are those that contain both some structure (e.g. HTML tags) and fragments of grammatical texts. For texts that do not exhibit too much linguistic information, they may have difficulties extracting data. Only WHISK attempted to provide a simple one-level relation-like rule for organizing the extracted data. This may due to the traditional focus of information extraction on the learning of extraction rules of individual items instead of extracting the structure of the data. Meanwhile, the problem of tagging training examples still remain largely unsolved, although WHISK takes an iterative approach that might reduce the total number of examples to be tagged.

2.2.3 Wrapper Induction

Similar to NLP-based systems, wrapper induction systems adopt the machine learning approach to generate extraction rules. However, unlike the NLP-based systems, wrapper induction systems aim to learn delimiter-based extraction rules that do not rely on linguistic constraints. Therefore, the wrappers induced by these systems are more suitable for information extraction from the Web, as information in Web documents often exist in short fragmented text that are separated by HTML tags. Delimiters based on HTML tags are especially useful for extracting data from such documents. In addition, HTML tags usually provide certain structural information that may allow wrapper generation systems to induce not only extraction rules but also logical structures. Representative systems of this category include WIEN [42], SoftMealy [35], and STALKER [62].

WIEN (Wrapper Induction ENvironment) [42] is the pioneering work in the wrapper induction field. The extraction rules in WIEN rely on *delimiters* that immediately precede and follow a piece of data. Six classes of wrappers are defined for extracting information

from HTML pages of different structures and complexities. The most flexible class is the HOCLRT class, which defines a Head and a Tail pattern that are left and right delimiters of the region that contains a list of tuples, a Open and a Close pattern to locate each single tuple, and a set of Left and Right patterns to extract the attributes of the tuple. Essentially, this HOCLRT class provides a flexible way to extract a list of tuples. The rules are sensitive to heterogeneity and changes of the target Web pages. When an attribute is missing or the order of attributes change, the wrapper will immediately fail without extracting any of the subsequent attributes. The extraction rules are also not very expressive, even less expressive than that of the TSIMMIS wrapper [33]. How the extracted data should be organized is defined implicitly in the extraction rules. The implicit data model is essentially a set of tuples, where each tuple consists of an ordered list of attributes that may be a simple type or another tuple. This is in fact a nested-list structure that is very similar to the NSL model of W4F.

The extraction rules, i.e. delimiters of the wrappers, are learned from marked training examples. For each class of wrapper, a brute force learning algorithm, based on finding the longest common suffix, is used. The searching of candidate delimiters was done at character level and does not take advantage of HTML tags. A GUI is provided for marking training examples. Instead of marking directly on the HTML source code, WIEN allows the user to operate on the HTML view, the same view that everyone usually sees from a web browser. This may improve the efficiency of the process of marking training data, since the user is not required to fully understand the HTML source. In addition, a technique called *corroboration* is developed to automatically label examples, based on a set of recognizers and domain-specific heuristics for identifying instances of the attributes to be extracted.

The most obvious disadvantages of WIEN are that it is not able to handle missing attributes and variable ordering of attributes of a tuple. These issues are addressed by SoftMealy [35], which adopts an information extraction approach that is based on finite state transducer (FST). The extraction rules encoded in such an FST approach are more expressive than that of WIEN. Disjunction of delimiters (or tokens in this case) is permitted. The logical structure is also implicitly defined by the edges between attribute states in the FST. More heterogenous structure can be expressed, since an FST is a non-deterministic finite automaton, i.e. each state may have multiple outgoing edges connecting to other states. It is possible to express missing attribute (by having an additional edge going

```
DOCUMENT (Restaurant listings):
  <p> Name: <b> Yala </b><p> Cuisine: Thai <p><i>
  4000 Colfax, Phoenix, AZ 85285 (602) 508-1570
  </i> <br> <i>
  403 Pico, LA, CA 90007, (213) 798-0008
  </i>
  <p> Name: <i> Vita </i><p> Cuisine: Vietnam <p><i>
  523 Vernon, Las Vegas, NV 89104 (702) 578-2293
  </i>

Extraction rule (for restaurant name):
  start rule: either SkipTo(Name) SkipTo(<b>)
              or SkipTo(Name) SkipTo(<i>)
    end rule: either SkipTo(</b>)
              or SkipTo(</i>)
```

Figure 2.3: A STALKER extraction rule for restaurant name

directly to the attribute after next) and various ordering of attributes (by having multiple edges to form different routes from initial state to final state). However, an FST only describes one tuple type, which is essentially a *set* of (possibly *optional*) attributes. There is no construct available to build a nested or hierarchical structure.

To induce the FST, a simple generalization algorithm is provided to learn from a set of labelled examples. To induce a correct FST, the training examples must contain cases that have all possible missing attributes or various ordering of attributes. The resulting FST can not handle any unseen cases of missing attribute or permutation of attribute ordering.

In STALKER [62], an *extraction rule* is associated with a data item of interest. To locate a piece of data, *start rules* and *end rules* are used, similar to the left and right delimiters. Each rule may contain a sequence of SkipTo or SkipUntil and may have disjunction using the "*either or*" operator, as shown in Figure 2.3, which depicts an extraction rule that extracts the name of a restaurant. The STALKER rule language is more expressive than that of WIEN and SoftMealy. However, when compared with NLP-based Web IE systems, STALKER is strictly less expressive. For example, the above generalized rule can be written in an equivalent form in WHISK syntax as: "`* Name * <HtmlTag>`", assuming the semantic class HtmlTag is defined.

To represent the structure of the target Web pages, STALKER introduced a formalism called *Embedded Catalog* (EC) which is a tree where each node may be atomic, a tuple or a *list* of tuples. An EC can be considered as the structure of the data model that describes

the organization of the extracted data. Each node is associated with an *extraction rule* that extract the corresponding data. Child nodes under a node are considered as a set, instead of an ordered tuple. This allows STALKER to have the same properties as SoftMealy to represent various ordering of attributes and missing attributes (missing of an attribute does not affect the rest because their extraction rules are independent). In this sense, the data model of STALKER is more flexible and expressive than that of SoftMealy (no nested structure) and WIEN. Similar to that of the TSIMMIS wrapper and unlike most of other systems described above, the data model is explicitly defined.

STALKER learns a rule from a set of sequences of tokens by applying a sequential covering algorithm that iteratively generates extraction rules that cover some portions of the positive examples and add these rules into a disjunction. The experimental results showed that STALKER required less than 10 examples to obtain a 97% average accuracy over the 500 trials [62]. The tokens (training examples) are selected by the users manually. STALKER provides the user with a GUI for selecting training examples and marking up the relevant data. To further reduce the cost of labelling training examples, *selective sampling* [61], a form of active learning, is used to present only the most informative examples to be labelled.

All the above wrapper induction systems use extraction rules that are not as expressive as that of NLP-based systems. However, experiments showed that they are sufficiently expressive for wrapping most common Web information sources [62]. The main advantage of these systems comparing with NLP-based systems is that they are able to learn extraction rules for sources that contain less grammatical information and more structural clues (e.g. HTML tags). In addition, the wrappers generated are able to handle Web documents with hierarchical (except SoftMealy) and heterogenous structure. Once the labelling of training examples is done, the learning or training process is fully automatic. The accuracy of the training is usually quite high.

The assumption made by these learning systems, including wrapper induction systems and NLP-based systems that learn, is that the efforts required to label training examples are significantly less than that of manually creating wrappers, especially extraction rules. However, sometimes the labelling process may be too tedious that it overtakes the advantage that the learning process brings. Although attempts had been made to ease this process,

it is still far from an automatic labelling process. In fact, a fully automatic example labelling technique may be impossible because if we are able to correctly identify the training instances without the help of extraction rules, we can simply use the same technique to perform the extraction from all examples (i.e. the technique itself is the extraction rule). This turns out to be something similar to the chicken-and-egg problem.

Finally, these systems only handle a single Web page or a set of Web pages with homogenous structure. In particular, there is no concept of a hyperlink in these systems. The induced wrappers are not able to extract data that exist across multiple pages.

2.2.4 Supervised Wrapper Generation

Most of the system described above focus on either the expressiveness of the extraction language or the learning of extraction rules based on examples. The systems surveyed in this section, called *supervised wrapper generation systems*, have a different focus that emphasizes on providing GUI tools to support generation of wrappers interactively. Coupled with these interactive tools is some internal induction algorithm, similar to that of wrapper induction systems, for deriving extraction rules. HTML related features, such as tags and DOM tree, are more heavily used in the extraction rules. The main goal of these systems is to reduce the required human intervention as much as possible, although the wrapper generation process is not fully automatic. Due to this reason, these systems are also called *semi-automatic wrapper generation* systems. Systems in this category include NoDoSe [1, 2], DEByE [44], XWrap [50, 51], and Lixto [7, 30]. The Mapping Wizard that will be presented in Chapter 3 also belongs to this category.

As these systems will be described in details in Section 3.6, here we will only present a summary and comparison with other categories of systems.

Comparing with the induction algorithms of wrapper induction systems, the algorithms in supervised wrapper generation systems tend to work at a smaller scale (i.e. deriving rules for individual data item or a small set of items) and based on fewer number of examples (typically one example). These systems take the approach of incrementally generating wrappers by specifying or learning the extraction rules node by node. The algorithms are more subtle and heuristics-based. On the contrary, wrapper induction systems usually perform batch labelling and batch learning – training instances of all data items are labelled first before a training process takes place to learn extraction rules of all data items at one

go. The training algorithms tend to be more complex and better founded on some learning theories.

All these systems also assume that the structure in the data model is to be manually built (i.e. no automatic or semi-automatic algorithms for deriving the data model). This may be considered as a fair assumption as the structure of information is usually domain-dependent and user-dependent. However, when the target Web site is large or when we need to handle a large number of sites, manually creating the site structure would become a problem. This issue will be addressed in Chapter 4,

The functions and components provided by these systems are more comprehensive than other systems described in earlier sections. In addition to the core functions of information extraction and wrapper generation, features such as flexible page fetching mechanism, transformation and postprocessing of extraction results are also supported. This is not surprising as the main goal of these systems is to provide usable tools that can be used easily, possibly by ordinary users. Similar to most systems described in earlier sections, all systems, except Lixto, do not incorporate the concept of hyperlinks, making them only suitable for creating wrappers that only extract from a single Web page or a set of homogeneous pages.

2.2.5 Ontology-based Systems

The work of the Data Extraction Group (DEG) at Brigham Young University (BYU) proposed an ontology-based approach to information extraction from the Web [24, 25]. It focuses on documents that are data rich, narrow in ontological breadth and contain multiple records of information for the ontology. The core concept of this approach is *application ontology*, defined based on the concept of ontology. An application ontology consists of a *conceptual model* that describes the relationship between objects, i.e., the structure of the extracted data, and a set of *data frames*, describes the string patterns for the object set, i.e., the extraction rules.

Given an HTML page that contains multiple instances of a given application ontology, the approach first builds a tree (similar to a DOM tree) of the page's structure and heuristically identifies the subtree that contain all the instances. It then finds the separator that divides all these instances using heuristics [25], divides the subtree into chunks each corresponding to one instance, and applies the data frames to extract object instances. Given an

application ontology, the process of extracting information and converting it into relational database format is fully automatic.

If carefully constructed, the same application ontology may be applied to different pages of the same domain, i.e. those pages that share the same ontology. This is an unique feature, called *adaptability*, that is not seen in other systems. The data frames are also supposed to be *resilient* to changes in page layout, since most of them rely on the patterns of the internal structure of the data values, instead of left or right delimiters.

However, the construction of an application ontology is a manual process. It requires a human expert that is familiar with the target domain as well as the definition of the application ontology, including the object-relationship model and the data frames. It was reported that a simple application ontology with 19 object sets and 17 relationship sets requires 1 or 2 man-weeks of effort [24]. Moreover, to be truly adaptable and resilient, the data frames in the application ontology have to be carefully built. This usually requires examining representative examples of the target domains to derive the extraction rules that extract the correct object set instances. Therefore, the extraction rules creation process is manual, which is similar to language-based systems. In fact, it is considered much more difficult to create an application ontology than to manually create a wrapper in language-based systems, because the human expert has to examine example pages from multiple Web sites in order to derive common data frames that fit all these sites whereas users of language-based systems just need to study the pages of each site that they are interested in and the rules can be specific to each site.

Another work based on ontology is the work of Snoussi et al. [69]. Similar to the BYU DEG approach, ontologies are used to express common domain specific data models that represent information and their organization for Web pages in the same category. Ontologies are specified in a more standard way, using SOX (Scehma for Oriented-Object XML[7]), rather than using some customized syntax. The extraction patterns of data items inside an ontology are defined in separate XML documents in order to support page-specific extraction on different pages bearing the same ontology. In addition, a GUI assistant tool is provided to help users generate and fine-tune extraction patterns.

[7] http://www.w3.org/TR/NOTE-SOX/

```
<HTML>Books of:<B>#PCDATA</B>
( <IMG src=.../> )?
<UL>
( <LI><I>Title:</I>#PCDATA</LI> )+
</UL></HTML>
```

Figure 2.4: A RoadRunner Wrapper

2.2.6 Automatic Wrapper Generation

Different from all systems described above, *automatic wrapper generation* systems attempt to produce wrappers in a fully automatic manner, without any human intervention. The two systems in this category, RoadRunner [22, 31, 21] and ExAlg [3], work in a similar way by comparing different pages of the same *class* to find out a common template that can be used as a wrapper to extract data from other pages from the same classes.

RoadRunner [22, 21] is a system for automatically generating wrappers that extract data from pages of the same class, i.e., those generated by the same server-side program using the same template. It works by comparing two pages to find out the similarities and differences and using this information to construct a wrapper. The underlying extraction language, as shown in Figure 2.4, is relatively simple. `#PCDATA` denotes the data that should be extracted whereas other tags (e.g. `<B>`) and text strings (e.g. `Books of:`) serve as the left and right delimiters for locating the `#PCDATA`. Since every single character in a Web page has to exactly match one character in the wrapper, except those target data to be extracted, the extraction language can be considered as the least expressive form of language among all systems described.

The brackets and "?" and "+" symbols in the wrapper in Figure 2.4 have the same meaning as in regular expressions. They could be considered as the constructs for building the data structure of the data model. Structural disjunction is not considered, which is one of the explicit assumptions made by this system [31]. This logical structure model is similar to that of most supervised wrapper generation systems and is sufficiently expressive to capture the structure of commonly seen Web pages.

The algorithm works by comparing a wrapper (initially also a page) with a page and resolving mismatches by turning *string mismatch* into a `#PCDATA` and *tag mismatch* into an *optional* or an *iterator*. Resolving tag mismatch involves searching and trying out different

alternatives. Nested mismatch is possible when some mismatches are found during the course of resolving the previous one. Thus, the algorithm has *exponential* time complexity with respect to the size of the input pages.

One important assumption made by RoadRunner is that every HTML tag in the input pages is generated by the CGI script, thus only serves as delimiters. This assumption leads to failure to producing wrappers that extract data content containing tags, such as <P>. Another assumption made is that the wrapper is union-free, i.e. no structural disjunction. This will cause the system to fail to generate any wrapper when the input pages exhibit heterogenous structure. Moreover, string mismatch may not detect all possible data fields, since text strings are treated as tokens as a whole and are not decomposed further. For example, two stock prices with change "<B>15.7 (-0.2)</B>" and "<B>15.8 (+0.1)</B>" will be merely treated as one single field like "<B>#PCDATA</B>", where in fact the correct wrapper should be "<B>#PCDATA (#PCDATA)</B>".

Similar to RoadRunner, ExAlg [3] is an automatic algorithm for extracting structured data hidden in HTML Web pages generated from the same template. Unlike RoadRunner, which works on two pages at the same time, ExAlg deals with a set of input pages, typically much more than two. The extraction language of ExAlg uses only exact pattern matching and does not allow any wildcards in-between delimiters, which turns out to be the same as that of RoadRunner. The data structure model considered in ExAlg is similar to that of RoadRunner, with *tuple* type and *set* type (list). However, ExAlg also allows optional and disjunction by combining tuple and set types. With disjunction, more heterogeneous structure can be described. Thus, ExAlg's data structure model is more expressive than that of RoadRunner.

The basic unit is a *token* which is either a word or a tag. The algorithm is based on the observation that HTML tags that appear in all input pages with the same occurrence frequency are likely to be the delimiters that appear in the same level of the hierarchical data model. Following this observation, the *first step* of the algorithm is to cluster tokens according to their occurrence frequencies. It also distinguishes the different roles of a token appearing multiple times in a page by comparing their root-to-node paths in the DOM tree and their relative positions to a particular cluster. Tokens with different roles are treated as different tokens and the clustering is repeated until no new tokens are found. The *second step* then uses this set of clusters to construct a template (wrapper). The construction starts

from the cluster whose tokens appear exactly once in each page and recursively constructs sub-templates using the clusters occurring in-between the non-empty gaps produced by the tokens in the parent cluster. Any token belonging to other clusters occurring in the non-empty gaps of the current cluster will be considered as in the next level of the template tree. Each cluster is converted to a template (e.g., a tuple or an optional) according to a predefined set of mappings based on its type. Once the construction of the template is completed, the template can be used to extract information from pages from the same class.

The two automatic wrapper generation systems discussed above rely solely on syntactic information. This is in contrast with NLP-based systems, which make use of semantic clues heavily. As a consequence, the structure discovered by these automatic systems is a syntactic one. The wrapper generated will also extract all possible data from the Web pages, without consideration of users' preference. For example, since advertisements change over different pages, they are likely to be identified as data fields and extracted by the wrappers.

Another disadvantage is that they deal only with a set of pages of the same class. Consequently, another challenge brought by these systems is how to identify the correct pages to supply to them as input. In fact, this problem of finding the correct target Web pages also applies to other wrapper generation systems that deal only with one page. For systems dealing with information sources at site-level, this may not be a big issue, because it is much easier to identify a Web site of interest rather than to find out a Web page that a wrapper can work on.

Another issue is the annotation of the generated wrappers. The fields in the generated wrappers do not contain any semantic labels. If the labelling of fields has to be done manually, then the overall approach becomes semi-automatic again. Therefore, techniques for automatic annotation of the generated wrapper or extracted data are required to make these systems more useful in practice.

2.2.7 Wrapper Maintenance Systems

The task of information extraction is not yet over after a wrapper is generated. Due to the dynamic nature of the Web, information sources may change and, as a consequence, the syntactic and semantic clues that wrappers rely on may change as well. Wrapper maintenance systems are responsible for monitoring the performance of wrappers, detecting failures of extraction, and (possibly) repairing the failed wrappers. Two systems belonging

to this category are Rapture [43] and DataPro [46]. NoDoSe version 2.0 [2] comes with an error reporting module that detects error in data extraction. This can be considered as a wrapper verification system, which is one of the two major components of a wrapper maintenance system.

Rapture [43] addresses a subproblem of the wrapper maintenance problem, called *wrapper verification*. It is based on the standard *regression testing* paradigm, which tests a given wrapper on input pages with known output. The algorithm works by first calculating the *distribution parameters* (average and variance) of the number of tuples and some other 9 features, including *word count*, *word length*, *letter density*, etc. For the newly extracted result, the number of tuples and other feature values are computed and the probabilities of observing these values given the pre-computed distribution parameters based on a normal distribution assumption are calculated. With different assumptions on feature dependency, a combined probability could be computed, which is then used to determine whether the new extraction results are invalid. A more complex model for verifying wrappers with hierarchical structure is also proposed.

Although Rapture was developed by the same author of WIEN [42], the nine features used are considered as domain-independent and the algorithm itself does not rely on any specific feature of WIEN. Therefore, the algorithm can be applied to the verification of extraction results from any wrapper system. This is in contrast to the error reporting module in NoDoSe version 2.0, where some of the heuristics for detecting errors are specific to the NoDoSe system.

DataPro [46] is an algorithm that learns structural information about data from positive examples. It deals with one data field (in the whole tree of extraction results) at a time. Each data field is described by its starting and ending sequences. As negative examples are considered not available, the problem of learning the structure of a piece of data is treated as a *characterization problem*, instead of a classification one. The algorithm tries to find *characteristic descriptions* of the data from positive examples alone. More specifically, it finds statistically significant sequences of tokens that represent the training examples, where a sequence of tokens is considered as statistically significant if it occurs more frequently than would be expected assuming tokens are generated randomly and are independent of each other [39]. The algorithm finds statistically significant patterns by incrementally building a prefix tree, where each node corresponds to a token whose position in the sequence is

given by the node's depth in the tree. Every path starting from the root is considered as a significant pattern.

The DataPro algorithm was applied to both subproblems of the wrapper maintenance problem: *wrapper verification* and *wrapper reparation*. To verify the correctness of the extracted results, DataPro first learns m patterns that describe known results. It then compares the number of training examples and extraction results that match each pattern. The extraction results are considered correct if the numbers are the same based on some significance level. For wrapper reparation, DataPro assumes that the wrapper could be learned by a machine learning algorithm, such as STALKER and WIEN, given some training examples. It finds all text segments from each new page that match the learned patterns and clusters them based on a few features such as their relative positions in the page and adjacent tokens. The cluster that is most similar to the known extraction results is chosen as the training examples to reinduce the wrapper.

The experiments of both systems showed relatively high precision for the wrapper verification task. At a first glance, it seems that the features or patterns obtained may be used to directly extract data from the Web pages, using a classification approach. However, direct application of such characteristic description to extract data may result in a large number of values extracted, which corresponds to a high rate of false positive. This is certainly not desirable. The clustering approach used by DataPro is one effort towards removing the incorrect data values. However, DataPro does not consider the whole data field (only starting and ending patterns), which may cause it to learn patterns that only partially describe the field. It also has limitation when applied to data fields that consist of very few number of tokens. For the layout of Web pages or structure of Web sites change, DataPro will fail to reinduce a new wrapper.

The approach to generate training examples for reinducing a wrapper can also be applied to learning wrappers for new sites. This might be feasible if the sites are in a similar domain and the data fields share similar characteristics.

Although the techniques proposed by Rapture and DataPro were a good start towards solving the wrapper maintenance problem, other issues such as reuse of previous extraction rules and larger scale of changes with page layout and site structure have not yet been addressed. Therefore, this area still remains open to new research.

2.3 Classification Schemes for Web IE Systems

The simple categorization that we used to discuss the various wrapper generation systems does not reflects all the important characteristics that distinguish the systems from each other. In fact, due to the diverse techniques used by the existing systems and the different purposes that these systems have been developed for, it is rather difficult to have a one-size-fit-all classification scheme that categorizes all existing systems appropriately. Therefore, we propose a multi-dimensional scheme that attempts to characterize these systems from different aspects. Due to the difference in the goals, wrapper maintenance systems are not included in this discussion. For comparison purpose, we also include the WICCAP system [47, 56, 55, 53, 54], which is a Web information extraction system where the research in this thesis is part of. The classification of the systems is summarized in Table 2.1

- **Degree of Automation:** This is the dimension that is used to categorize the systems in the discussion in last Section. It is also the dimension that is most commonly used to evaluate the efficiency and usability of a wrapper generation system, because the degree of automation directly affects the speed of constructing a wrapper. We classify the systems into: MAN (manual), ML (machine learning), SWG (supervised wrapper generation), and AUTO (fully automatic). Note that we group NLP-based systems and wrapper induction systems under a common category ML. This is because these systems share an important characteristic that they require labelling of training examples followed by a training process using the examples. They are considered as semi-automatic. The BYU DEG's work is considered as manual because the creation of the application ontology, which is essentially the extraction rules, is manual.

- **Type of Data Source:** Except the early NLP-based systems, denoted as GT (grammatical text), most systems deal with HTML Web pages. However, their reliance on HTML tags vary depending on the specific techniques that they use. For wrapper induction and later NLP-based systems, HTML tags are still treated as plain text strings. We call them PT (plain text) systems since they do not rely too much on the grammatical information nor on the HTML tag hierarchy. For most supervised and automatic wrapper generation systems, either the HTML DOM tree is used or the derivation of extraction rules make use of HTML tags as an important clues. These systems are denoted as HTML.

Table 2.1: Classification of Web IE systems

Systems	Degree of Automation	Type of Data Source	Scope	Data Model	Visual Support
TSIMMIS	MAN	PT	MP	TREE	NO
Minerva	MAN	PT	MP	OO	NO
W4F	MAN	HTML	SP	TREE	ITR
WebOQL	MAN	HTML	MP	TREE	NO
Jedi	MAN	PT	SP	TREE	NO
WHISK	ML	GT/PT	TS	FLAT	LB
RAPIER	ML	GT/PT	TS	FLAT	NO
SRV	ML	PT	TS	FLAT	NO
WIEN	ML	PT	SP	TREE	LB
SoftMealy	ML	PT	SP	FLAT	LB
STALKER	ML	PT	SP	TREE	LB
NoDoSe	SWG	PT	SP	TREE	ITR
DEByE	SWG	HTML	SP	TREE	ITR
XWrap	SWG	HTML	SP	TREE	ITR
Lixto	SWG	HTML	MP	TREE	ITR
BYU DEG	MAN	PT	SP	OO	NO
Snoussi et al.	SWG	HTML	SP	OO	ITR
RoadRunner	AUTO	HTML	SP	TREE	NO
ExAlg	AUTO	HTML	SP	TREE	NO
WICCAP	SWG	HTML	MP	TREE	ITR

- **Granularity of Scope:** Early NLP-based systems focused on text segments each containing a single event or name entity of interest. Later systems developed techniques that can be used to extract a list of tuples. A few recent systems recognized the importance of dealing with interlinked Web documents and explicitly introduced hyperlinks in their systems. According to the granularity of the information source, we classify the systems into: TS (text segment), SP (single page or a set of pages with homogeneous structure), and MP (multiple interlinked pages or a whole Web site).

- **Data Structure Model:** A large variety of data structure models are used to represent the information sources and to store the extracted results. Systems that deal with text segments often extract a tuple of attributes using either multi-slot rules or single-slot rules in conjunction with simple one-level data model. We denote these systems as FLAT. Most recent systems allow semi-structured, tree-like data models (e.g. Embedded-Catalog and NSL) that capture the hierarchical structure of the information. We call them TREE. We consider the data model in TSIMMIS wrapper as

TREE since it does not make use of the full graph model of OEM. A few other systems use data models that are of object-oriented favor by allowing a complex graph model with relationships among object classes. We term this last category as OO. For the Araneus Data Model used together with Minerva and Editor, the extracted data is stored in relational database. However, the data model represents information in an OO manner. Thus we consider this model as OO.

- **Visual Support:** Providing visual tools to support the information extraction process is a critical factor that makes the IE systems easy-to-use. We classify the systems into: No (no visual tools provided), LB (tools for labelling training examples), and ITR (interactive tools for wrapper generation).

2.4 Summary

In summary, there are several important features that a Web information extraction system should have.

For a Web information extraction system, there are two main components: the wrappers themselves and the wrapper generation tools. The definition of a wrapper consists of a logical structure and an extraction language. It is important that the logical structure is hierarchical and is able to represent information available across multiple pages. The extraction language behind the extraction rules should have an expressive power at an appropriate level – expressive enough to extract data from HTML pages but not too complex to restrict the derivation of rules from examples.

Manual creation of wrappers has been proven to be difficult. The degree of automation should be the main concern of wrapper generation tools. Higher degree of automation is typically achieved by extraction rules derivation algorithms and interactive visual tools. Although automatic generation of extraction rules might be achieved at page level, the generation of logical structures still remains as an important challenge.

In the following chapters, some of the directions mentioned above will be pursued.

Chapter 3

Building Logical Views of Web Sites

3.1 Introduction

As the first step to creating a wrapper, defining data models for mapping between the actual information on the Web pages and the pieces that the users are interested is important for a wrapper generation system. This process usually requires in-depth technical knowledge and, if done manually, is difficult and time-consuming. In addition, the rate at which new Web sites are appearing and old Web sites are changing is much faster than what the current IE systems can handle, even if expert users are the target users. Therefore, helping users quickly create the required data model for extracting information has become the top priority of IE system for the Web.

Information presented in a Web site is usually organized according to certain logical structure that corresponds to users' common perception. If the data models that IE systems produced reflect this commonly perceived structure, most users will be able to understand and use them easily. Unfortunately, the data models of most current IE systems [1, 7, 21, 44, 50, 62, 68] either only deal with information in page-level or represent the extracted information in some proprietary and ad-hoc data structures that do not truly reflect the original logical view of the Web sites.

3.1.1 The WICCAP Approach

In this chapter, we propose a data model, called the WICCAP Data Model (WDM), to map information from a Web site into its commonly perceived organization of logical concepts. WDM is designed to model information from the Web at site-level so that the data extracted

according to this model represents a complete view of the original site and no additional efforts are required to further integrate data from different pages. The aim of WDM is to enable ordinary users to easily understand the logical structures of a site in order to perform extraction tasks without worrying about the technical details. WDM is proposed as part of the *WWW Information Collection, Collaging and Programming* (WICCAP) system [56], which is a Web-based information extraction system to enable people to obtain information of interest in a simple and efficient manner.

The approach taken by WICCAP is to explicitly separate the tasks of information modeling and information extraction so as to allow ordinary users to perform information extraction. In WICCAP, expert users construct the WICCAP Data Model and specify how to extract the information; ordinary users indicate what to extract based on the given WDM and use the tools provided by the system to extract and view the information.

Besides being easy-to-understand and easy-to-use to ordinary users, WDM is designed to be *flexible* enough to work with heterogeneous categories of Web sites and *open* enough to allow other systems and applications to understand and make use of it.

To cope with the rapid changes of old Web sites and establishment of new Web sites, a software tool, called the Mapping Wizard, has been implemented to assist users to quickly generate the data model of a given Web site. The focus of the Mapping Wizard is on how to quickly generate a logical data model for a given Web site. To do this, we formalize the process of creating a WICCAP Data Model for a given Web site to permit as much automation as possible and develop assistant tools and algorithms to help automate the process of creating data models.

3.1.2 Outline of Chapter

The remainder of the chapter is organized as follows. Section 3.2 elaborates the logical data model in detail. Section 3.3 then presents the process for creating such data model. Section 3.4 describes how the tool helps users to quickly generate data models. Some experimental results are presented and discussed in Section 3.5. Section 3.6 compares our work with related systems. Finally, Section 3.7 gives some concluding remarks.

3.2 WICCAP Data Model

In most current Web Information Extraction systems, when defining data models, users usually have to directly specify which portion of a Web page within a Web site constitutes

the target information. The isolated pieces of target information are later re-organized to make them more readable and accessible. As a result, the data model produced is usually not intuitive to other users and is very specific to the particular Web site.

In WICCAP, we try to derive a logical data model (WDM) of the target Web site first and then extract information from the Web site based on this model. The role of the WICCAP Data Model is to relate information from a Web site in terms of a *commonly perceived logical structure*, instead of physical directory locations. The logical structure here refers to people's perception on the organization of contents of a specific type of Web site.

For example, when a newspaper Web site, such as BBC Online News [8], is mentioned, people generally think of a site that consists of sections such as World News, or Sports News; each section may have subsections and/or a list of articles, each of which may have a number of attributes such as title, summary, the article itself, and maybe links to other related articles. This hierarchy of information is the commonly perceived structure of a newspaper Web site, which most users are quite familiar with. If we model and organize the information from a newspaper Web site in this way, the resulting data model will appear familiar to and be easily accepted by most users. In addition, since this logical structure is applicable to the entire category of Web sites, e.g. all newspaper Web sites, it can be re-used in the future when creating data models for Web sites of the same type.

The logical model is essentially a tree, where each node represents a logical concept perceived by the user. The organization of individual tree nodes into a tree forms the logical skeleton of the target Web site. This logical structure is what the users of the next layer in WICCAP (for information extraction) will be operating on. In order to allow extraction to occur, each tree node contains an element named "*Mapping*" (discussed in detail in Section 3.2.3) that associates the node with the corresponding data in the actual Web site. This *Mapping* element in fact represents the extraction rule(s) that enable the extraction agent to perform the actual extraction of information. For leaf nodes of the tree, the *Mapping* element points to the data item that the user wants to extract. For example, a leaf node "Title" may have a Mapping element that contains the extraction rule to extract the title of the news from the Web page. For internal nodes, the Mapping element refers to the portion of the Web site that contains the desired data to be extracted by all the leaf nodes of the subtrees rooted from those internal nodes. By assigning a Mapping element to each logical node, the logical view of the target Web site can be decoupled from the physical

Figure 3.1: Logical View of BBC Online News

directory hierarchy and page layout of the original Web site, while the mapping between the two is still maintained.

An important feature of WDM is that it is able to model not only information in a single Web page, but also in a set of Web pages. This set of Web pages may or may not have similar layout structures in terms of HTML syntax. It could be a small collection of Web pages located in the same directory, a whole Web site, or pages across several Web sites, so long as all pages together form a uniform logical view. Typically, WDM will be used to model a Web site or a self-contained portion of a large Web site.

Figure 3.1 shows a portion of the logical view of BBC Online News. In this model, we have a root node, under which there are different sections, such as "World" and "Business". There may be subsections under each section. Each node (Section or Subsection) may have a list of articles, denoted by the node "ArticleList" with a child node "Article". Attributes and contents of the article are represented as child nodes of "Article". The mapping information is stored as an attribute of the elements to hide the technical details from ordinary users and to maintain the logical overview of the data model.

3.2.1 WDM Schema and Mapping Rules

A WDM representation of a specific Web site is also called a mapping rule, since it is to map the physical structure to the logical one. Each mapping rule represents a logical tree with one Mapping element for each tree node. Every mapping rule has a schema called WDM schema. A WDM schema defines the possible structure that a mapping rule could have. Mapping rules are instances of their WDM schema. The relationship between a WDM schema and a mapping rule is similar to the Class-Object relationship in object-oriented

concept. A WDM schema is defined using XML Schema [73], while mapping rules are defined as XML document. This makes mapping rules well-structured, interoperable, and easy to process.

The WDM schema was first designed to describe the logical structure of a category of Web sites that exhibit similar logical view [53]. We have extended and generalized the definition of WDM schema to make it suitable for representing the logical view of any Web site. Thus there is now only one global WDM schema that defines the syntax that all mapping rules should follow. In the remainder of this chapter, this global WDM schema is referred to as *the* WDM *schema.*

All elements defined by the WDM schema can be classified into two groups: elements for logical modeling purpose and elements for mapping purpose. The former refers to the elements constituting the logical tree, whereas the latter refers to the group of elements defining the mapping between the logical element and the actual Web data to be extracted.

3.2.2 WDM Elements for Logical Modeling Purpose

There are four elements for logical modeling: *Item*, *Record*, *Region*, and *Section*. These elements serve to represent some logical concepts in a mapping rule. Each element corresponds to a logical node and all elements in a mapping rule together form the logical tree. The actual semantic meaning of each element in different mapping rules depends on the Web sites. This semantic meaning can be partially reflected by setting the values of attributes such as "Name" or "Desc". Each element also contains a Mapping element that describes how to link the element to the corresponding portion on the Web site.

Item

An *Item* represents a piece of information that the user is interested in. It is the atomic unit of all the information in the logical data model. In the logical tree, Items are represented as leaf nodes. An Item contains a Mapping element that indicates where is the data to be extracted, and other attributes that describe its semantic meaning. In a mapping rule, only the data extracted according to Items will be eventually stored as the extraction results. The data extracted from other elements (internal nodes) is only treated as intermediate results to be worked on for processing of subsequent extraction rules.

Record

In most cases, we do not just specify a single Item to extract in an entire logical tree. Within a Web site, it is likely that there will be a lot of Items that the users would like to extract. Typically, some data will appear to be logically related to each other. Thus, we introduce the *Record* element to group several related Items together.

A Record contains a set of Items, or a tuple of attributes. It is usually used to repeatedly retrieve data in a table or a list when its "Repetitive" attribute is 'True'. It makes WDM more flexible and expressive by allowing it to describe iterative structure, which appears frequently in complex logical representations. As with other main elements, there is a Mapping element to map the Record to the actual Web pages where all the data specified by the Items exist.

An example of Record is the Article element in the BBC News mapping rule, with the different attributes of the article, such as Title, Link, Short Description and Content, forming the group of Items under this Record. This article record is iterative since there can be more than one article under each section.

Although Items within a Record are listed in certain order, the Mapping element of each Item (i.e. the extraction rule) is independent of each other's. This allows Record to handle some variation of the target Web page, such as missing Items and varying-order of Items.

Region

A *Region* is defined to help identify the area where an iterative Record repeats itself. A Region is typically used together with a "Repetitive" Record to repeatedly extract a list of Records. It can be considered as the rough area containing a list of repeated set of data that we are interested in the Web site.

Section

A Record corresponds to a tuple while a Region represents a list. They are only able to model Web sites with flat structure (i.e. one level). To make WDM more flexible, a *Section* element is provided to allow nesting of elements, which eventually enables WDM to model a more general form of hierarchical tree-like structure that most existing Web sites exhibit.

A Section is defined to contain a set of Regions together with a set of nested Sections. Both the Regions and the Sections are optional and can interleave each other. But for the

Section element to be meaningful, it should not be empty. One example of using Section to create hierarchical structure is depicted in Figure 3.1, where the Section "World" contains an "ArticleList" (Region) and two subsections "Asia-Pacific" and "Americas".

The root node of the logical tree is in fact just a Section element. However, to distinguish it from other Sections, we have decided to give it a different name as *Wiccap*. Since the Wiccap element is just a Section element with a different name, we shall not treat it as a new type of element.

With the introduction of Section, we extend the definition of Record to include Section as possible child elements of a Record. This means that a Record will have a list of disjunctions of Item or Section. This will allow WDM to handle the situation where a record may contain a list of repeating sub-items, for example, a news article with a list of related news.

3.2.3 WDM Elements for Mapping Purpose

With all the four logical elements, the logical view of a Web site can be composed. However, this is just a representation of users' perception of the structure of the Web site. It does not contain any instruction that allows the extraction of data to occur. To make the extraction possible, we defined a set of elements dedicated for mapping from the logical tree node to the corresponding portion within the actual Web site. These elements are the basic constructs of the extraction rules used in the WICCAP Data Model.

The WDM elements for mapping purpose include: *Locator*, *Link*, *Form*, and *Mapping*.

Locator

A *Locator* is the fundamental element in WDM. It locates a portion within a Web page. This element corresponds to a single-slot extraction rule (in information extraction terminology) [60]. The current WICCAP system only implements a ByPattern Locator that relies on the starting pattern and ending pattern to locate a specific portion that resides between these two text patterns. The expressive power of such extraction language is obviously restrictive. However, it should be pointed out that the more complex the extraction language is, the more difficult it is for users to manually write and for tools to automatically derive the rules. This is the main reason that only a simple "ByPattern" language is chosen in our data model.

Although the current implementation uses the simplest type of extraction language, the definition of the WICCAP Data Model does not exclude the usage of other more powerful

```
 1:          <xsd:complexType name="LocatorType" content="elementOnly">
 2:              <xsd:choice>
 3:                  <xsd:element name="LocatorPattern" type="LocatorPatternType"/>
 4:                  ... ...
 5:              </xsd:choice>
 6:              <xsd:attribute name="Type" use="required">
 7:                  <xsd:simpleType base="xsd:NMTOKEN">
 8:                      <xsd:enumeration value="ByPattern"/>
 9:                      ... ...
10:                  </xsd:simpleType>
11:              </xsd:attribute>
12:          </xsd:complexType>
13:          <xsd:complexType name="LocatorPatternType" content="elementOnly">
14:              <xsd:sequence>
15:                  <xsd:element name="BeginPattern" type="BeginPatternType"/>
16:                  <xsd:element name="EndPattern" type="EndPatternType"/>
17:              </xsd:sequence>
18:          </xsd:complexType>
19:          <xsd:complexType name="BeginPatternType" base="xsd:string">

                 ⋮  ⋮  ⋮
```

Figure 3.2: Definition of the Locator Element

languages. Since Locator is the only mapping element that contains the extraction rules, it is possible to import external extraction languages into the definition of Locator, while keeping the rest of the data model untouched. This means that more expressive languages, such as STALKER [62], HEL [68], Elog [7] and context-based languages [5], could potentially be included, if the corresponding implementation is available. But the derivation of such rules within the framework of the Mapping Wizard tool (discussed in Section 3.4) remains to be investigated.

With a Locator, we are able to retrieve any part within a given Web page, regardless of whether it is a link, attribute, element name, or text. This is the most fundamental functionality that any information extraction system should have. The definitions of Locator and some of its relevant children are depicted in Figure 3.2, where a Locator is defined to contain a *LocatorPattern* (line 2–5), which in turn contains two elements (*BeginPattern* and *EndPattern*) (line 13–18) that define the left and right patterns enclosing the target data item to be extracted (see line 19 for the definition of BeginPattern). The LocatorPattern element is put inside a choice under Locator (line 2–5) because in our initial design, a Locator may be of other types, such as *ByPath* (i.e. using DOM-tree based extraction patterns). But the current implementation only handles the ByPattern type of Locator. The same reason applies to the Type attribute of Locator (line 6–11), where the set of possible attribute values could include other values (e.g. ByPath).

```
 1:          <xsd:complexType name="LinkType" content="mixed">
 2:              <xsd:choice>
 3:                  <xsd:element name="Locator" type="LocatorType"/>
 4:                  <xsd:element name="Form" type="FormType"/>
 5:              </xsd:choice>
 6:              <xsd:attribute name="Type" use="required">
 7:                  <xsd:simpleType>
 8:                      <xsd:restriction base="xsd:NMTOKEN">
 9:                          <xsd:enumeration value="Dynamic"/>
10:                          <xsd:enumeration value="Form"/>
11:                          <xsd:enumeration value="Static"/>
12:                      </xsd:restriction>
13:                  </xsd:simpleType>
14:              </xsd:attribute>
15:          </xsd:complexType>
```

Figure 3.3: Definition of the Link Element

Link

The ability to extract information across multiple Web pages is critical to WDM, since it models information at site-level. A *Link* element corresponds to the concept of a hyperlink in the Web context. It gives WDM the ability to follow hyperlinks in Web pages; thus, traversing the entire Web site. Isolating the concept of a link out of the extraction language definition is one of the distinguishing features of the WICCAP Data Model, since most Web data extraction systems either can not handle multiple pages or have the "linking" instruction embedded inside their extraction languages.

When a Link appears in a mapping rule, it indicates that there is a hyperlink in the actual Web page that should be followed. When the extraction agent follows a Link in a mapping rule, it attains the same effect as a user clicking on a hyperlink in a browser and being brought to the new page pointed by that hyperlink.

As shown in the definition in Figure 3.3, a Link can be *Static*, *Dynamic* or a *Form* (defined in line 9–11). A static link is just a simple fixed URL given by users when they create the mapping rules. It is typically used to indicate the base address of a mapping rule. For example, in the root element of the BBC mapping rule, we specify a static Link with a value "http://news.bbc.co.uk/". Although not listed in the choices in line 3–4, a static link is just a simple text string and is allowed in XML syntax by defining the content attribute to be mixed (line 1). A Dynamic type of Link's link value must be extracted by the extraction agent at run-time. The Link element contains a Locator that locates the hyperlink to be extracted. The agent will follow this link to reach another Web page to

continue the extraction. This Dynamic type of Link is critical to a flexible and reliable extraction language because the links in a Web page are likely to change while the positions where these links appear remain relatively stable. A good example of dynamic Link is the links of news headlines that point to the detailed news.

The third type of Link is to be generated from HTML forms. The *Form* element caters to a special group of HTML tags: FORM and other related HTML elements. Form is defined as a type of Link because a FORM ACTION causes the server to perform some operation and return another page; this, to the users, has the same effect as clicking on a hyperlink. Typical examples where the Form element may be useful are the search function and user login authentication.

When the type of a Link is Form, the URL is dynamically constructed by concatenating all the name and value pairs of the input elements contained in the specified form. The input parameters are obtained from the user by dynamically constructing a form that assimilates that of the original HTML form. The extraction agent then posts this composed URL to the remote Web server and obtains the returned Web page. This ability to handle HTML forms is important for a Web IE System due to the extensive use of forms in Web sites. It is also one of the characteristics that distinguish WDM from other similar systems, as currently only few of these systems [6, 68] take forms into consideration or directly incorporate them into their systems.

Mapping

Locator allows us to access anywhere within a single page while *Link* enables us to jump from one page to another. The *Mapping* element combines these two basic elements to allow us to navigate throughout an entire Web site, or even the whole World Wide Web.

A Mapping is a simple container element that holds either a Link, a Locator or both. A Mapping may also contain a child Mapping element. The definition of the Mapping element can be seen as a higher level extraction language based on the primitive single-slot language provided by Locator and the hyperlink concept given by Link.

The recursive definition of Mapping allows WDM logical elements to jump through multiple levels of links and essentially makes it possible to completely separate the logical structure from the physical structure defined by the hierarchy of links and physical directories. With the Mapping element, it is not necessary that a child logical element (e.g.

Region) must be contained within the area on the same Web page as defined by its parent logical element (e.g. Section). The child element can actually be several clicks of hyperlink away from the parent element.

Combining the definition of all the logical and mapping elements described above, we obtain the complete WDM schema. All the mapping rules conform to this WDM schema. In a mapping rule, every logical element, which is shown in the logical tree view as a node, should have a Mapping element to indicate how to get to this node in the actual Web site. When each logical element that constitutes a logical structure is augmented with a Mapping element, this logical structure is considered as a complete mapping rule that can be used for extraction of information.

3.2.4 An Example on BBC Online News

In this section, we illustrate with an example of how the WICCAP Data Model translates the physical structure of a Web site into a logical view. The Web site that we will be looking at is the BBC Online News [8]. The mapping rule for this Web site is shown in Appendix A. Due to the limitation of space, only details of one Section and one subsection of the mapping rule are shown. The Section element here represents a newspaper section such as "World News". Region and Record correspond to ArticleList and Article respectively. With such semantic meaning associated with each logical element, the schema actually represents the logical view of most of the online newspaper Web sites.

In the root element, the XML Schema file is specified; this is also the file that contains the definition of the WDM schema. The Mapping of the root element Wiccap is a static Link that points to "http://news.bbc.co.uk". This is the case for most mapping rules because a Web site needs to have a base URL or home URL for any navigation to start from.

Under the root node "Wiccap", there are a few Sections. In this example, we specify "World", "Business" and so on. Again, the "World" Section has a Mapping, which extracts the link to the page containing the actual "World" section on the Web site. An ArticleList (Region) is included under the "World" Section to indicate that we are interested in some content in this section. The Article (Record) and Item elements under this ArticleList node give details of the contents to be extracted. In this subsection, for each article (the Record whose attribute Name is valued *Article*), there are three Items specified: the title, the short description of the article and the URL to the full article.

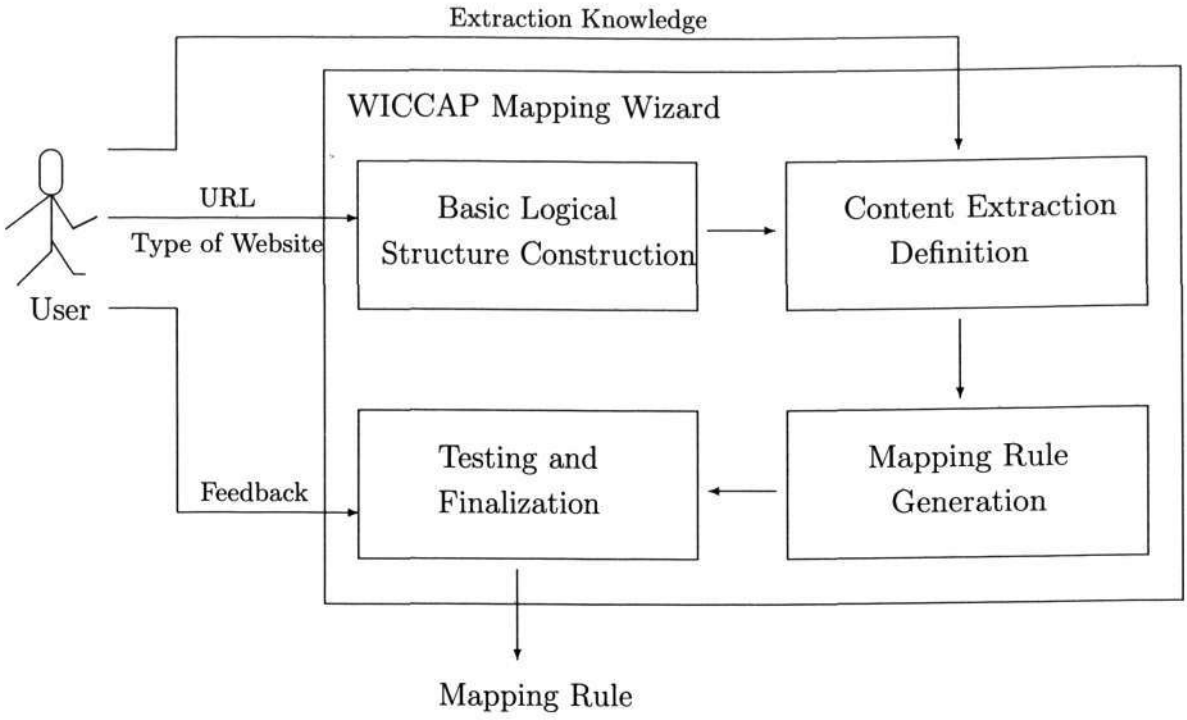

Figure 3.4: Formal Process of Mapping Rule Creation.

Two Sections "Asia-Pacific" and "Americas" are inserted after the ArticleList. The internal definition of these two subsections are similar to the Section "World". All the Sections and ArticleLists together form a tree structure of the logical view (Figure 3.1), which is quite close to our common understanding of a newspaper: a hierarchy of sections and subsections with articles in each section.

A sample of data (in XML format) that is extracted using this mapping rule is given in Appendix B.

3.3 WDM Creation Process

The manual process of creating any data model for Web information sources, including the WICCAP Data Model, is tedious and slow. To cope with the large number of new and changing Web sites, we formalize the process of creating mapping rules with the help of the Mapping Wizard.

The formal process of mapping rule generation consists of four stages: *Basic Logical Structure Construction*, *Content Extraction Definition*, *Mapping Rule Generation*, and

Testing and Finalization. Figure 3.4 illustrates the overall process of producing a Mapping Rule using Mapping Wizard.

Basic Logical Structure Construction. In this stage, the user focuses on building the overall logical structure instead of the details of how to map the logical tree nodes into the actual Web site. To start constructing the logical view of a Web site, a user supplies the home URL of the target Web site, such as "http://news.bbc.co.uk" to the Mapping Wizard. It then figures out the basic logical organization either by using the user's knowledge and understanding of the Web site or by navigating the Web site. A logical tree can be constructed with the understanding of the organization of the content provided by the Web site. In the example in Appendix A, this step would involve browsing the BBC Web site and creating a tree that contains World, Business, etc as Sections at the first level and Asia-Pacific, Americas, etc as Sections under the World Section. The Regions, Records and Items under each Section would be created as well.

The basic logical structure built in this step consists of the WDM elements for logical modeling purpose. As we mentioned earlier, Web sites of the same category share similar logical structure. Although currently not implemented in the Mapping Wizard, it is possible to re-use (at least part of) the basic logical structures constructed for other Web sites by retaining the relevant WDM elements for modeling purpose in a mapping rule generated earlier for a similar Web site. With some modification effort, the user can proceed to the next step to specify the mapping details.

Content Extraction Definition. After the basic logical tree has been clearly defined, the user continues with the *Content Extraction Definition* stage to define specific information that is of interest and relates the nodes in the logical tree to the actual Web site. For instance, the Mapping elements (shown in Appendix A) for each element created at the step earlier (e.g. the World Section element and the Item elements) would be discovered and defined at this step.

As described earlier, the Mapping element uses Link and Locator to eventually point to certain portion of a Web page. To specify the Mapping element, it is the same as figuring out the patterns that enclose each piece of information. The Mapping Wizard provides a GUI and tools to assist the users so that the manual efforts required for this step can be reduced as much as possible.

Mapping Rule Generation. Once all the information is confirmed, the Mapping Wizard generates the mapping rule according to the tree structure and all the corresponding properties. This process is fully automated. The Mapping Wizard validates the information that has been specified against the syntax defined in the WDM schema and produces the mapping rule.

Testing and Finalization. In this stage, the user tests the generated mapping rule to see whether it represents the correct logical structure and whether it locates all the required information. The testing is a trial-and-error process, similar to the process of debugging a program. The Mapping Wizard performs the actual extraction using the generated mapping rule and returns either the retrieved information or error messages if any error occurs during the retrieval process. The user verifies whether the returned information is what he or she originally wants to extract.

The mapping rule is finalized once the user is satisfied with the information retrieved from the Web site. The final version can either be stored in some mapping rule repository or be delivered to users of the extraction agent for the extraction of information.

3.4 The Mapping Wizard

The current version of the *Mapping Wizard* is implemented using Visual C++ 6.0. This section discusses the GUI of Mapping Wizard and how it helps users to quickly generate Mapping Rules.

3.4.1 Graphical User Interface

Figure 3.5 shows the main GUI of Mapping Wizard. The main components in this User Interface are the *Mini Web Browser*, the *Logical Tree Editor*, the *HTML Source Viewer*, and the *Property Dialog*.

Mini Web Browser. The Mini Web Browser on the right panel is a small Web browser for users to navigate the whole Web site. It has nearly all the functions of a normal Web browser. Most assistant tools rely on the Mini Web Browser to perform certain operations. For example, to use the Pattern Induction Tools, the user first highlights the target information on the Mini Web Browser and then activates the induction tool to discover the Mapping properties. The use of this browser instead of other external Web

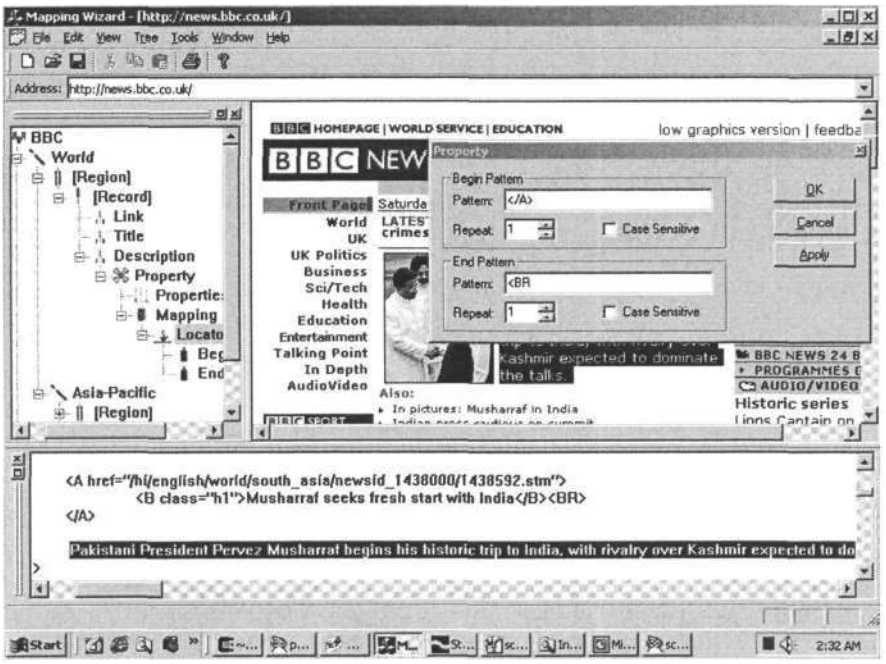

Figure 3.5: Main GUI of Mapping Wizard

browsers is quite critical because the integrated features that rely on the Mini Web Browser can significantly improve the speed of the whole process.

HTML Source Viewer. The HTML Source Viewer displays the HTML source code of the Web page that the user is viewing in the Mini Web Browser. It is usually either located at the bottom of the main window or it pops up as a floating dialog window. Using the HTML Source Viewer enables users to view and analyse the HTML source directly. The assistant tools usually allow users to specify target information in two ways: on the Mini Web Browser or the HTML Source Viewer. The use of the HTML Source Viewer may yield higher accuracy in selecting the desired text string but it also requires relevant knowledge such as HTML and JavaScript.

Logical Tree Editor. The panel on the left is the Logical Tree Editor. The logical model of the Web site that the user is modeling is presented as a tree in this dialog. The editor allows users to perform normal tree manipulation on the logical tree, including insertion and deletion of tree nodes and modification of their properties.

Property Dialog. When the Property Dialog is opened, all the properties related to the current highlighted tree node in the Logical Tree Editor will be shown. Other than the

use of the assistant tools to derive the properties, the Property Dialog is the only place that users are allowed to modify them.

3.4.2 Assistant Tools

The Mapping Wizard provides a set of assistant tools to help automate the process of generating a mapping rule. These tools are the critical components that make the Mapping Wizard practically useful. We believe that, in practice, it is not possible to have a single tool to fully automate the whole mapping rule generation process because the modeling of users' perception of the Web site's logical structure requires human intervention from time to time and the mapping from logical nodes to physical Web pages is complex.

In the Mapping Wizard, rather than trying to provide a single tool, we build a set of tools that address different aspects of the problem. We identify the main difficulties and bottlenecks that tend to slow down the overall process and develop assistant tools to accelerate them. In this way, the efficiency of the overall process is improved and the time required to generate a mapping rule of a given Web site is greatly reduced.

The assistant tools provided by the current implementation of the Mapping Wizard include Pattern Searching, HTML Source View Synchronizer, Pattern Induction Tools, Section Extraction Tool, and Structure Copier.

Pattern Searching Tool

Pattern Searching is similar to the "Search" function that most text editors provide. It allows the searching of certain text pattern within the HTML source and the Web browser view. Searching directly on the Mini Web Browser may be more helpful and straightforward to the users.

Searching with respect to current cursor position is an important feature for identifying the enclosing patterns of target information. When the user locates a text string as the target information and chooses the text string before that to be the BeginPattern, he or she would like to know whether this pattern appears before that position so as to make sure that the *Repeat* attribute of that pattern is specified correctly. The Pattern Searching tool can help in this case to search backward for the same pattern.

This tool may not seem important with the existence of the Pattern Induction Tools, which will be described later. However, the patterns derived by the Pattern Induction Tools

Figure 3.6: Using the Source View Synchronizer

may not be optimal in some cases. Users may want to modify and optimize the Mapping properties manually. In this case, Pattern Searching tools will be useful to test the manually modified patterns.

HTML Source View Synchronizer

When users locate information of interest, they usually browse through Web pages using the Mini Web Browser. When the user has found the information on the Web page, he or she would like to see the corresponding HTML source codes of that portion. However, to locate the HTML source of certain part of a Web page is not straightforward, especially when the size of the Web page is large. The *HTML Source View Synchronizer* is implemented to accomplish this task of synchronizing the cursor position of the HTML source with the HTML View, and vice versa. The tool automatically highlights, in the HTML Source Viewer, the HTML source of the current selected portion on the Mini Web Browser, or the reverse if the current selected position is in the HTML Source. As shown in Figure 3.6, the user highlights the word "WORLD" and invokes the command. The corresponding HTML source is then highlighted by the Mapping Wizard, as indicated in Figure 3.7.

The synchronizer is almost always used when users try to locate the target information. The result of the synchronization has been proven to be quite accurate, with rare exceptions where complicated non-HTML techniques are used in Web pages.

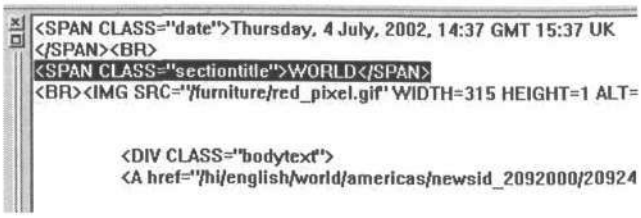

Figure 3.7: Synchronized HTML Source

Pattern Induction Tools

The *Pattern Induction Tools* are the most useful ones among all the assistant tools provided. They are built based on the two tools described previously and combined with pattern induction algorithms.

These tools are developed to release users from manually figuring out the patterns that enclose the target information. Without these tools, after having identified the information, the user has to use the HTML Source View Synchronizer to locate the corresponding HTML source codes to analyze the patterns appearing before and after the target text string, and to use the Pattern Searching tool to confirm that the starting and ending patterns are unique, and to finally record these patterns in the Mapping properties of the logical tree node. All these steps can be done automatically with one click using the Pattern Induction Tools.

There are two types of target information that these Pattern Induction Tools deal with. Elements like Section, Region and Record only need to identify the approximate area that contains the information of interest. They do not require their Mapping properties to enclose the exact boundary of the area, so long as all the interested information are enclosed. On the other hand, an Item element exactly locates a piece of information. It requires the Mapping properties to locate that information precisely. Therefore, the starting pattern and ending pattern must enclose the exact text string.

The current implementation of Mapping Wizard provides three Pattern Induction Tools: "Extract Region", "Extract Record", and "Extract Item". The first two are for the first type where only a rough area is required, while the third one is for the second type.

Rough Area Pattern Induction For the first two tools, Mapping Wizard will detect what portion on the Mini Web Browser is highlighted and invoke the HTML Source View Synchronizer internally to determine the corresponding HTML source. Once the HTML

source is found, the tools will apply an algorithm to derive the starting pattern and ending pattern that enclose this part of the HTML source. The algorithms used for inducing starting and ending patterns are symmetric. Here, we only discuss the algorithm for starting pattern.

The algorithm involves two repeating steps: *generating* a candidate and *testing* the generated candidate. It uses a trial and error approach to generate and test each candidate until a suitable candidate is found.

To generate candidates, the algorithm starts from the left of the target text string, and looks for the first complete HTML tag. For example, if the whole HTML source is

```
<TABLE><TR><TD>it's <FONT color=red>something interesting</FONT></TD></TR><TABLE>
```

and the target text string is

```
<FONT color=red>something interesting</FONT>
```

then the first candidate generated will be "<TD>". To generate the second candidate, the starting position of the first candidate is expanded to the left until another complete HTML tag is found, which produces "<TR><TD>" in this example. The same rule applies for the rest of the candidates.

Once a candidate is found, it will be tested with two criteria: the length of the candidate pattern and the frequency that it appears before. These two criteria are chosen to ensure that the derived pattern is stable and flexible to the possible changes in the Web page in future. The more frequent that this pattern appears in the HTML page, the more likely that the number of times that it appears might change. Once the number of times changes, the Mapping no longer locates the correct information. The same applies to the length of the candidate: the shorter the pattern, the more likely that it has a higher frequency of appearance. Therefore, to have a good induction result, suitable threshold values are required.

To test the first candidate, the algorithm calculates its length and uses the Pattern Searching tool to find out the number of times that this pattern "<TD>" appears, which is once. The testing is done by comparing these two number with the predefined thresholds. The values of the thresholds are determined empirically. In the implementation of the Mapping Wizard, we consider a candidate is suitable if it has a length longer than 5 and frequency less than 10. If a candidate is tested as suitable, it is taken as the derived value for

the pattern and the algorithm terminates. Otherwise, the algorithm continues to generate the next candidate and test it.

In this example, the final patterns derived are "<TR><TD>" for BeginPattern and "</TD></TR>" for EndPattern. The text located by these patterns is

```
it's <FONT color=red>something interesting</FONT>
```

This string is not exactly the same as the one that we want. But since it contains the target information and we only require a rough area, the pattern induction result is acceptable.

This algorithm is relatively simple as compared with those used in similar systems, such as STALKER [62]. The extraction language used in STALKER is more complex and its training algorithm requires more than one training example in order to induce accurate patterns. Since in the context of the Mapping Wizard, the number of training examples is always one, we prefer a simpler extraction language and simpler training algorithm so as to achieve a higher accuracy of pattern induction. Nevertheless, as we mentioned earlier, a more complex extraction language, such as STALKER, can always be plugged into the WICCAP Data Modal and the Mapping Wizard, so long as the implementation of the extraction module and the training module is available.

Exact Area Pattern Induction For the "Extract Item" tool, patterns that enclose exactly the target information are required. The first step for determining the corresponding HTML source is the same as the Rough Area Pattern Induction. A similar algorithm is also applied to derive the patterns.

The algorithm differs from the first one only in the selection of the first candidate. Since exact enclosing patterns are required, for starting pattern, any candidate must end exact at the last character next to the left of the target text string. Therefore, the first candidate is selected from the left of the target string to the first '<' character. The generation of second candidate and so on is the same as the first algorithm.

In the same example above, the final patterns derived are "<TD>it's" for BeginPattern and "</TD></TR>" for EndPattern. The text string located by these patterns now is exactly the same as the target text.

Exact Area Pattern Induction is particularly useful for deriving Mapping properties for *Item* elements. For example, to derive the patterns for an Item with type Link in the following text string

```
<A HREF="http://cais.ntu.edu.sg:8080/">
```

The final patterns induced are "`<A HREF="`" for BeginPattern and "`">`" for EndPattern. The text string located by these patterns is "`http://cais.ntu.edu.sg:8080/`", which is exactly the link that we want to extract.

As pointed out previously, these assistant tools can be used both on the Mini Web Browser and the HTML Source Viewer. If the user highlights on the Mini Web Browser, a synchronization between the HTML view and source is performed internally. Due to the highlighting mechanism implemented in the Microsoft Foundation Class (MFC), the user may not be able to select arbitrary portion of the HTML View on the Mini Web Browser. This is tolerable for Pattern Induction of rough area, which does not require an exact match and the user can always select a larger area. However, to achieve precise pattern induction for exact area, it is usually better to do the highlighting of the target information in the HTML Source Viewer.

Section Extraction Tool

When creating mapping rules, the first level of tree nodes is usually a list of links pointing to different pages. For example, in BBC Online News, the first level tree nodes are Sections such as "World", "UK" and "Business", each of which points to its own Web page. The links of these Sections exist in the homepage of the Web site and are located close to each other within a small area.

With the tools described above, users have to highlight and let Mapping Wizard derive the patterns for each of these Sections, although they are all very similar. Since there could be quite a number of Sections, specifying the Mapping properties of them one by one is quite slow. The *Section Extraction Tool* is developed to handle this special case by attempting to figure out all the possible Section nodes in the same level and derive the Mapping properties for them in a single click.

The user highlights the area that contains the list of Sections on the Mini Web Browser and the tool finds out the HTML source using the HTML Source View Synchronizer and applies suitable heuristics, based on common menu layout patterns, to identify the Sections. It then uses a similar induction algorithm to that of Exact Area Pattern Induction to derive the patterns that locate the links to the Sections' own Web pages.

Normally, home pages or index pages of Web sites use some kind of menu to display a list of Sections. These menus usually make use of HTML TABLE or other tags to layout the menu items nicely. Based on this fact, we develop heuristics to locate each Section, depending on the type of tags that the Web site uses to construct its menu. The current implementation of Section Extraction Tool uses a simple heuristic, which blindly searches for the tag `<A>` and treats it as a Section. The patterns for locating the link are induced in a similar manner as illustrated in the example in last Section. The Name attribute of the Section node is extracted from the text enclosed by the tag `<A>` and `</A>`. More complex heuristics that exploit the common menu layout structures will be explored in the future.

As an example, a Web site may have a menu like this

```
<TABLE>
    <TR><TD><A HREF="worldnews.htm">World</A></TD></TR>
    <TR><TD><A HREF="biznews.htm">Business</A></TD></TR>
</TABLE>
```

After applying the Section Extraction Tool, two Sections will be identified and inserted into the logical tree in the Logical Tree Editor automatically. These two Sections will have Name attribute of "World" and "Business" respectively. Their Mapping properties will be a Link containing a Locator that has a BeginPattern of "`<A HREF="`" and a EndPattern of "`">`".

Our experiments showed that the simple heuristic used in our implementation performs quite well in the actual extraction. The tool successfully extracted Sections for all the tested Web sites. Even when deriving subsections under a Section, the tool is still able to recognize the Section names and patterns for the links correctly.

Structure Copier

After all the Sections nodes have been automatically inserted, the user has to insert the Region and Record and other child nodes under each Section and derive their Mapping properties. This works fine for a Web site with two or three Sections. But for a Web site that has 10 Sections in the first level, the same process of specifying Region, Record and Item under the first Section has to be repeated for another 9 times.

Fortunately, in most Web sites, the structure of each Section is nearly identical, as they are normally generated by the same back-end engine with the same formatting styles. To

make use of this fact to further accelerate the process, the Mapping Wizard provides another tool, the *Structure Copier*. The basic idea is to copy the structure of the constructed Section to other empty Section nodes, hoping that the same sub-tree hierarchy will fit into those Sections with little modification.

It should be pointed out that this assistant tool is in fact a special case of "Copy and Paste". It can be seen as a combination of a single deep "Copy" and a batch "Paste". A deep "Copy and Paste" means recursively copying all the descendants of the source node into the target node.

Combining All Tools

Using all the assistant tools together to create a mapping rule, the user first uses Section Extraction Tool to generate all the Sections, uses the Pattern Induction Tools to identify sub-tree nodes under the first Section, then activates the Structure Copier to copy the structure of the first Section to other Sections. After some modification on the rest of the Sections, a mapping rule is generated.

3.5 Experiments

We have chosen four example sites (see Table 3.1) to conduct an evaluation of the Mapping Wizard. Most of these sites have been used for testing purposes by other similar systems. These four Web sites are chosen because they represent four very different categories of Web sites. Three users (with 1 to 2 years' experience of HTML and some knowledge of XML) were asked to build a mapping rule for each Web site. For BBC Online News, the news title, link to the full article and the short description of the headlines in each section and subsection were modelled. For Amazon, we modelled the link, title, author and price of top 5 best sellers in the first 10 categories. For CIA, a selected set of 5 attributes of all countries starting with an "A" were wrapped. For DBLP, the pages of VLDB conferences, the names of authors, paper title, pages, and link of each paper from 1991 to 1999 were modelled.

Table 3.2 summaries the key statistics of the experimental results. Users are required to record: (a) the number of nodes whose mapping properties are derived by the Pattern Induction Tool; (b) the number of sections that are extracted using the Section Extraction Tool; (c) the number of nodes that are created by using the Structure Copier; (d) the number

Table 3.1: Test-sites used for Mapping Wizard

Name	*Web site*	*Description*
BBC Online News	news.bbc.co.uk	News from BBC
Amazon	www.amazon.com	Amazon Books
CIA Fact Book	www.odci.gov/cia/publications/factbook/	Countries starting with A
DBLP	www.informatik.uni-trier.de/~ley/db/	VLDB Papers

Table 3.2: Evaluation of Mapping Wizard

Name	*Nodes Derived*	*Section Extracted*	*Nodes Copied*	*Manually Modified*	*Time (mins)*	*Time Modeling*	*Time Mapping*
BBC	4	8	28	4 (9.1%)	27	11 (41%)	8 (30%)
Amazon	5	10	54	8 (10.4%)	41	17 (41%)	19 (46%)
CIA	11	20	238	7 (2.5%)	51	14 (27%)	29 (57%)
DBLP	12	9	96	6 (4.9%)	44	12 (27%)	20 (45%)

of nodes that required manual modification; (e) total time to construct the mapping rule; (f) time spent in constructing the logical model and (g) time spent in specifying the mapping details. Note that there is some amount of the time that was spent on waiting for testing result due to network speed and this time is the difference between (e) and the sum of (f) and (g).

From the result, we noticed that, except Amazon, all the three other Web sites required less than 10% of *manual modification* to the nodes automatically generated by the assistant tools. The amount of manual effort required is an important indicator of the efficiency of the Mapping Wizard and the degree of automation that it can offer. This figure largely depends on the efficiency of the two higher level tools Section Extraction Tools and Structure Copier. As we mentioned earlier, these two tools manipulate a group of nodes instead of a single node at a time. The more nodes they can generate, the higher the efficiency of the overall process, thus the higher the degree of automation. This is especially true for the Structure Copier, which may result in a large number of nodes copied. In the case of CIA Fact Book, as the page for each country has the identical structure, simple structure copying worked very well and had saved the users a lot of effort, which resulted in the lowest manual modification. The 10.4% manual effort required for Amazon was partly due to the fact that the users were required to model only 10 categories of the books. If all (35) categories were modelled, this figure will be significantly reduced (estimated 3.2%).

The total time required to construct a mapping rule for each Web site is within one hour. One of the most recent systems, Lixto [7], requires about half of the amount of time.

However, the time required depends very much on the granularity of the information that the constructed wrapper can extract (i.e. the depth of the tree and the number of nodes in the tree). In addition, the WICCAP Data Model deals with the whole Web site while Lixto seems to focus on a single Web page in its experiment. To form a site-level view, similar amount of time has to be spent on each Web page that the users are interested in and some additional time is also required to combine all the separate page view into an integrated site view. The total time would obviously be much larger. Therefore, we consider the Mapping Wizard a much more efficient tool than other similar systems.

We also saw a relatively consistent amount of time (from 11 to 17 minutes) spent on figuring out and constructing the logical skeleton of the Web sites. The time for specifying the mapping details varied depending on the number of nodes in the actual mapping rules. The CIA Fact Book required the longest time because there is a huge amount of information for each country, which leads to a large number of nodes. Thus, the total time needed is dependent on the granularity of the mapping rules constructed.

All users were able to obtain 100% accuracy for modeling the Web sites using the Mapping Wizard. No users had to manually edit the generated mapping rule's XML document.

3.6 Related Work

Web information extraction systems are also known as *wrapper generation systems*, which was traditionally divided into three groups: *manual wrapper generation*, *wrapper induction*, and *supervised wrapper generation*. This classification scheme has gradually become too simplistic with recent appearance of several interesting Web information extraction systems. A more detailed classification is discussed in [45].

Systems such as Tsimmis [33], Araneus [58] and Jedi [36] support only *manual wrapper generation*. These systems provide expressive and comprehensive languages for information extraction. The extraction languages in these systems are so complex that they have to be written manually by some experts users. There is no algorithm that helps derive the languages from examples or GUI support for the ease of writing the extraction rules. Thus, manual wrapper generation systems tend to be difficult to use.

Another category of systems, including Wien [42], SoftMealy [35] and STALKER [62], take the *machine learning* approach to learn the extraction rules by examples. They are termed *wrapper induction systems*, because algorithms have been developed

to induce wrappers from a set of marked examples. These systems also provide visual support to allow easy selection and labelling of training examples. However, Wien is the pioneering work in this area and can not handle sources with nested structure and missing and varying-order attributes. SoftMealy requires seeing all possible special cases (missing attributes or out of order attributes) in the training phase in order to induce a correct wrapper. STALKER provides a more expressive extraction language by allowing disjunction and multiple patterns for locating information (WDM only supports a single starting or ending pattern). All these wrapper induction systems have been traditionally seen as automatic wrapper generation systems. But before the automatic training process, they all require a phase of marking of examples which might be non-trivial. When the overhead of marking training examples overtake the advantage given by automatic training, it may not be worthy to use such systems. In addition, all these three systems, and most of other similar systems, only deal with a single Web page or a set of Web pages with similar structure. They lack the ability to traverse along hyper-links; thus unable to extract information appearing in multiple inter-linked Web pages.

Our work is most closely related to *supervised wrapper generation systems*. Systems in this category include DEByE [44], Lixto [7], NoDoSE [1], XWrap [50] and W4F[1] [68]. A common feature of these systems is the GUI tools together with internal algorithms for extraction rules derivation that have been developed to support the generation of wrappers interactively. Although the wrapper generation process is not fully automatic, they attempt to reduce the required human intervention as much as possible.

DEByE allows users to specify a structure according to their perception of the information source. This is similar to WDM's logical view of Web site, although DEByE deals with information only at page-level. Unlike in the Mapping Wizard, this user-perceived structure is not specified explicitly but at the time when the users construct the examples using nested-table. The proposed data model and extraction language (called *oe-patterns*) only allow extraction patterns to exist in leaf nodes. This leads to the preference to a bottom-up extraction strategy. On the contrary, the WICCAP Data Model allows both internal nodes and leaf nodes to have their extraction patterns (the Mapping element). This makes a top-down extraction strategy more suitable and eliminates the need for an assembling phase for grouping instances of attributes together, which is required in DEByE.

[1]W4F was discussed in Chapter 2 as a language-based system. However, it is included here under the category of supervised wrapper generation systems due to its graphical support for extraction rule derivation.

Another very similar tool is the Lixto system. Lixto allows users to interactively build wrappers by creating *patterns* (tree nodes) in a hierarchical manner. The extraction rule associated with each pattern is a disjunction of one or more *filter*, each may consist of a conjunction of *conditions*. Such extraction language, called Elog, is as expressive as Monadic Datalog [30]. One advantage of Lixto is that although the internal Elog language is rather complex, the visual tools provided allow users to deal only with the GUI and the browser view of the HTML Web pages, while not touching the underlying Elog rules. This is possible because a filter can be represented by an Elog rule and Lixto provides tools of very fine granularity that support the specification of every single filter and condition. The nested structure of patterns implicitly forms certain logical structure as the users compose the pattern hierarchy. This is again different from WDM where the logical structure is explicitly specified. The Elog language includes a predicate *getDocument(S,X)* that allows crawling across Web documents. This potentially gives Lixto the ability to model and extract data from multiple Web pages, although the tools do not explicitly provide such support. The extracted results in Lixto can be manipulated by translating them into XML. Similarly, in the WICCAP system, the second layer that performs the extraction is also capable of performing (more involved) post-processing tasks such as filtering and consolidation [47].

Although DEByE and Lixto both have an expressive user interface for creating wrappers, the tools available are mostly for wrapper generation operations that are of the lowest granularity, i.e. for specifying extraction patterns for each individual node. Such kinds of tools are too low level when dealing with large scale Web sites. The wrapper generation process will be tedious and time-consuming, although the derivation of patterns for each individual node is efficient. On the contrary, the Mapping Wizard provides higher level tools such as Section Extraction Tool and Structure Copier that manipulate several nodes and even sub-trees. Although such high level assistant tools may not achieve 100% accuracy, they can greatly shorten the time and reduce the manual effort involved, as shown in our experiments.

NoDoSE adopts a similar approach as the Mapping Wizard to generate wrappers. It uses a similar top-down strategy to decompose a document into a hierarchical structure. However, in NoDoSE, the whole documents must be decomposed completely, while in the Mapping Wizard the logical structure is constructed independently of the Web pages and only representative portions of the pages need to be marked. Moreover, NoDoSE is unable

to follow links in HTML pages, hence unable to model a whole Web site. Nevertheless, NoDoSE is designed to handle a broader category of information sources, namely plain text. This is a feature that most Web information extraction systems lack.

A different approach based on HTML DOM tree is used in XWrap and W4F, where the users see the HTML DOM tree instead of the HTML source when building the wrappers. In XWrap, users indicate the portions to be extracted by browsing the DOM tree and selecting relevant nodes. A set of pre-defined heuristics can be applied after the users have selected some portion to derive the actual extraction rules. The ability for the users to specify their views of the logical structure is limited because the hierarchical structure extraction is performed based on predefined XML templates that contains instructions on deriving the structure.

W4F offers an expressive extraction language, called HEL, that can be used to perform extraction based on the HTML DOM tree. However, the users are required to write the extraction rules in HEL manually. Although W4F offers a set of GUI toolkits to help users accomplish this goal, there is no algorithm for automatically deriving the extraction rules. From this point of view, W4F is closer to the category of manual wrapper generation, rather than supervised wrapper generation.

One system that does not fall into the 3-group classification is the work by the BYU Data Extraction Research Group (DEG) [24]. It is based on an ontology-based approach that makes use of a semantic data model, called *application ontology*, which contains both a narrow ontology about the domain of interest and the keywords and constraints for the extraction of individual data items. Once the application ontology of a particular domain has been produced, the extraction of data from any Web page from the same domain is fully automatic, which is what all the systems mentioned above, including WICCAP, can not achieve. However, the construction of an application ontology is time-consuming and requires an expert who is familiar with both the syntax of the ontology language and the target domain. Although the structure exhibited by the application ontology reflects the logical organization of the data on the target page, it is rather narrow and usually has only one or two levels. In addition, it only deals with a single page. Therefore, the ability to model logical views of Web sites is considerably limited.

Two other systems that go further in terms of the degree of automation are RoadRunner [22, 21] and the ExAlg algorithm [3]. The whole wrapper derivation process in these

two systems are fully automated. They do not require any human intervention except for supplying target pages with similar structure (pages that are generated from the same template). This is a distinct feature comparing with all other wrapper generation systems. However, one disadvantage of such kind of automatic wrapper derivation systems is that they extract the syntactic structure, instead of the semantic one. The users have to perform some post-processing steps in order to get the real logical view that they want. RoadRunner and ExAlg are also limited to deriving wrappers at page-level.

Among all these related systems, only few (DEByE, Lixto and Wien) offer the similar functionality as the Mapping Wizard to allow users to directly operate on the browser view of Web pages instead of the HTML sources. Being able to operate directly on the browser view is critical to a supervised wrapper generation system because it does not require the user to have in-depth knowledge about HTML and JavaScript.

Most systems, including DEByE, Wien, XWrap, W4F, the BYU DEG work and RoadRunner, deal with only a single Web page or a set of Web pages with similar structure. This greatly limits the ability to modeling a Web site as a whole. Although it is possible to aggregate several page views, produced by these systems, to form a higher level site view, the efforts required to compose the site view out of several page views may not be trivial, especially when this has to be done manually without any support of visual tools. On the contrary, the WICCAP Data Model allows extraction of data from multiple inter-linked Web pages by explicitly incorporating the concept of a hyper-link. This gives users the flexibility in deciding the scale of the information sources that they want to model, i.e. the users can either choose model the whole Web site, a single Web page, or just a segment of a page.

The Araneus data model (ADM) used in the Araneus project is able to model an entire Web site. It adopts a variant of the ODMG model [15] and explicitly introduces the concept of a *page scheme* to model pages of similar structure. The attributes of a page scheme can be rather complex and may point to other page schemes. Although the ADM is able to model a wider class of Web sites (it uses a directed graph rather than a tree), the use of page in the model makes the resulting models close to the physical structure of Web sites. In the WICCAP Data Model, the concept of a Web page is eliminated and the data on a Web site is modelled solely based on their logical structure. The user may choose to represent a page with some WDM element; but most of the time, WDM elements will be used to represent just a fragment of a page. Unlike in ADM, where the user sees the hyperlinks

when they look at the data model, the WICCAP Data Model hides the mapping details in the Mapping related elements. When showing the logical tree with only the WDM elements for logical modeling purpose, the physical structure of the Web site is completely hidden from the user.

3.7 Summary

In this chapter, we introduce a new data model (WICCAP Data Model) for mapping Web sites from their physical structures to logical views. This data model is capable of describing the hidden logical structure of multiple inter-linked Web pages. We believe that the ability to model at site-level into the data model is important for building logical views of Web sites. Our proposed model also achieves a complete decoupling of the logical view from the physical structure of Web sites by defining a dedicated group of elements for mapping purpose.

The Mapping Wizard has been implemented as a software tool to automate the process of generating a data model. Since a fully automatic algorithm is not possible, several tools have been developed to address various bottlenecks in the generation process. Assistant tools such as Section Extraction Tool and Structure Copier offer higher degree of automation by operating on a group of nodes at the same time. By combining all the tools, the overall process of creating logical views of Web sites can be significantly accelerated.

The assistant tools introduced in Section 3.4 work quite well in speeding up the wrapper creation process. However, during the experiments, we also noticed the basic logical structure construction step is one main bottleneck that slowed down the entire process. The main reason is because only two assistant tools, section extraction tool and structure copier, are for this step. The section extraction tool only works on one level at a time while the structure copier is only useful when one whole subtree has been constructed. Therefore, to further improve the degree of automation, it is important to study how to speed up the process of building basic logical structure for a given site. This issue will be investigated in the next Chapter.

Chapter 4

Extracting Web Site Skeletons

4.1 Introduction

4.1.1 Motivation

Wrappers created by most existing wrapper systems [42, 3, 50, 33, 21] deal only with one document. However, information existing within a single page may not be complete by itself. Sometimes, a complete set of information can only be accessed by navigating several linked Web pages. More importantly, pages within a Web site often form some coherent structure that reflects the implicit logical relationship between information residing in different pages. In order to extract complete and logically structured information, wrappers must be able to extract data at site-level.

The process of site-level information extraction can be divided into two steps: identifying the Web pages containing the target information; and applying page-level wrappers to these pages to extract the information. In contrast to the second step, which has been quite well researched in recently years [3, 22, 33, 58], the first step remains relatively unexplored. It has been largely ignored by the Web information extraction (IE) community mainly because identifying relevant Web pages has been traditionally considered as the task of information retrieval (IR). In fact, this step is not merely an IR task of retrieving all the Web pages that contain the required information. To logically organize the information extracted by page-level wrappers, the implicit structure hidden behind the interlinking between the Web pages in a site has to be identified, too. Thus, it leads to the problem of identifying the hyperlink structure that connects the target pages within a site.

In the few wrapper generation systems [7, 57] that produce wrappers with the capabilities to traversing hyperlinks, including the Mapping Wizard presented in Chapter 3, the first

step is a relatively manual or semi-automatic (with the help of visual tools) process. This has, to a large extent, limited the degree of automation of the entire wrapper generation process. On the other hand, some recent automated wrapper generation systems [21, 3] are able to automatically create wrappers that extract information at page-level, which fully automates the second step. If we could have a complementary automated solution to the first step, we would be able to fully automate the site-level wrapper generation process.

To address these needs, more automated ways of discovering the skeleton of the target Web site are needed. The *skeleton* here refers to the hyperlink structure that is used in the Web site to organize the *core contents* that the site is providing. For example, a newspaper Web site provides news articles as its core contents and organizes news into different sections and subsections in different pages. The hierarchical structure formed by the interlinking between the homepage and the pages containing different sections and subsections is considered as the skeleton of the site.

To analyze the hyperlink structure of a Web site, existing IR and WWW searching approaches, such as HITS [40] and PageRank [12], use eigenvector-based techniques to discover important pages from a set of candidate pages. These discovered pages are usually separated. An additional assembling step is required to organize these pages into a meaningful structure. However, this is non-trivial since some of the intermediate pages that connect these important pages may not be discovered. Thus these methods often fail to find the skeletons of Web sites.

In this chapter, we attempt to bring Web IR and IE together to provide an integrated solution to the problem of site-level information extraction. In particular, we focus on the first step of the process and propose the SEW[1] algorithm that automatically discovers the skeleton of a given Web site. Instead of a bottom-up approach (finding out the content pages first and organize them later), the proposed algorithm starts from the homepage of a Web site and discovers the links to the pages in the next level in a top-down manner. It relies on a combination of several domain independent heuristics and features to identify the most important set of links within each page and organizes the pages into a hierarchy that captures the logical structure of the site.

[1] SEW stands for Skeleton Extraction from Web sites

4.1.2 Applications

The immediate application of such a Web site skeleton extraction algorithm is, of course, site-level information extraction. It would be a useful complement to both automated page-level wrapper generation systems [3, 21] and supervised wrapper generation systems [7, 57]. For the former, the pages appearing as leaf nodes (i.e. those pages containing the core contents) in the extracted skeleton can be supplied as input to produce page-level wrappers; thus making it possible to compose a site-level wrapper from a site skeleton and a set of page-level wrappers. For the latter, the extracted skeleton may serve as a starting point where users can refine the skeleton and specify extraction details of finer granularity in each individual page; hence greatly improving the degree of automation in wrapper generation.

Besides its direct applications in wrapper generation, the algorithm would also be useful to a number of Web mining applications. Web classification has mainly focused on individual Web pages or a small set of pages. Ester et al. [26] proposed to classify based on *Web site tree*, which is simply the entire hyperlink structure of the Web site. Given the skeleton of a Web site, classification can now be performed at a larger range of granularity (from page-level to site-level) and on a more focused (and probably more important) sub-structure of the entire hyperlink structure. The discovered skeleton may also be used together with Web server logs for *Web usage mining* and personalization. *Mining structures of Web sites*, which has not been well studied previously, may take advantage of the skeletons extracted to perform structure mining. Another example is *Web site comparison* in terms of content structure using their skeletons, which may help to discover hidden Web site replication based on content structures instead of presentation styles.

Applications of the algorithm are not limited to wrapper generation and Web mining. In fact, we believe that any task related to Web site structure may take advantage of it. For example, it is possible to periodically extract the skeleton of a Web site and detect the changes in structure. This is important to the task known as *wrapper maintenance*, where detection of structural changes is required (and yet still remains unexplored). The extracted skeleton may also help Web crawlers prioritize crawling by equipping them with knowledge of what pages are about the core contents in a particular Web site. The relationship among different pages within a Web site skeleton could also be used to determine the ranking or authority of the pages.

4.1.3 Outline of Chapter

The remainder of the chapter is organized as follows. Section 4.3 elaborates the SEW algorithm in details. The preliminary experimental results is presented in Section 4.4. Section 4.5 compares our work with related systems. Finally, Section 4.6 gives some concluding remarks.

4.2 Problem Definition

In this chapter, a *Web site*, or simply a *site*, refers to a set of interlinked Web pages that can be reached from some starting URL, such as "http://news.bbc.co.uk/" or "http://www.cnn.com/" The starting URL of a Web site is called the *base URL* of the site[2]. The page with the base URL is called the *homepage* of the Web site. The *core contents* of a Web site refer to the information that the majority of users visiting the Web site are interested in. For example, news articles in a online newspaper Web site are the core contents of the site whereas the advertisements or the "about us" information are not.

Pages containing core contents are called *content pages* and pages containing links to content pages are called *navigation pages*. A page that contains links to a set of navigation pages is also considered as a navigation page. A page can be both a content page and a navigation page if it contains core contents as well as links to other content pages. For example, the Web page about "World" news in CNN.com is considered as a content page because it provides news articles on events happening around the world. The same page also contains a set of links to the subsections under the "World" section, such as "Europe" and "Asia Pacific". Thus, it is also a navigation page. Among all the links in a navigation page, those that point to content pages or other navigation pages are called *navigation links*, which are collectively called the *navigation link set* of the page.

The *skeleton* of a Web site refers to the hyperlink structure that content pages in a Web site are organized into. We assume that there is only one skeleton for a Web site and the skeleton is of a tree-like structure[3], where leaf nodes are content pages and internal

[2]In general, it is possible for a site to have more than one URLs that can serve the purpose of a base URL. In such cases, we take the most commonly used one as the base URL.

[3]In practice, more than one navigation links in different pages may point to the same navigation page, making the structure of a Web site a directed graph instead of a tree. For simplicity, we take as the skeleton the spanning tree obtained by keeping, for each multi-parent node, only the parent that gives the node the smallest depth. However, the proposed algorithm, to be described later, is able to produce a general directed graph, with some simple post-processing.

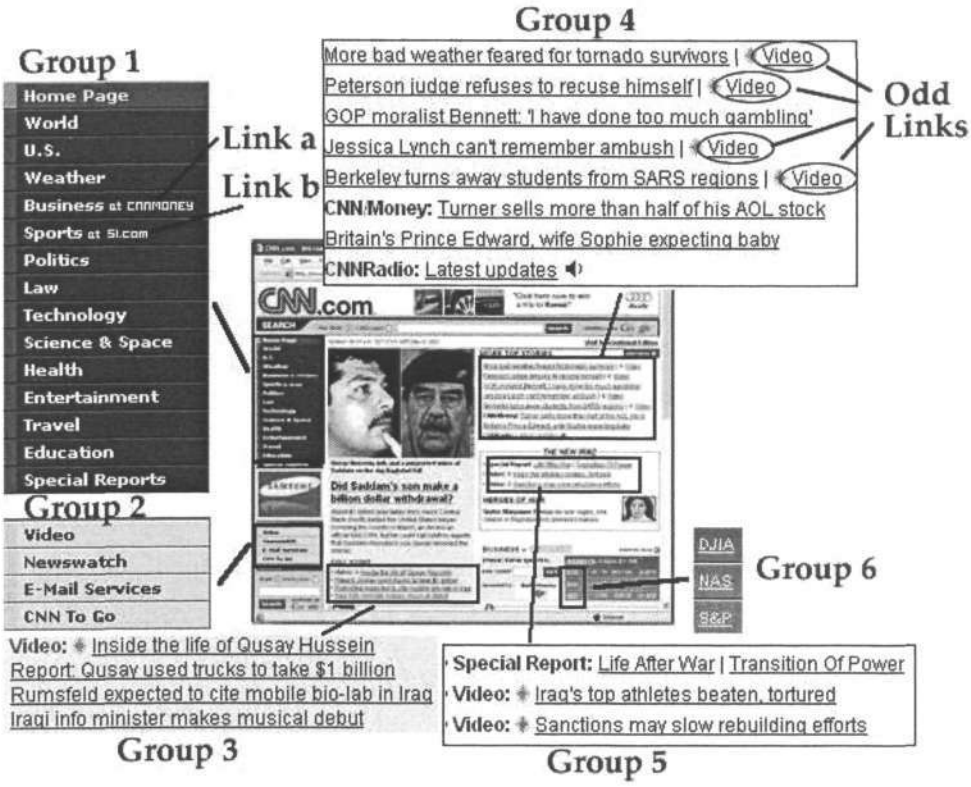

Figure 4.1: The Homepage of CNN.com

nodes are navigation pages containing links to their child nodes that could be either other navigation pages (internal nodes) or content pages (leaf nodes). Since navigation pages may also contain core contents, internal nodes could be content pages, too. With all the definitions above, the Web site skeleton extraction problem is simply defined as: *given a Web site, find its skeleton.*

To distinguish the wrappers that we are trying to build from others', we call wrappers that extract information from a Web site *site-level wrappers* and those that extract data from only one (content) page *page-level wrappers.*

4.3 The SEW Algorithm

In this section, we present the SEW algorithm for the Web site skeleton extraction problem. Section 4.3.1 gives an overview of the algorithm by walking through an example. Section 4.3.2 to 4.3.4 then elaborate the algorithm in details.

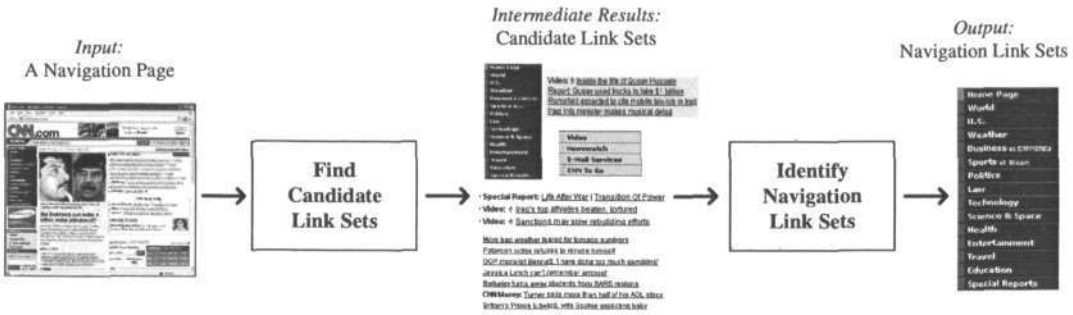

Figure 4.2: Discovering Navigation Links from a Navigation Page

4.3.1 Overview

Given the homepage of a Web site, the process of discovering navigation links from this page is divided into two steps: 1) finding *candidate link sets* and 2) identifying navigation links from the candidates, as illustrated in Figure 4.2. Candidate link sets are those sets of links in a page that could potentially be the navigation links. Consider the homepage of CNN.com[4] (shown in Figure 4.1), there are several clusters or groups of links where links within each group are structurally similar (in terms of DOM[5] tree) to each other[6], as highlighted in blocks in the figure. Each of these blocks may be considered as a candidate of the navigation link set of this page.

However, not all the link sets obtained by calculating structural similarity are good candidates. Several heuristics have been developed to prune the candidates based on observations of characteristics of navigation link sets. For example, group 6 in Figure 4.1 has only three links and thus is very unlikely to be a navigation link set, because a page usually contains more than three navigation links. In addition, within each group of links, there might be some links that are unlikely to be navigation links. For example, in Figure 4.1, link a and b in group 1 appear abnormally comparing with other links in the group. These two links should be removed to make the group a better candidate link set. The removal of these two links is also supported by the fact that they point to the same pages as two other links (Business and Sports) in the same group. After applying these heuristics, some link sets are discarded and some links within the remaining link sets are removed and a

[4]http://www.cnn.com

[5]Document Object Model. See http://www.w3.org/DOM/

[6]We consider a group of links as structurally similar if they have the same path to the root node in the DOM tree.

collection of candidate link sets is obtained. In the example in Figure 4.1, the candidate link sets are group 1, 2, 3, 4, and 5 (i.e. group 6 removed), with some odd links removed from group 1 and 4.

After obtaining these candidate link sets, the next step is to identify the link sets that are the most likely to be navigation links. This is done by looking at several features of each candidate and checking whether they have feature values similar to that of the navigation link sets that we have discovered. Examples of such features include average number of words in anchor texts and type of URL pointed by the links. For instance, in Figure 4.1, the average length and the number of words in the anchor texts of group 1 and 2 are quite close to the expected values of observed navigation link sets; while those of the rest of the groups are not. In addition, the URLs of the links in group 1 (not shown in the figure) are all relative URLs; whereas those of group 2 have some absolute URLs mixed together with relative ones. Based on navigation link sets found earlier, the URLs of their links tend to have the same type of URL, i.e., either all are absolute URLs or all relative. Thus we are confident that group 1 is more likely to be the navigation link set than group 2 based on this feature. To decide which candidate is the best, the confidence scores obtained from all the features are combined to give a final confidence. In the above example, both features prefer group 1 to others. Therefore, group 1 is selected as the navigation link set of this page. Note that we could have selected more than one candidate as navigation link sets if they all have high confidence scores.

The above process discovers the navigation links in the homepage of CNN.com. As each navigation link points to either a navigation page or a content page, all pages pointed by the discovered navigation links are retrieved and the same process of discovering navigation link set is applied to each of these pages. For pages other than the homepage, more pruning of candidates can be performed by removing those link sets similar to the navigation link sets that have been discovered in other pages. Pages without any candidate link set or those with candidates that give confidence scores lower than some threshold are considered as content pages. For pages where navigation links are found, the relevant pages are again retrieved and analyzed in a similar manner. The overall process terminates when no more new pages are to be retrieved for analysis. The skeleton of the Web site is then constructed by treating the homepage as the root node and the pages directly retrieved from the homepage as nodes in the second level and navigation links in the homepage as edges to these nodes.

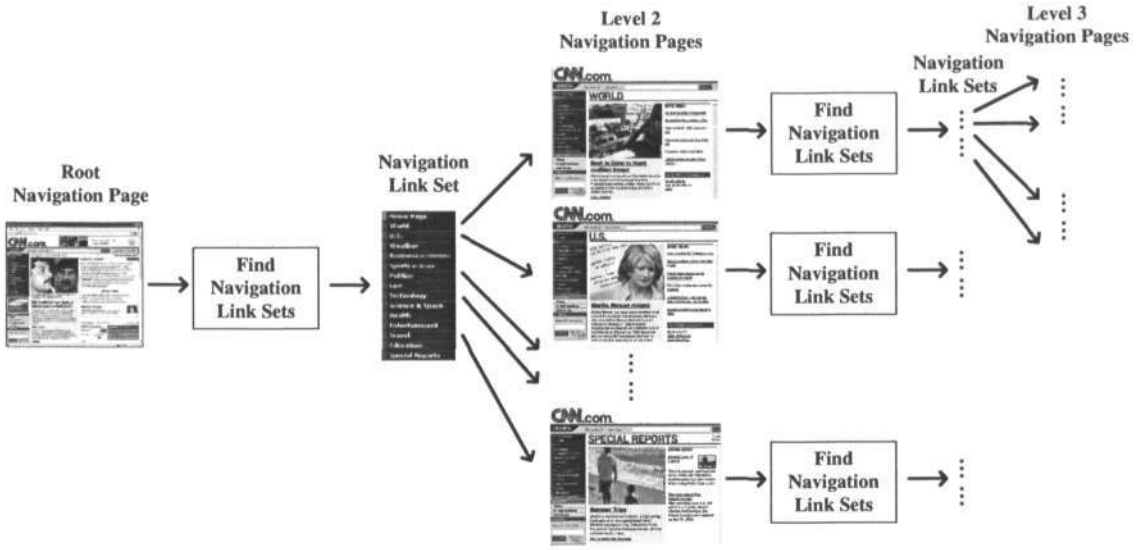

Figure 4.3: Discovering Skeleton from Navigation Pages

Subtree rooted from the nodes in the second level are constructed recursively. For the case of CNN.com, we obtain a tree with maximum depth of 3, where the second level contains nodes such as "World", "U.S.", "Weather", etc, and the node "World" has children "Asia", "Africa", "Europe", etc, as shown in Figure 4.3.

4.3.2 Finding Candidate Link Sets

As it is generally very difficult to directly identify the set of navigation links from all links in a page, we divide this process into two steps. First, we group the links into sets and select those that are more likely to be navigation link set for further investigation. This is the step for *finding candidate link sets*. Next, the *navigation link sets identification* step examines the candidate sets and chooses the best ones as the navigation link sets. This subsection describes the first step in details; the second step is discussed in next subsection (Section 4.3.3).

In the local context of a few pages (one page in the case of the homepage), a single link by itself provides little information. It is difficult to tell whether a link is a navigation link by looking at the link alone. Therefore, while other link-based analysis methods (e.g. HITS and PageRank) analyze individual links one by one (in a global context), we choose to analyze links in groups (in a local context). We consider a set of links as a candidate and explore the common characteristics exhibited by links within each set and their relationships

to further prune them. The same strategy of considering links in groups is also used in the next step to pick out the best candidate. It should be noted that, unlike typical link-based analysis methods, this approach makes very little use of information on the connectivity between pages. Nevertheless, such information is complementary to the proposed algorithm and could be incorporated later to achieve further improvement.

Two issues are involved in finding a good collection of candidate link sets out of all links in a page. Firstly, we have to decide how to cluster the links into groups. Secondly, since the number of groups and the number of links in each group might be large, we have to prune the groups. Before going into the details of the algorithm that addresses these two issues, we briefly describe the XHTML data model for representing Web documents.

The XHTML DOM Tree Model

Given an HTML Web page, we first convert it into XHTML[7] format (by cleaning bad and ill-formed tags and inserting end tags for bachelor tags[8]) and parse the XHTML page into a DOM tree by treating it as an XML document. Each start tag in the XHTML document (e.g. `<TABLE>` and `<A>`) is represented as a node. The label of the node, called NodeName, is the same as the name of the start tag (e.g. `TABLE` and `A`). Attributes of a start tag are converted to attributes of the corresponding node. Tags appearing between the occurrences of a start tag and the corresponding end tag are treated as child nodes of the node representing the start tag. Texts are converted to text nodes.

In this DOM tree model, links (those tags with tag name A) are represented as nodes with NodeName A, which we call *link nodes*. Link nodes are numbered in the order that they are visited in a pre-order, depth-first, left-to-right traversal of the DOM tree. It is the same as the order in which the A tags appear in the HTML source document. Among the attributes of a link node, the one named HREF, alternatively called the URL attribute, is of special interest because it is the URL of the page that the link is pointing to. Each link node is also associated with an additional attribute SecName that represents a short description of the page that the link is pointing to. The value of SecName is obtained by concatenating all the anchor texts of the link. If the anchor is an image, the ALT attribute of the IMG tag is taken as the value. If the value of the ALT attribute is not set, the value of SecName is an empty string.

[7] http://www.w3.org/TR/xhtml1/
[8] http://tidy.sourceforge.net/

```
1: function GENCANLINKSET(URL, CurNode)
2:   SRC = CLEANANDREPAIRHTML(URL)
3:   DOMTree = PARSEXHTML(SRC)
4:   CanLinkSets = {}
5:   AllLinks = FINDALLLINKNODES(DOMTree)
6:   while AllLinks is not empty do
7:     RefLink = first link node in AllLinks
8:     CanLinks = FINDLINKNODESATSAMELEVEL(RefLink, AllLinks)
9:     if CanLinks.size() > 0 then
10:      add RefLink into CanLinks
11:      PathSet = GROUPBYPATH(CanLinks)
12:      for each linkset LinkSet in PathSet do
13:        PRUNELINKSET(LinkSet, CurNode)
14:        if MinNL < LinkSet.size() < MaxNL then
15:          add LinkSet into CanLinkSet
16:        end if
17:        remove links in LinkSet from AllLinks
18:      end for
19:    else
20:      remove RefLink from AllLinks
21:    end if
22:  end while
23:  return CanLinkSets
24: end function
25:
26: function PRUNELINKSET(LinkSet, CurNode)
27:   PHANCHOR(LinkSet)
28:   PHLINKDUP(LinkSet)
29:   PHSTYLE(LinkSet)
30:   PHDISDIFF(LinkSet)
31:   PHMAXNUMWORD(LinkSet)
32:   PHLINKSETDUP(LinkSet,CurNode)
33: end function
```

Figure 4.4: The GENCANLINKSET Function

Generating Candidate Link Sets

Figure 4.4 depicts the algorithm for generating candidate link sets from a page. After obtaining the XHTML data model DOMTree, all the link nodes are identified and placed into a list, denoted as AllLinks, in the order of their numbering. An iterative process (the while loop in line 6–22) is then started to pull link nodes out of AllLinks and generate candidate link sets until AllLinks is empty.

Clustering Links At each iteration, the first link node in AllLinks, denoted as RefLink, is taken out and all link nodes remaining in AllLinks that have the same depth (i.e. number of nodes in the path to the root node in the XHTML DOM tree) as RefLink are found and added into the set CanLinks. An initial clustering of link nodes in CanLinks is performed by calling the GROUPBYPATH function, which simply divides the nodes into clusters where nodes in the same cluster have the same path to the root node in the XHTML DOM tree.

Clustering links by path to produce potential candidates is based on two observations. Firstly, link nodes corresponding to navigation links within a page are almost always located at the same level in the DOM tree. Secondly, since navigation links usually are presented in menu-like styles, they often have the same path in the DOM tree. It should be noted that physical proximity of links is not used, because sometimes navigation links may be regularly distant from each other. For example, a two-level menu may sometimes be used, where the links at the first level are separated by some links at the second level. In such a case, physical proximity may easily put links from two levels into one group; whereas the clustering by path approach can effectively distinguish the first level links from those at the second level, as they are very likely to have different paths or even appear at the different levels in the DOM tree.

Pruning Candidates As mentioned earlier, the initial candidate link sets produced may contain too many groups or groups too large in size. Therefore, at each iteration, after obtaining PathSet, which is a set of initial candidate link sets, we apply several simple heuristics to prune each initial candidate LinkSet (by calling the function PRUNELINKSET) and only consider the candidate if it survives after the pruning. The *pruning heuristics* used here all correspond to some observations of characteristics that navigation link sets have. Currently, the pruning heuristics that have been implemented include:

- **Anchor** The anchor of a link node refers to its child nodes. Most navigation links in a page are formatted similarly; thus having the same type of presentation style in their anchor contents. Based on this observation, the Anchor heuristic clusters the link nodes in LinkSet into groups by the type of their anchor contents and only keeps the group with the largest number of links. Anchor contents are considered as different types if their presentation styles are different. Since the most commonly used styles are plain texts and images (IMG tag), we only consider these two types here.

- **LinkDup** Link nodes in the same LinkSet with the same SecName or URL are considered redundant and only one will be kept. A pairwise comparison of the SecName and URL attributes of link nodes within the group is performed. If duplication of links is found, the one that appears later will be discarded.

- **Style** Navigation links usually not only have the same path but also the same Cascading Style Sheet (CSS) style and other presentation-related attributes in the nodes along the path to the root node. The Style heuristic further clusters the link nodes in LinkSet by comparing the CSS style of the nodes in the path. Other presentation related attributes that are used include SIZE and COLOR of the FONT tag, BGCOLOR of the TR and TD tags, etc. Only the largest cluster will be kept.

- **DisDiff** Although physical proximity is not appropriate for generating the initial candidates, it is still useful for pruning the links. The DisDiff heuristic computes the distances between every two consecutive link nodes (in their numbered order). We compute the distance between two nodes by counting the number of leaf nodes inbetween these nodes in the DOM tree. If some distance is found to be statistically much larger than the rest, it is considered as an indication that the LinkSet might be a combination of two or more candidate link sets. Therefore, if there is a statistically larger distance, the LinkSet is divided into two sets by the position of that distance in the link node sequence and only the first subset is retained.

- **MaxNumWord** We observe that navigation links usually contain few number of words in their anchor text (typically one or two words). Thus after all the above heuristics have been applied, the average number of words in the SecName attribute of all remaining link nodes in the group is calculated. If this number is greater than or

equal to a predefined threshold (4 in our experiments), the entire group of link nodes is discarded. This heuristic is applied after those introduced above because if applied earlier, the "bad" link nodes that are supposed to be removed by previous heuristics may introduce noise (longer SecName) that leads to the removal of a valid candidate link set.

- **LinkSetDup** If the page that is currently being analyzed is not the homepage, we have to ensure that the navigation link sets generated from previously analyzed pages are not reconsidered here. This heuristic simply checks whether LinkSet is the same as some navigation link set discovered earlier (from other pages). If yes, it will be discarded.

The heuristics discussed above, when applied to a potential candidate link set, either remove some "bad" candidate link from the set or discard the entire set. As an additional heuristic, we only consider candidate link set that has the size that is not too small and not too large (line 14 in Figure 4.4). LinkSet with the size in the range between MinNL and MaxNL is added into the candidate list CanLinkSets, where MinNL and MaxNL are predefined parameters denoting the minimum and maximum number of links for each candidate link set. All link nodes in LinkSet are removed from AllLinks. The same candidate pruning process is then repeated on other link sets resulted from the GROUPBYPATH function call in line 11 in Figure 4.4. When all initial candidate link sets have been pruned, another iteration of generating initial candidates and pruning them is started. The entire iteration process (line 6–22) terminates when AllLinks becomes empty. At this point, the function returns CanLinkSets, which contains all the generated candidate link sets.

4.3.3 Identifying Navigation Link Sets

With the candidate link sets generated from the previous step, the next task is to identify the navigation link sets from all the candidates. As the navigation links within a page might be logically divided into several groups, we do not restrict the number of navigation link sets to be exactly one. In order to find out the navigation link sets from all candidate sets, some sort of ranking mechanism is required such that all the candidates can be ordered in terms of the likelihood of each one being a navigation link set. The candidates with a likelihood score over certain threshold can then be selected.

We adopt a feature-based approach to evaluate the candidates based on a set of predefined features. We compute the value of each feature of each candidate link set and compare the value with the norm value of that feature. The closer the feature value to its norm, the more confident we are that the candidate is the right navigation link set. Finally, the confidence scores obtained from all features are combined to give the final confidence score of a candidate link set being the navigation link set. Using this combined confidence score, all candidates can be ranked.

Link Set Features

Similar to the pruning heuristics, the features are derived based on observations on the common characteristics of navigation links. Given a link set, the value of a feature is a numeric value. For each feature, we compute, based on observed navigation link sets, the average feature value and the standard deviation of the value. This average and standard deviation pair is used to calculate the likelihood, also called the *confidence score*, of a particular link set being a navigation link set, given the feature value that it has. The calculation of likelihood is based on the normal distribution probability density function, as shown in the function NORMDIST in Figure 4.5 (line 16–19). Note that this is a standardized normal distribution so that the shapes of the probability density functions of the features are normalized and the computed confidence scores for different features can be directly compared and combined. Based on the computed confidence score, the closer a feature value is to the average, the higher the score is; thus the more likely that a link set having that feature value is a navigation link set.

Currently, we have defined six features: NumWordName, LenName, VarLenName, URLType,
VarURLNumDir, and URLContain. The first three make use of the characteristics of the anchor texts of links. Given a link set, the value of the NumWordName feature is the average number of words in the SecName attribute of all link nodes in the set. Similarly, LenName calculates the average length (i.e. number of characters) of the SecName attribute. Rather than taking simple average, the VarLenName feature computes the variance of the length of SecName among all link nodes.

The last three features are based on the observation that the URL attributes (HREF) of navigation links tend to exhibit similar patterns. To measure the similarity of values of the

URL attributes among all link nodes, the URLType feature compares the types of the URLs. Five types of URLs have been defined: *relative* (the link is relative to the current directory), *absolute* (the link is relative to the current site), *cross-site* (the link is independent of the current site), *anchor* (links with '#'), and *others* (all other types of link, e.g. those involving JavaScript functions). We find out the type with the largest number of link nodes. We then count the number of link nodes of this type and normalize the count with the total number of nodes in the link set.

The VarURLNumDir feature is derived based on the fact that the number of levels of directories in the URL attributes of navigation link sets tend to be the same. To quantize this tendency towards the same value, we compute the variance of the number of levels of directories of all links in a link set. For relative and anchor URLs, the URL is prefixed with the sub-path from the directory of the homepage to the current directory. For absolute and cross-site URLs, the URL without the HTML filename is considered. For URLs whose type is *others*, the number of levels is taken to be a significantly large number (1000 in our experiment).

The last feature URLContain counts, for each candidate, the number of links whose URLs are contained under the directory of the current page. This corresponds to the observation that the hierarchy in a skeleton is often related to the directory hierarchy, where the navigation links contained within a navigation page often exist within the sub-directories of the directory of that page. For example, the Entertainment page of CNN.com is in the directory /SHOWBIZ/ and the navigation pages from this page such as Movies and Music all exist in the subdirectories of /SHOWBIZ/ (/SHOWBIZ/Movies/ and /SHOWBIZ/Music/). The count is normalized by the number of links in each candidate set.

All these features are based on either anchor texts or the URL attributes of link nodes. The DOM tree structure is not utilized, mainly because it has already been heavily used to prune link nodes in candidates. Since the link nodes in the candidates all go through the pruning step, they should have similar characteristics in terms of DOM tree structure. Therefore, the DOM structure is unlikely to give much information in this step.

Combined Ranking

The confidence score of each feature is calculated independently, as they each have their own pre-computed average and standard deviation. Using the confidence scores of each

feature, the candidate link sets can be ranked. However, the scores from different features might not agree on which candidate is the best, i.e. not all of them rank the same candidate link set as the top one. In fact, each feature may work well only for certain Web sites (or pages) but not others. Therefore, the ranking from different features have to be combined to provide a more accurate estimation of the navigation link set.

To combine the five heuristics, we define an event $e_{i,j}$ to represent the fact that the value of feature f_j of the ith candidate is consistent with the expected value of that feature. The confidence score $c_{i,j}$ of feature f_j of the ith candidate is the probability $P(e_{i,j})$ of event $e_{i,j}$ occurring. Thus, the problem of computing the combined confidence score c_i of the ith candidate is reduced to that of computing the combined probability $P(\wedge_j e_{i,j})$.

However, to compute $P(\wedge_j e_{i,j})$ exactly, a set of conditional probabilities is needed, which requires a large number of training data. For simplicity, we follow the common practice of assuming that the observations of events for each candidate are independent[9]. The combined probability is thus calculated by

$$P(\wedge_j e_{i,j}) = \prod_{j=1}^{n} P(e_{i,j})$$

Therefore, the combined confidence is given by

$$c_i = P(\wedge_j e_{i,j}) = \prod_{j=1}^{n} P(e_{i,j}) = \prod_{j=1}^{n} c_{i,j}$$

Figure 4.5 depicts the algorithm for selecting the best candidates as the navigation link sets. For each candidate link set, the algorithm calculates the confidence score of each feature (line 4–7) and combines the confidence scores based on the independence assumption by multiplying the confidence scores from individual features (line 23–25). It then adds the candidate into the set of final navigation link sets, if the candidate has a likelihood value above the predefined threshold τ (line 8–11). After iterating through all candidate link sets, the set $NavLinkSets$ is returned.

Given a candidate with a high score, we can't just simply accept it as the navigation link set because we do not know whether the page that we are dealing with is a navigation page or content page. To determine this, the algorithm tests the score of the candidate against the average μ_c and standard deviation σ_c of combined confidence scores pre-computed from

[9]Other dependency assumptions are also possible. See [43] for related discussion.

```
1: function SELECTNAVLINKSET(CanLinkSets)
2:   NavLinkSets = {}
3:   for each linkset s_i ∈ CanLinkSets do
4:     for each feature f_j ∈ F do
5:       c_{i,j} = NORMDIST(f_j(s_i), μ_{F_j}, σ_{F_j})
6:     end for
7:     c_i = COMBINECONF({c_{i,j} | 1 ≤ j ≤ |F|})
8:     p = PROB(c_i, μ_c, σ_c)
9:     if p ≥ τ then
10:      add s_i into NavLinkSets
11:    end if
12:  end for
13:  return NavLinkSets
14: end function
15:
16: function NORMDIST(f, μ, σ)
17:   f_st = (f − μ)/σ
18:   return 1/√(2π) e^{−f_st^2/2}
19: end function
20: function PROB(f, μ, σ)
21:   return ∫_{−∞}^{f} NORMDIST(f', μ, σ) df'
22: end function
23: function COMBINECONF(C)
24:   return ∏_{c_i ∈ C} c_i
25: end function
```

Figure 4.5: The SELECTNAVLINKSET Function

known navigation link sets. The test is performed by calculating the cumulative probability density p of the combined confidence score over a normal distribution curve determined by μ_c and σ_c (line 8–9, 22–24) and comparing p with a pre-defined threshold τ. If p is greater than τ, the candidate link set is accepted as the navigation link set. If no candidates have been accepted, the function returns an empty set, indicating that the page is a content page.

4.3.4 The Algorithm

The two functions GENCANLINKSETS and SELECTNAVLINKSET together provide a way to determine navigation links of a given page. With this, we are ready to introduce the complete SEW algorithm that extracts the skeleton of a Web site.

The complete algorithm is shown in Figure 4.6. To start the skeleton extraction process, we invoke the SEWALGORITHM function passing the base URL of the target Web

```
1: function SEWALGORITHM(BaseURL)
2:   return FINDSKELETON(BaseURL)
3: end function
4:
5: function FINDSKELETON(URL)
6:   return, if textitURL has been explored before
7:   CurNode = empty node
8:   CanLinkSets = GENCANLINKSETS(URL,CurNode)
9:   if CanLinkSets is not empty then
10:    NavLinkSets = SELECTNAVLINKSET(CanLinkSets)
11:    for each link set NavLinkSet in NavLinkSets do
12:      PseudoChildNode = a new pseudo node with empty url
13:      add PseudoChildNode as a child of CurNode
14:      for each link NavLink in NavLinkSet do
15:        ChildURL = HREF attribute of NavLink
16:        ChildNode = FINDSKELETON(ChildURL)
17:        add ChildNode as a child of PseudoChildNode
18:      end for
19:    end for
20:  end if
21:  return CurNode
22: end function
```

Figure 4.6: The SEW Algorithm

site as parameter. This function simply calls the FINDSKELETON function passing in the base URL. FINDSKELETON is a recursive function that, given a page, invokes GENERATE-CANLINKSETS and SELECTNAVLINKSET to discover the navigation links and, for each discovered navigation link, recursively calls itself with the link as parameter to discover more navigation links. One pseudo node is created for each navigation link set to provide logical grouping of navigation pages. However, if there is only one navigation link set, no pseudo node is created (not shown in Figure 4.6). The order in which nodes in the skeleton are generated is that of a pre-order traversal of the skeleton. At the end, SEWALGORITHM returns the root node of the skeleton.

The recursive call in FINDSKELETON returns when the function SELECTNAVLINKSET returns an empty set, i.e., the computed cumulative probability density p of all candidates is less than the threshold τ. Therefore, the testing of the values of p against τ can be considered as the stopping criterion of the entire algorithm.

The FINDSKELETON function also returns if the given URL has been explored earlier. In Section 4.2, it was mentioned that if a skeleton is a general directed graph, its spanning

Table 4.1: Dataset and Related Statistics

name	*base URL*	*total no. of nav pages*	*max depth*	*no. of nav pages at L1*	*no. of nav pages at L2*
Washington Post	www.washingtonpost.com	327	3	19 / 5.8%	162 / 49.5%
CNN News	www.cnn.com	119	3	15 / 12.6%	50 / 42.0%
New York Post	www.nypost.com	90	3	8 / 8.9%	36 / 40%
Washington Times	www.washtimes.com	50	2	18 / 36%	50 / 100%
BBC News	news.bbc.co.uk	24	3	13 / 54.2%	19 / 79.2%

tree will be used as the skeleton. In the SEW algorithm, if a node has multiple parents, i.e., a navigation page or content page is pointed to by multiple navigation pages, multiple (duplicate) nodes will be created, one for each incoming navigation link. However, the corresponding URL will only be processed once. Given the top-down left-to-right nature of the algorithm, the left-most node with shortest path will be expanded by FINDSKELETON. Nevertheless, with some simple post-processing, the resulted tree could be converted to the original directed graph. Therefore, the algorithm is, in fact, capable of handling directed graphs.

4.4 Experiments

The SEW algorithm has been implemented in C++ using Microsoft Visual C++. In this section, we describe the experiments that we have conducted on some real-life newspaper Web sites.

4.4.1 Dataset

In the experiments, the dataset consists of five Web sites with a total number of 610 Web pages from online newspaper domain. Table 4.1 lists the five Web sites together with the related statistics. All five Web sites provide online news that cover a diverse range of topics. The layout styles and locations of navigation links in the Web pages in these Web sites are rather versatile, ranging from nicely formatted vertical menu on the left hand side to links arranged in a grid-like table in the middle of a page. The Web sites were crawled and all experiments are performed on the crawled copy of the Web sites. The SecName and URL attributes of each navigation link are extracted and stored for computing the average and standard deviation of each feature.

The Web sites were manually inspected to figure out the skeleton. As shown in Table 4.1, the numbers of nodes in the skeleton (including navigation pages and content pages) vary

among sites depending on the complexity of the Web site skeleton and the coverage of the news contents. The percentage of nodes within a maximum depth of 1 and 2 are also calculated.

4.4.2 Overall Performance

This section describes the overall performance of the algorithm.

Methodology and Performance Metrics

The SEW algorithm requires a set of parameters, including the threshold for accepting a candidate and the average and standard deviation of all features and the combined confidence score. These parameters have to be computed during a training stage. To test the performance of the algorithm on the dataset, we performed a *leave-one-out* (LOO) cross validation by taking out one Web site for testing and using the other four as training data. The results were averaged and the standard deviation calculated. To test the maximum performance of the algorithm, we also performed a *test-on-training-data* (TOT) by testing each Web site using parameters trained from all five Web sites. This should in theory yield a better result than LOO since the training data will inevitably over-fit the testing data. For both LOO and TOT, the algorithm was run three times by limiting the maximum number of levels to be explored to 1, 2, and 3. These runs are denoted as LOO1, LOO2, LOO3, TOT1, TOT2, and TOT3 respectively. The threshold for accepting the best candidate link set was set to the lowest combined confidence score computed from the training data.

To measure the overall performance, for each Web site, the precision p_{oa} and recall r_{oa} of finding navigation links was computed as

$$p_{oa} = \frac{no.\ of\ correctly\ found\ nav\ page}{no.\ of\ pages\ found}$$

$$r_{oa} = \frac{no.\ of\ correctly\ found\ nav\ page}{no.\ of\ nav\ pages\ in\ the\ site}$$

and for each p and r pair, the F1-measure f is

$$f = \frac{2pr}{p+r}$$

Results and Discussion

Table 4.2 shows the overall performance of the algorithm by summarizing the results from all five Web sites. Each entry in the table is of the form "*average* ± *standard deviation*". For extraction of the first level of the skeletons (LOO1 and TOT1), the results were rather encouraging – all navigation links had been correctly identified and no false positives[10] produced. This indicates that the algorithm would perform well for sites with a small skeleton. This observation is further supported by the 100% recall ratios in Table 4.3 (showing the detail LOO results of each site) for New York Post, Washington Times and BBC News, which are the sites with the smallest skeletons among all sites. Being able to perform well in extracting small skeleton is a desirable property of a Web site skeleton extraction algorithm, because most of the Web sites are unlikely to have a structure as complex as that of Washington Post and CNN News.

It can also be seen that for the extraction of complete skeletons (LOO3 and TOT3), a recall of above 88% had been achieved. However, precision was relatively much lower, mainly due to the fact that for many leaf nodes in the skeleton, the algorithm failed to reject the candidates, which resulted in some candidates being (incorrectly) considered as navigation link sets and generating subtrees of false positives rooted from those candidates. This implies that the stopping criterion used in the algorithm does not perform very well.

The recall values with the number of levels limited to 2 (LOO2 and TOT2) were quite satisfactory, too. Precision was also improved (compared with LOO3 and TOT3), because the number of leaf nodes was less; thus giving less chances for false positives in leaf nodes to be accepted.

Another observation from Table 4.2 is that TOT did not perform significantly better than LOO; on the contrary, it even performed slightly worse than LOO in terms of recall. This shows that the feature set that we derive is site-independent since the sites did not really benefit from the parameters trained using the sites themselves and the parameters obtained from training were not really affected by the source of the training data, i.e., no overfitting.

We believe that, in practice, typically in a supervised wrapper generation environment, a high recall value is more important because it is much more difficult to find out the

[10]False positives refer to links that are wrongly identified as navigation links. Zero false positive is equivalent to 100% precision. False negatives refer to navigation links that are not discovered. Zero false negative is equivalent to 100% recall.

Table 4.2: Overall Performance

Method	p_{oa}	r_{oa}	f_{oa}
LOO1	1 ± 0	1 ± 0	1 ± 0
LOO2	0.530 ± 0.142	0.982 ± 0.0406	0.680 ± 0.122
LOO3	0.337 ± 0.210	0.883 ± 0.187	0.456 ± 0.215
TOT1	1 ± 0	1 ± 0	1 ± 0
TOT2	0.533 ± 0.179	0.982 ± 0.0406	0.678 ± 0.153
TOT3	0.341 ± 0.214	0.880 ± 0.188	0.461 ± 0.228

Table 4.3: Overall Performance by Site

	LOO1			LOO2			LOO3		
	p_{oa}	r_{oa}	f_{oa}	p_{oa}	r_{oa}	f_{oa}	p_{oa}	r_{oa}	f_{oa}
Washington Post	1	1	1	0.517	1	0.682	0.382	0.570	0.457
CNN News	1	1	1	0.454	0.909	0.606	0.225	0.845	0.356
New York Post	1	1	1	0.583	1	0.737	0.274	1	0.431
Washington Times	1	1	1	0.735	1	0.847	0.676	1	0.806
BBC News	1	1	1	0.358	1	0.528	0.130	1	0.230

correct navigation pages (in the case of low recall) than to remove the incorrectly identified navigation pages from a list (in the case of high recall but low precision). Meanwhile, since false positives often exist as a whole subtree, the removal of an incorrect internal node would eliminate a large number of incorrect nodes (those in the subtree). In addition, when used together with automated page-level wrapper generation systems, false positive could be easily detected because these systems will simply fail or return nothing indicating that the input pages are not the correct content pages. Therefore, we consider the results obtained in Tables 4.2 and 4.3 (high recall but moderate precision) quite satisfactory.

4.4.3 Performance of Individual Steps

To better understand the performance of the algorithm and to find out the reason why precision is not high, we also investigated the performance of the two steps in identifying navigation link sets of a page.

Performance Metrics

We computed two additional sets of precision and recall for the two sub-steps (*finding candidate link sets* and *identifying the navigation link sets*). To test the step of finding candidate link sets, for each navigation link set $nls_{i,j}$ in each correctly identified navigation

Table 4.4: Candidate Generation Performance

Method	p_{cg}	r_{cg}	f_{cg}
LOO1	1 ± 0	1 ± 0	1 ± 0
LOO2	1 ± 0	0.988 ± 0.074	0.992 ± 0.047
LOO3	0.981 ± 0.136	0.974 ± 0.148	0.976 ± 0.141

Table 4.5: Nav Link Set Selection Performance

Method	p_{ns}	r_{ns}	f_{ns}
LOO1	1 ± 0	1 ± 0	1 ± 0
LOO2	0.732 ± 0.225	0.978 ± 0.050	0.822 ± 0.166
LOO3	0.742 ± 0.230	0.945 ± 0.093	0.815 ± 0.170

page p_i, let $lsmax_{i,j}$ be the candidate link set that contains the largest number of correct navigation links in $nls_{i,j}$, we computed precision p_{cg} and recall r_{cg} of candidate link set selection by

$$p_{cg} = \frac{1}{\sum_{i=1}^{N} M_i} \sum_{i=1}^{N} \sum_{j=1}^{M_i} \frac{no.\ of\ correct\ nav\ links\ in\ lsmax_{i,j}}{no.\ of\ links\ in\ lsmax_{i,j}}$$

$$r_{cg} = \frac{1}{\sum_{i=1}^{N} M_i} \sum_{i=1}^{N} \sum_{j=1}^{M_i} \frac{no.\ of\ correct\ nav\ links\ in\ lsmax_{i,j}}{no.\ of\ links\ in\ nls_{i,j}}$$

where N is the number of correctly identified navigation pages that are internal nodes in the skeleton of the Web site and M_i is the number of navigation link sets in each page N_i of these N pages.

To test the step of identifying navigation link sets, for each Web site, we computed precision p_{ns} and r_{ns} of navigation link set selection by

$$p_{ns} = \frac{no.\ of\ correct\ nav\ link\ sets\ picked}{no.\ of\ nav\ link\ sets\ picked}$$

$$r_{ns} = \frac{no.\ of\ correct\ nav\ link\ sets\ picked}{no.\ of\ correct\ nav\ link\ sets\ in\ the\ candidate\ link\ sets}$$

Results and Discussion

Table 4.4 depicts the experimental results on the performance of the algorithm in terms of candidate generation. The results were quite encouraging, as we achieved precision and recall of above 97%. In our experiments, almost all candidates generated contained the correct navigation link set, with only one or two exceptions for each Web site. These high

precision and recall ratios are very important to the high recall achieved in the overall performance, because if the correct navigation link sets are not inside the candidates generated, the algorithm will have no way to pick them out from the candidates.

The implication of such a good performance in candidate generation is that the algorithm could be used in an interactive environment, in particular, the Mapping Wizard, to present the top-k ranked candidates (not just those with a confidence score strictly above the threshold) to allow the users to choose from. This would significantly improve the recall ratios, especially the recall of "Washington Post" in LOO3 of Table 4.3.

The precision of navigation link set identification is shown in Table 4.5. Similar to the first step, the recall ratios are still quite high. Indeed, the high recall in both steps is the main reason why the algorithm achieved a high recall in terms of overall performance. In contrast to recall, precision dropped to about 73% for LOO2 and LOO3. A low precision in this step means that the algorithm picks out some false positives (candidate link sets that are not navigation link sets). Once the false positives are considered as navigation pages by the algorithm, each of them will produce a whole subtree of false positives in subsequent recursive generation of navigation pages. This is kind of a snowball effect, where one false positive could actually lead to a big drop in precision ratio. It also explains why an averaged 74% of precision in Table 4.5 resulted in a precision as low as 13% in Table 4.3.

The drop in precision in the second step may be due to the inefficiency of the stopping criterion. Another reason may be the way the confidence scores of individual features are combined, which is currently based on the independent assumption. Nevertheless, we note that the algorithm worked extremely well when extracting only the first level of the skeleton.

4.4.4 Performance with Restricted Skeleton

In this subsection, we develop a slight variant of the SEW algorithm and compare its performance with the original algorithm. The purpose of this experiment is to verify the observations that we made in previous experiments (Section 4.4.2 and 4.4.2). In particular, we want to test whether the stopping criterion is the main cause of the low precision.

We call the modified algorithm "SEW with restricted skeleton". It is almost the same as the original SEW, except that at the second step, i.e., the navigation link set selection step, it picks out at most one navigation link set. This modification can be considered as giving the algorithm a tighter stopping criterion, since now even if there are more than one

Table 4.6: Overall Performance with Restricted Skeleton

Method	p_{oa}	r_{oa}	f_{oa}
LOO1	1 ± 0.0	1 ± 0.0	1 ± 0.0
LOO2	0.663 ± 0.169	0.908 ± 0.118	0.752 ± 0.132
LOO3	0.448 ± 0.212	0.879 ± 0.205	0.569 ± 0.206

Table 4.7: Candidate Generation Performance with Restricted Skeleton

Method	p_{cg}	r_{cg}	f_{cg}
LOO1	1 ± 0.0	1 ± 0.0	1 ± 0.0
LOO2	0.994 ± 0.027	0.981 ± 0.078	0.986 ± 0.054
LOO3	0.981 ± 0.123	0.974 ± 0.134	0.976 ± 0.127

candidate link sets with a confidence score above the threshold, the algorithm would pick only one of them (the one with the highest likelihood).

Table 4.6 depicts the results of overall performance of the SEW with restricted skeleton algorithm. Comparing the precision and recall ratios in this table with that of Table 4.2, we observe that precision of the modified algorithm was improved by a margin of 11 to 13 percentage points while recall dropped by about 1 to 7 percentage points (F-measure improved by 7 to 11 percentage points). The tightening of the stopping criterion resulted in the improvement on precision. Although the improvement is not very significant, it serves as an evidence to support our observations. For the second step (Table 4.8), the improvement on precision is much more significant - an about 20% jump in precision ratios. The results in Table 4.7 were rather close to those in Table 4.4, since this step is not directly affected by the stopping criterion.

With the results in Table 4.6, 4.7 and 4.8, we are quite confident to say that the stopping criterion is indeed the bottleneck.

4.4.5 Qualitative Analysis

The previous subsections discuss the experimental results in terms of quantitative measures. However, it would also be useful to give some examples of the extracted skeletons to demonstrate the quality of the extraction, especially the navigation link sets that are incorrectly extracted. In particular, as the quantitative results show unsatisfactory precision level, we would like to study some of the false negatives to understand why they are mistakenly identified as navigation link sets by the algorithm.

Table 4.8: Nav Link Set Selection Performance with Restricted Skeleton

Method	p_{ns}	r_{ns}	f_{ns}
LOO1	1 ± 0.0	1 ± 0.0	1 ± 0.0
LOO2	0.941 ± 0.096	0.941 ± 0.096	0.941 ± 0.096
LOO3	0.930 ± 0.102	0.924 ± 0.106	0.927 ± 0.104

Figure 4.7: An Example of False Negative in CNN.com

Figure 4.7 shows a navigation page "Special Reports" in the second level of the CNN.com skeleton. In the manually identified skeleton (the expected skeleton), this navigation page has no children, i.e., no navigation link sets in this page. However, as shown in Figure 4.7, a candidate link set containing a list of all archives by years is considered to be a navigation link set by the algorithm. Since categorizing archives by year is a usual practice, it seems also reasonable to consider this set of links as child nodes of the current page in the skeleton. This is, of course, a little bit subjective, since different people may have different opinion in what constitute the core content of a Web site. Nevertheless, this example does demonstrate the SEW algorithm's ability to identifying potential navigation link sets.

Another example of false negative appears in the same "Special Reports" page. As shown in Figure 4.8, two more candidate link sets are incorrectly identified as navigation link sets. By carefully examining these two link sets together with other set of links nearby, we note that it might be reasonable to include each of these link sets (U.S., BUSINESS, SPORTS, etc) as pseudo child nodes of the page, since these seem to be another type of categories, in addition to the archives by year discussed in previous paragraph. The

problem with the algorithm in this scenario is that it only extracts two out of the nine sets of links. Thus, regardless of whether to include these link sets as navigation link sets into the skeleton, the algorithm is not performing well. It should also be pointed out that the candidate link set in Figure 4.7 receives the highest combined score and is the one retained in the restricted skeleton version of the algorithm.

To study the reason why 100% recall could not be achieved, we examine a page where a false positive occurs. Figure 4.9 shows the "World" page of CNN.com. The six links highlighted in circles form a navigation link set in the skeleton. These six links as a group seem to be regularly arranged with similar look and feel. However, we notice that there are two additional segments under top two links "EUROPE" and "ASIA PACIFIC" with links pointing to the corresponding international editions. These two segments introduce some irregularity into the overall look and feel of the size links as a group. This could be the main reason why this candidate link set is not ranked the highest.

In the same "World" page in Figure 4.9, the candidate link set ranked with highest score is the set of links appearing at the bottom. This link set is a common footer that appears in many pages in CNN.com. They look perfectly like a navigation link set, since all links in this group share consistent look and feel. This kind of false negative might be eliminated using entropy based algorithm, such as LAMIS [38].

In summary, although the precision is not very satisfactory from quantitative point of view, a closer look into the quality of extracted skeleton suggests that some of the quantitative measures could be subjective. Nevertheless, this qualitative analysis also identifies some potential room for improvement of the algorithm.

4.5 Related Work

The development of the SEW algorithm has been mainly motivated by works in Wrapper Generation [3, 7, 21, 24, 50, 57], where toolkits have been built to automatically or semi-automatically generate software programs that are capable of automatically extracting data from Web pages. As mentioned in Section 4.1, most existing wrapper generation systems deal with information contained within a single page, although some of them [3, 21] can produce wrappers in a fully automated manner. Other systems [6, 7] allow construction of models that represent information at site-level. But the degree of automation in these systems is relatively low (manual or semi-automatic). In comparison with these systems, the

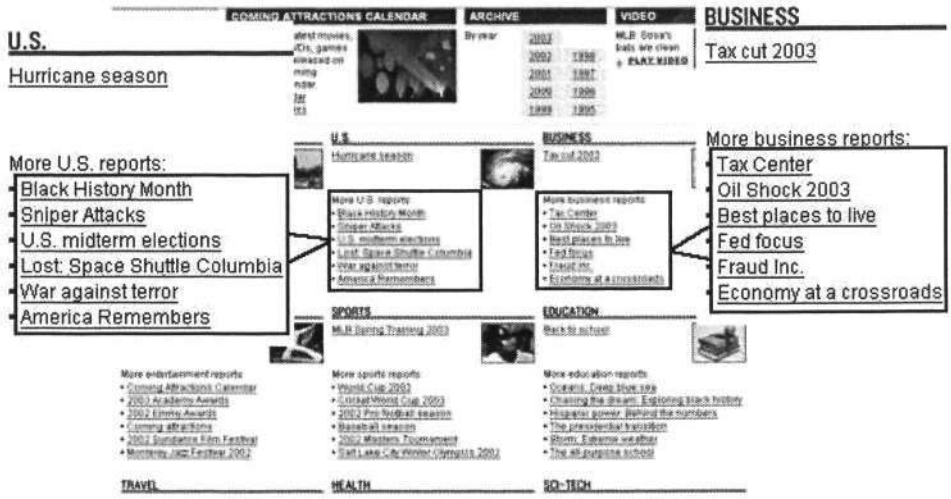

Figure 4.8: Another Example of False Negative in CNN.com

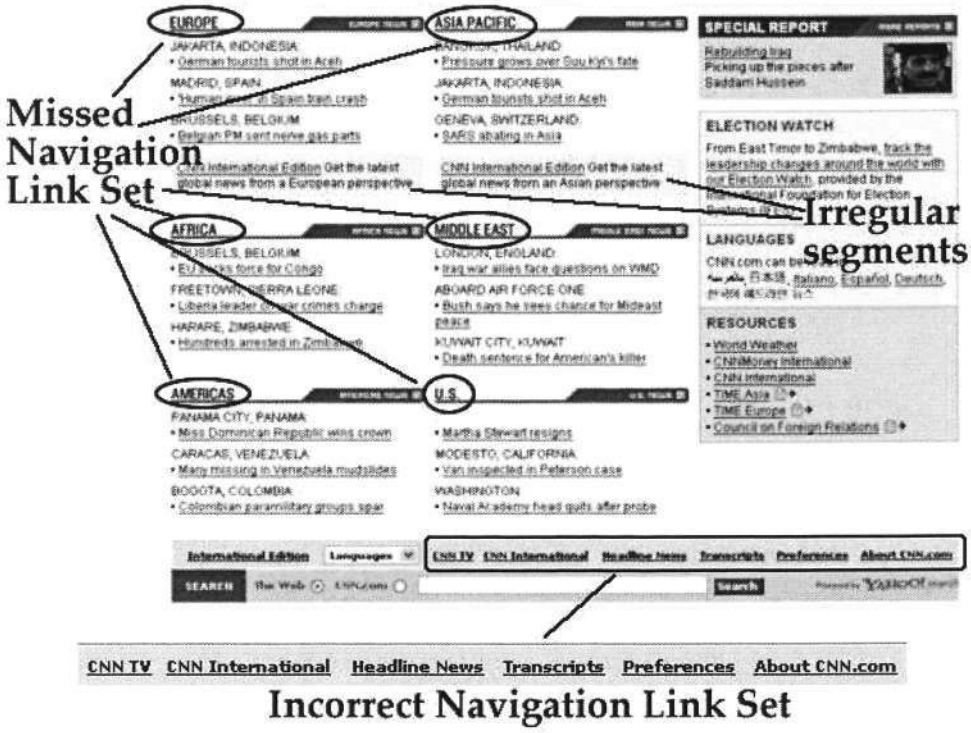

Figure 4.9: An Example of False Positive and False Negative in CNN.com

SEW algorithm distinguishes itself by providing an automated way to discover the hidden structure (skeleton) of a Web site. Automated algorithms such as those in [3, 21] are nice complements to the SEW algorithm since they work at page-level while SEW works at site-level by treating pages as basic units. The combination of both would provide a completely automated solution to wrapper generation with a broader coverage of granularity, from data distributed across an entire Web site to those inside a single page.

Two other systems addressed similar problems to that of the Web site skeleton extraction. LAMIS [38] is an entropy-based link analysis algorithm that tries to discover *informative structure* of news Web sites. The informative structure that it discovers consists of a set of TOC (table of contents) pages and a set of article pages. These two types of pages, although limited to news domain, correspond roughly to navigation pages and content pages defined in Section 4.2. LAMIS computes the entropy of each link and use it as weight in the HITS algorithm [40] to analyze all pages to find out the most probable TOC pages. Comparing with the top-down approach of finding navigation pages in SEW, this HITS-based algorithm tends to generate more false positives and false negatives since it looks at all pages in a site while SEW only deals with links in the pages that it analyses. In addition, it is not clear how all the discovered TOC pages can be organized to form an informative structure of a Web site.

Li et al. [48] proposed a system for constructing multi-granular and topic-focused Web site maps. The approach is based on discovery of entry pages to *logical domains* (a set of self-contained pages) within a physical domain (a Web site). Multi-granular and topic-focused site maps can be constructed by adjusting the weights that determine the importance of an entry page and displaying only the more important pages in the map. The organization of entry pages into site map structure is based on the physical directory structure, which is only one of the six features that were used in our algorithm (the URLContain feature). One disadvantage is that some intermediate navigation pages may be discarded if they are not ranked important; thus resulting in a large number of disconnected subtree of pages that could only be blindly clustered and organized under a single root node. Nevertheless, the proposed weight assignment techniques may also be used in SEW to produce a customized skeleton tailored to certain topic or certain required granularity of details.

WWW Search algorithms, such as PageRank [12] and HITS [40], rely on the link structure of Web pages to assign importance scores to the pages. These scores can be used

to answer user queries by selecting the most relevant pages based on both the query keywords and importance scores. The importance scores in these systems measure mostly the popularity or authority of pages, which roughly corresponds to the likelihood that a page is a (good) content page. Thus only some content pages can be accurately found and no navigation pages are found. Although HITS also introduces the concept of a *hub* that captures part of the meaning of a navigation page, the importance score of a hub is closely related to how many *authorities* that it links to. As a result, navigation pages pointing to content pages that are unpopular will not be discovered. One important characteristic that distinguishes the SEW algorithm from these algorithms is that it analyzes links in groups, which gives it much more information and allows it to extract more interesting features of the links. This larger amount of information enables SEW to perform well in a local context (e.g. a small Web site) as opposed to a global context (e.g. searching the entire Web), where link analysis based algorithms tend to perform better. In addition, the pages returned by these link-based algorithms are unorganized. Extra efforts are still required to produce a meaningful structure out of the list of search results, which sometimes may not be possible.

4.6 Summary

In this chapter, we study the Web site skeleton extraction problem, which is an important step in wrapper generation for Web information extraction. We present the SEW algorithm for automatically discovering the skeleton of a Web site. The algorithm works in a recursive manner by applying a two-step process to discover the navigation links in a page and retrieving pages from the links to discover more navigation links. The two-step process involves generating candidate link sets and selecting the best candidates as the navigation link sets. All the navigation links discovered and the pages retrieved form the skeleton of the Web site. Our experiments on real life newspaper Web sites showed that the algorithm performs well in recalling most of the navigation pages.

With the SEW algorithm, we are now able to quickly build up the logical structure of a Web site, which further contribute to the improvement in the degree of automation in the overall wrapper generation process. In next Chapter, we turn our attention to page-level information extraction and study how to extract information from a specific type of Web pages that users would also likely to encounter during the process of creating a wrapper.

Chapter 5

Extracting Streams of Structured Records

5.1 Introduction

5.1.1 Motivation

As the Web has gradually become one of the major media for publishing information, many organizations and individuals are making use of this media to disseminate information (internally or publicly) in a timely manner. With the introduction of tools such as content management software, Web blog, and RSS[1], information that might require hours or days to disseminate traditionally could now be made available on the Web in a matter of seconds or minutes. Given such convenience, information disseminators tend to update information more frequently, but with fewer changes in each update. This results in a significant portion of information on the Web that changes continuously but only slightly each time.

The dynamic characteristics of this type of information make it difficult for users to keep track of the changes. Since the changes occur quite frequently, users' attention is required at a high frequency correspondingly. Furthermore, most Web pages contain not only the information that users are interested in but also other content, such as advertisements, that might be changed from time to time as well. Thus, users have to spend extra efforts to check whether the occurred changes are those that they want to see. Existing Web browsers are not very helpful in this aspect as they lack the ability to tracking changes and to distinguishing changes in advertisements and in core content. Heavy user involvements are required to manually perform these tasks.

[1]RSS is a family of web feed formats used to publish frequently updated pages, such as blogs or news feeds. More details are available at http://en.wikipedia.org/wiki/RSS_(file_format)

As the amount of such dynamic information is increasing to a nearly unmanageable degree, the traditional information overloading problem of Web content adds to the complexity, making even the manual way of browsing Web pages to consume this information become very inefficient. Keyword searching, although used to be a quick way of retrieving information, could only return results on certain snapshot but not fail to capture the continuous changes of Web pages. The adoption of RSS helps ease this problem. However, there remain a large number of Web sites that do not provide RSS support.

To reduce information overload and to capture the dynamic aspects of this type of information, it is desirable to have tools that could detect when the information is updated, extract the newly updated portion, and determine the structure of the new information. Knowing the structure of the extracted changes is important because it allows us to perform further processing on the detected changes (e.g., filtering, transformation or querying). Such tools should be able to identify changes of the core content and ignore other non-relevant changes. Given the scale of the Web and the requirement to reduce human involvements, for the tools to be practically useful, they should also be as automated as possible.

In recent years, Web information extraction has been one of the active research areas aiming to solve the problem of information overload. Several approaches (manual, semi-automatic, and automatic) [3, 7, 21, 24, 42, 50, 57] have been proposed to selectively extract information from Web pages and transform the extracted information into structured format. The dynamic changing characteristics of the information mentioned above are not considered. Hence these approaches are not directly applicable to the problem.

Research in change detection and Web monitoring [11, 17, 18, 27, 49, 63] has focused on locating changes between different versions of same Web pages and detecting when the changes occur. However, these works do not provide the functionality to discover the schema of changes and to convert the detected changes into structured data for post-processing. In addition, all changes are treated uniformly, hence changes in core content could not be distinguished from other changes.

In this chapter, we propose a new framework that is capable of generating streams of structured records from continuously updated Web pages. This framework addresses all of the issues mentioned above. It detects changes, extracts only the changes in core content, and converts them into structured records. More importantly, it is automated and only

requires the URL of the target Web page as input parameter; thus is able to cope better with the large number of Web pages.

The proposed framework combines the approaches in wrapper generation and change detection. It works by repeatedly sampling the target Web page to retrieve instances of the Web page at different time points, computing the differences among these instances and using these differences to train a wrapper for extracting the data in structured format. Non-core content is identified by a different comparison algorithm on page instances obtained using different sampling frequencies and another wrapper is trained to filter it out from the training instances of the wrapper for core content. The trained wrappers are combined with other techniques to form the final record stream extractor.

5.1.2 Applications

The immediate application of the proposed framework is to generate RSS files for the Web sites that have not yet provided such facility. Since the extracted data is a stream of structured record, it is fairly straightforward to construct RSS XML files, after assigning meaningful tags to elements in the record schema. A more specific application could be monitoring competitors' Web sites for certain line of new products or special types of events or news by applying some filtering rules [56] to the extracted data. The extracted record streams from large Web site, such as news or stock sites, are also good candidates as input data to data mining and analysis software. Web crawlers could also benefit by prioritizing their crawling towards links extracted from new content and maybe retiring pages pointed to by links in removed core content.

5.1.3 Outline of Chapter

The following sections describe the framework in details. Section 5.2 defines the problem to be studied. Section 5.3 outlines the framework and Section 5.4 and 5.5 describe two of the main components in details. The performance of the implemented framework is studied in Section 5.6. After reviewing related works in Section 5.7, Section 5.8 concludes the chapter.

5.2 Problem Definition

In this chapter, we focus on a specific type of Web pages – those that are updated frequently but each update only consists of a small number of additions and/or deletions. Here, a Web

page or simply a *page* refers to a location on the Web pointed to by a URL. At any point in time, by sending a HTTP request with the URL to the Web server determined by that URL, we can obtain a snapshot of the page's content, which we call a *page instance*. Since the page is changing, page instances retrieved at different time points could be different.

The content inside each page instance is divided into two groups: *core content* and *non-core content*. Core content refers to the content that most users visiting this page are looking for; whereas non-core content refers to all other content presented on the same page.

In this chapter, we assume that core content in a page consists of a list of *records*. A record is the unit of each update, i.e., each update may add and delete zero, one, or more records in the list. Each record has a set of *attributes* that could be organized as a flat list or as a more complex, nested structure. We assume that all records in a page adhere to the same hidden schema[2]. Since HTML does not have a type system, we consider all attributes as strings.

Core content could be updated in various manner, depending on the maintainer of the page and the nature of the content. Common update patterns include: *insertion only* (new records are appended at one end of the list and no old records are deleted), *hybrid* (new records are appended at one end of the list and old records are deleted at the other end), *insertion with clearance* (same as insertion only except that all records in the list could be totally cleared from time to time). The third one could be handled similarly as the first one, with an extra step to check for "clear-list" events. Therefore, we focus on the first two types of update patterns in this chapter. *Deletion only* is not considered since it is not very common to present a whole list first and then delete the records without adding new ones.

It should be noted that for each update pattern, the list could be grown at either end. We assume that for a page, new records are added at one end of the list consistently and old records are deleted at the other end in the order that they are added. We say that the list grows *upwards* if new records are added at the head of the list (near to the beginning of the page) and *downwards* otherwise.

All other information presented in a page that is not core content is considered non-core content. Examples of non-core content include advertisements, site menu and other navigation aids. Most non-core content, such as menus and navigation aides, is relatively

[2]Since almost all large Web sites are generated by server-side programs, it is very likely that the records in a same page share the same logical structure and physical presentation templates (HTML tags).

static across different page instances, as compared with core content. We consider this type of non-core content as static content. On the other hand, some non-core content could change in almost every page instance, even though the core content might remain the same. We call this type of non-core content *record independent change* (RIC). In this chapter, we assume that RICs change more frequently than core content does. Advertisements are one typical example of record independent changes[3]. We also assume that the relative positions of all RICs in the HTML source of all page instances are the same, i.e., if RIC1 appears before RIC2 in one page instance, it also appears before RIC2 in all other page instances.

Given a page, we call a program a *record stream wrapper* if it could extract new core content as a stream of records when the page changes, with attributes in each record structured according to the hidden record schema of that page. The two key abilities of a record stream wrapper is to extract new core content but ignoring changes of non-core content and to organize attributes according to the record schema. Note that scheduling of page instances fetching, although part of a record stream wrapper, is not the main focus of this chapter. A simple periodic fetching technique is assumed to be used for retrieving page instances when the page stream wrapper is executed. Since given a record stream wrapper, it is rather straightforward to execute the wrapper to extract a stream of records, we consider the main technical difficulty of the problem lies in producing a record stream wrapper. Therefore, based the above definitions, the problem of extracting structured record stream could be simply stated as: *given a page, generate a record stream wrapper.*

5.3 System Overview

Our solution to the problem is to adopt a divide-and-conquer approach. We propose a general framework that decomposes the problem into five steps, i.e., five sub-problems, where each step could be studied individually. Figure 5.1 illustrates the decomposition. The following paragraphs describe each step in the framework in details.

Step 1. The first step is **to find record independent changes**. Since the final goal is to extract the changes in core content, we should, in the first place, be able to filter other irrelevant changes, i.e., changes in non-core content. Since record independent changes are

[3]For machine generated Web pages, advertisements are usually picked randomly from a predefined/changing set.

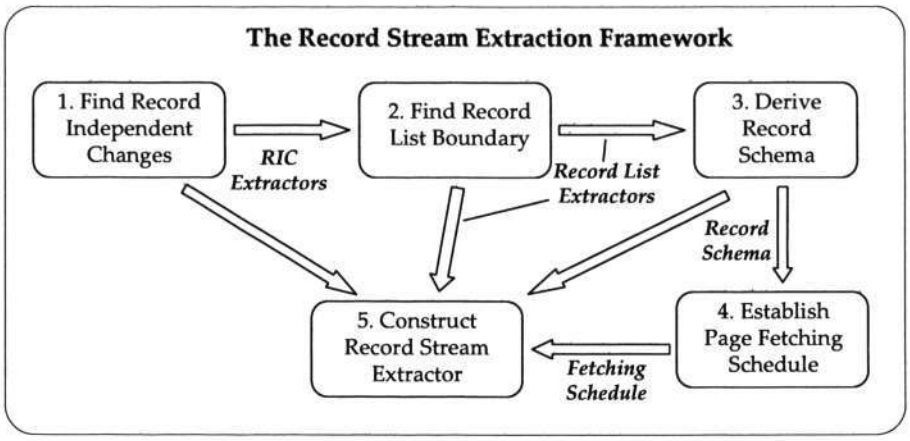

Figure 5.1: Overview of the Framework.

the only source of such irrelevant changes, if they are removed from each page instance, we would be sure that any changes occurring in the filtered page instances could only be caused by core content.

To find record independent changes, we retrieve several page instances with the same core content but different non-core content. By comparing these page instances, we can obtain some occurrences of each RIC in the page. All occurrences of an RIC could be treated as the training examples to feed to a wrapper generation program to produce a corresponding *RIC extractor*. Once all RICs have their RIC extractors generated, we can use them to extract and remove record independent changes in any (new) page instance.

Step 2. The second step involves **finding record list boundary and its growth direction**. This step locates the region within the page where the list of records could occur. It also finds out at which end of the list that new records are inserted. The record list boundary and the list growth direction are useful in both deriving the record schema and later in the extraction of records given new page instances. Ideally, the boundary should be as tight to the record list as possible.

To identify the boundary, we retrieve several page instances with slightly different core content such that the record list boundary in each page instance is different. By comparing these page instances, insertions and deletions of records can be identified. Based on the clues in the movements of the insertions and/or deletions, we can determine the growth direction of the list and the record list boundary in each page instance. The boundary in each page instance could be considered as one instance of the list boundary in the page;

hence, the list boundary could be derived by feeding list boundaries in all page instances to a wrapper generation program to produce a *record list extractor*. The resulting record list extractor is expected to be general enough to extract the list boundary given a new page instance where new records have been added and/or old records deleted.

Step 3. Once the record list boundary is identified, the third step is **to derive the record schema**. A record schema here refers to not only the internal structure of attributes inside a record but also the extraction rules that extract each individual attributes. It could also be called a record wrapper, borrowing the term "wrapper" from the Web information extraction terminology[4]. The key to derive the record schema is to obtain a set of record instances that could be used as training examples. Once such training examples are available, this sub-problem becomes rather similar to the traditional wrapper generation problem of finding a page wrapper given a set of page instances, which is rather well studied by now [3, 7, 21].

In the second step, we have already obtained the insertions and deletions of records in each page instance. All these insertions and deletions in all page instances could be treated as training examples for record schema. Note that we are not able to guarantee that every insertion and deletion contains exactly only one record. It is possible to have a bulk insert or delete with multiple records together. In this case, the output from the wrapper generation program might not be the schema for a single record. Instead, it could be the record schema with a repetition (Kleene star), i.e., $(R)*$, or the record schema followed by the same record schema marked as optional, i.e., $R(R)?$. In either case, we could still recover the actual record schema, assuming that the actual record schema itself will not be of the form $(R)*$ or $R(R)?$.

Another issue involved with deriving record schema is that the insertions and deletions might not be representative enough to allow the wrapper generation program to output the correct record schema. To validate the generated record schema, we apply the record schema to extract data from the entire record list in each page instance retrieved in the second step. If the extraction is not able to proceed before exhausting the whole record list in some page instance, it means that the corresponding record list contains an unseen variant of the record schema. This conflict could be resolved by adding that record list as another training example and re-run the wrapper generation program. The new output will

[4]In Web information extraction, a *wrapper* refers to a software program that could automatically extract data from a set of heterogeneous pages and convert the data into certain structure.

be of the form $(R')*$, from which the updated record schema R' could be obtained. We can then continue the above validation process using $(R')*$ and resolve any further conflict until all record lists could be successfully exhausted.

Step 4. With the RIC extractors, record list extractor, and record schema, new records could be extracted if the new and old page instances are given. Thus, the next step is to **estimate a page instance fetching schedule**. This schedule is required for determining when to sample the page to retrieve page instances for processing by the three extractors. The actual models and representations of the schedule vary depending on the nature of the page and the actual requirements. They could range from a simple fixed frequency to complex models that balance timeliness and completeness of information [63].

Since this step is not the main focus of this chapter, we adopt a simple model that continuously fetches page instances at a fixed frequency. To estimate the interval between two fetches, we take the interval used in the second step to retrieve page instances as the initial value. If all insertions between every pair of consecutive page instances contain only one record, then we consider this initial value as the estimated fetching interval. Otherwise, we divide the initial value by the maximum number of records found among all insertions, and consider the result as the estimated fetching interval.

Step 5. The last step is to **construct the final record stream wrapper**, using the outputs from the first four steps as building blocks. We adopt a simple combination approach to compose the record stream wrapper with the RIC extractors, record list extractor, record schema, and page instance fetching schedule resulted from the previous steps. Page instances are retrieved from the page periodically based on the derived interval. For each new page instance, the RIC extractors are applied to filter out the record independent changes. Given the new filtered page instance, we can apply the record list extractor to extract the list of records and the record schema to extract the attributes in structured format for each record and compare this new list with the old list extracted from the previous page instance to find out the new records to output. Alternatively, we could compare the new filtered page instance with the previous page instance to find out the insertion and apply the record schema to extract the new structured records from only the insertion. In either way, we can generate new structured records from each new page instance, thus producing a stream of structured records as new page instances are periodically retrieved.

The above five steps together constitute the sequential process in which, given a new page, a record stream wrapper could be derived. For each step, there could be more than one choices of implementation. For example, in the last step, we have given two alternative ways to extract new records from a new filtered page instance. What have been described above are our proposed solutions to each step.

In this chapter, we mainly focus on the steps to produce the three extractors – the RIC extractor, the record list extractor, and the record schema. Since the process for deriving the record schema is relatively more straightforward, it will not be elaborated further. In the next two sections, we will discuss the first two steps in more details.

5.4 Locating Record Independent Changes

Since we are only interested in changes of core content, changes caused by non-core content should be discarded. As non-core content, in particular RIC, usually changes at a different frequency from that of core content, it is difficult to directly locate changes of core content which are often mixed with RICs. On the contrary, it is easier to identify and remove the changes of non-core content because it is more likely to find page instances that only change in non-core content, due to our assumption that RICs change at a higher frequency.

To identify RICs, we first retrieve several page instances that only change in RIC. Next, the changes in these page instances are located and are treated as training examples for training RIC extractors for the RICs. This section elaborates these three steps in details.

5.4.1 Sampling Page Instances

The key to finding record independent changes is to identify occurrences of all these changes in some page instances such that we can use these occurrences to train extractors, or wrappers, that could extract RICs from new page instances. Before we can identify RIC occurrences, the first step is to obtain some page instances that are suitable for this task. These page instances should allow us to easily and correctly compute the occurrences of each RIC and at the same time be as representative as possible such that the wrappers resulted from the training are general and flexible enough to recognize occurrences of the same RICs in unseen page instances of the same page, especially those whose core content is different. In addition, since retrieving page instances requires connecting to remote servers, thus consuming network bandwidth and communication times, we would like to minimize

the number of page instances retrieved and, if possible, share these page instances with subsequent steps.

To ensure that RIC occurrences could be correctly computed, we retrieve page instances that have the same core content but differ in RICs. This is done based on the assumption that RICs change more frequently than records. Given this assumption, within a reasonably short period, the core content should remain the same while the RICs would change when page requests are sent repeatedly within this short period. Therefore, we could obtain page instances sharing the same core content but different RIC occurrences by repeatedly submitting page requests in a short amount of time. Ideally, if all RICs change at each page request, two page instances would allow us to compare and find out their occurrences. However, since there could be more than one RICs in a page and not all of them are changing at the same frequency, we have to fetch more page instances to increase the probability of seeing each RIC changing at least once.

Having multiple page instances with the same core content allows us to train extraction patterns for one RIC that are independent of other RICs. In addition, we also want the trained extraction patterns to be independent of core content, i.e., be valid when core content changes. To ensure that the computed RIC occurrences are sufficiently representative for training in this aspect, we need to see examples with different core content. Therefore, we retrieve, in a similar manner, another set of page instances that has the same characteristics as the previous set, i.e., having same core content but different RICs. However, we deliberately keep the core content of the two sets different, by waiting for a sufficiently long interval between the two batches of page requests to allow the core content to change. Comparisons within this second set of page instances would give us another set of RIC occurrences that appear in page instances that differ from the previous set in core content. We repeat the same process several times to obtain more sets of page instances with different core content and more set of RIC occurrences.

In the process of sampling page instances described above, let the number of page instances fetched within each set be m and the number of sets fetched be n, there will be $m \times n$ page instances retrieved in total. The problem that we study in this step (first step) could be stated as: *given n sets of page instances where within each set, there are m page instances that only differ in record independent changes, and where across different sets, page instances also differ in core content, find wrappers that could extract all record independent changes from new page instances of the same page.*

5.4.2 Identifying Change Occurrences

To find occurrences of record independent changes, we perform comparisons between page instances within each set. Page instances in different sets are not compared, since they differ both in RICs and core content. For page instances within each set, we do a pair-wise comparisons between all of them. The fundamental problem here is how to compare two page instances that share the same core content to find out the occurrences of RICs in these two page instances.

Since the two page instances have the same core content, record independent changes are the only possible changes between them. Unlike core content, whose changes are either insertions or deletions, changes of RICs occur in the form of replacements[5]. To be more accurate, such replacements might not be one-to-one exact mismatch. For example, advertisements before and after changes might have different lengths. To better capture the semantics of changes of RICs, we define a slightly modified concept called *relaxed replacement*, where the replaced portion on the first page instance and the replacing portion on the second might not be of the same length but both lengths must be positive. Our task now is to perform change detection on two given page instances to find out relaxed replacements.

However, before change detection could be applied, the page instances have to be represented in a suitable format. Using the HTML source directly, which is the most straightforward approach, could lead to incorrect matching of HTML tags since the same tag could be used in various places and its semantic meaning is not reflected from the syntactic format. On the other hand, transforming the HTML page into its corresponding DOM[6] tree requires tree-based change detection which is too complicated in our context. Therefore, we have decided to use a path-based representation. For each page instance, its HTML source is parsed to produce a DOM tree, which is in turn converted into a list of root-to-leaf paths. The whole list of a page instance is considered as one string, with each path in the list as one token. A match between two string tokens requires an exact match of the entire path, i.e., all nodes (node name and attributes) in both paths should match. With this representation, page instances comparisons could now be performed by applying standard string comparison algorithms.

[5] In traditional string matching, insertion, deletion, and replacement (or substitution) are the three basic edit operations.

[6] http://www.w3.org/DOM/

```
1: function FINDRELAXEDREPLACEMENT(Str1, Str2)
2:   Len1 = STRLEN(Str1), Len2 = STRLEN(Str2)
3:   InfoMatrix = new Info[Len1][Len2]
4:   InfoMatrix[0][0].cost = 0; InfoMatrix[0][0].op = Init
5:   for i = 1 to Len1 do
6:     InfoMatrix[i][0].cost = i × cost_d; InfoMatrix[i][0].op = Del
7:   end for
8:   for j = 1 to Len2 do
9:     InfoMatrix[0][j].cost = j × cost_i; InfoMatrix[i][0].op = Ins
10:  end for
11:  for i = 1 to Len1 do
12:    for j = 1 to Len2 do
13:      InfoMatrix[i][j].cost = min(InfoMatrix[i-1][j].cost + cost_i,
             InfoMatrix[i][j-1].cost + cost_d, InfoMatrix[i-1][j-1].cost + cost_m)
14:      assign InfoMatrix[i][j].op to Ins, Del, or Match, accordingly
15:    end for
16:  end for
17:  i = Len1; j = Len2; i_p = Len1; j_p = Len2;
18:  while InfoMatrix[i][j].op != Init do
19:    if InfoMatrix[i][j].op == Match then
20:      if i < i_p && j < j_p then
21:        substring pair (Str1[i + 1..i_p], Str2[j + 1..j_p]) is one relaxed replacement
22:      end if
23:      i--; j--; i_p = i; j_p = j;
24:    else if InfoMatrix[i][j].op == Ins then
25:      j--;
26:    else if InfoMatrix[i][j].op == Del then
27:      i--;
28:    end if
29:  end while
30:  return all relaxed replacements found
31: end function
```

Figure 5.2: The FINDRELAXEDREPLACEMENT Algorithm

The problem of finding relaxed replacements between two strings is equivalent to the problem of finding longest common subsequence (LCS) of two strings. Although there are faster LCS algorithms [32], we use a dynamic programming algorithm for a simple implementation. The algorithm, as shown in Figure 5.2, is basically a weighted edit distance algorithm, where the cost for a match ($cost_m$), insertion ($cost_i$), deletion ($cost_d$), and replacement are 0, 1, 1, and 10000, respectively (line 4–16). Replacement is ignored in this algorithm since its cost is deliberately set to be significantly higher than the total cost of an insertion and a deletion. Insertions and deletions will always take precedence over re-

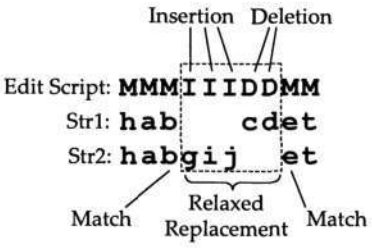

Figure 5.3: An Example of Finding Relaxed Replacement.

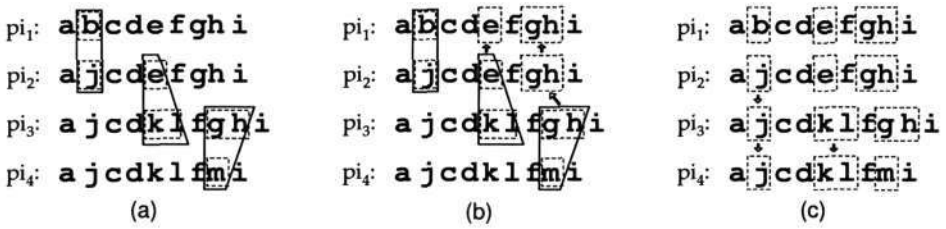

Figure 5.4: An Example of Occurrences of RICs.

placements. In the case of a tie in determining the minimum in line 13, priority is given first to (i-1,j) (insertions), then to (i,j-1) (deletions). This allows the replacing string segment in the second string in a relaxed replacement to be listed (in the edit transcript) as a group of insertions before the deletions of the replaced segment in the first string. From the resulted edit transcript, it is straightforward to compute the relaxed replacements: each group of consecutive insertions followed by a group of consecutive deletions is considered as one relaxed replacement (line 17–29), as illustrated in figure 5.3.

Given the relaxed replacements found, the change occurrences on both page instances could be easily obtained by mapping the replacements back to the two path lists. With the changes in all pages in a set found, the task now is to map these change occurrences to the RICs, i.e., associating each change occurrence with a RIC. The most straightforward scenario is when all RICs change in every page instance. In this case, if there are k RICs in the page, we will find k change occurrences in each page instance. Based on the assumption that the relative positions of all RICs do not change over page instances, we only need to order all change occurrences in each page according to their positions and group the changes in all page instances based on their ordered positions, e.g., the changes appearing first in all page instances are considered as occurrences of the first RIC in the page.

However, sometimes, some RICs may not change in some page instances. In this case, the number of relaxed replacements found by comparing these page instances is less than

the number of RICs. To solve this problem, in the sampling step, we fetch multiple (m) page instances of the same core content, in a hope to see each RIC changing at least once. If this condition is satisfied, we will still be able to map all found change occurrences to their corresponding RICs. To do this, we order all pages in a set according to the time they are retrieved and compare every two consecutive page instances to find the relaxed replacements and change occurrences, as shown in Figure 5.4a. If each RIC changes at least once, each RIC must have at least one occurrence in one of the $m-1$ comparisons. We take one page instance, say the first one (pi_1 in Figure 5.4a), and map each change occurrence in the other $m-1$ page instances on to pi_1, as shown in Figure 5.4b). By mapping all changes in these $m-1$ page instances on to the first page instance, we will know the change occurrences of all RICs in the first page instance. These change occurrences could then be propagated back to other page instances through the matches found during comparisons (See Figure 5.4c).

Therefore, as long as we can guarantee that each RIC changes at least once in the m page instances in a set, we will be able to identify all occurrences of all RICs in all these m page instances. Repeating the above process for the other n-1 sets, we can find all occurrences of all RICs in all the retrieved samples. If in some set, the condition that each RIC changes at least once is not met, we will find less number of RIC occurrences in each page instance[7]. When this happens, we ignore the set entirely, hoping that the rest of the sets could give us sufficient data to train the RIC extractors. This could be done by checking the number of RIC occurrences per page in each set and keeping only those sets with the highest number.

5.4.3 Training RIC Extractors

Given all occurrences of the RICs in all page instances, the problem is now reduced to a wrapper generation problem. We deal with the RICs one by one. For each RIC, we take all its occurrences in the $n \times m$ page instances as training examples and feed them into a wrapper generation program. Wrapper generation for Web pages has been a rather well researched problem since it was first proposed in 1997 [42]. In our case, generating extraction patterns for one RIC at a time is a single-slot extraction problem [60]. The output from the training, or wrapper generation, process is a wrapper that is capable of

[7]The number of RIC occurrences that could be found is equal to the number of RICs that change at least once.

identifying occurrences of the RIC given all the retrieved page instances and any new page instances. As mentioned in Section 5.3, we call this wrapper an RIC extractor.

The essential component of a wrapper is the extraction patterns that determine how to locate the portion that we want to extract within a page instance. In this chapter, instead of the more general but complex regular expressions, we use a simple type of extraction patterns, which is a four-tuple $< p_b, n_b, p_e, n_e >$. In this pattern, p_b and p_e, called *beginning pattern* and *ending pattern*, are the tokens that appear immediately before and after the occurrences of a RIC. Since page instances are represented as a list of paths, both p_b and p_e are in fact just two paths. The other items n_b and n_e denote the number of repeated appearances, from the beginning and end of the page instance respectively, that we should count for p_b and p_e. For example, $<$ html/body/hr, 2, html/body/p/b, 1 $>$ locates an RIC by looking for the second appearance of the path html/body/hr from the beginning of a page instance and the first appearance of the path html/body/p/b from the end. The paths in-between the two paths found are an occurrence of the RIC.

The wrapper generation process for producing such extraction patterns is a simple generalization from all paths appearing before and after the occurrences given as training examples. Repeating this generation process for all k RICs gives us k RIC extractors. With these RIC extractors, we have completed the tasks in the first step of the framework.

5.5 Finding Record List Boundary

This section focuses on the second step in the framework. As described in Section 5.3, the goal of the second step is to find out the boundary of the list of records in the page and whether it grows upwards or downwards.

Since records are inserted into one end of the list and deleted from the other end, we can identify the two ends of the list by monitoring the locations at which new records are added and old records are removed. To do this, we need to retrieve page instances that have different records. We then try to locate the insertions and deletions by comparing these page instances. From the movement of the locations of insertions and deletions on the page instances, we can determine the growth direction and the boundary of the record list.

5.5.1 Filtering Page Instances

To find insertions and deletions, the two page instances being compared should at least have some records in common. Preferably, they should differ only slightly – one or two records inserted, one or two records deleted, or both. There should also be an ordering in the page instances retrieved, since we need to know which page instance is new and which one is old so that we can tell whether the changes are insertions or deletions.

In step 1, we retrieved n sets of page instances where the core content, i.e., the records, are different across different sets. We could reuse these page instances in this step, which also fulfils the requirements of reducing network bandwidth usages and sharing page instances as mentioned in Section 5.4.1. Since these page instances are sampled chronologically from the URL, there is a natural ordering based on the time they are retrieved. To keep the number of inserted or deleted records between the consecutive page instances to minimum, we try to adjust the time interval between sampling two different sets of page instances such that it is long enough to see changes in the records and at the same time not too long so that most records still remain in both sets.

Recall that each set actually has m page instances that share the same core content. We apply the RIC extractors obtained from the previous step to remove all the record independent changes, resulting in what we call *filtered page instances.* An important consequence of this filtering is that all page instances within the same set become identical, because the only differences among them, the RICs, have now been removed. Therefore, at the end of the removal of RICs, we have a sequence of n distinct filtered page instances, ordered by the time that they are retrieved.

The problem that we study in this step now becomes: *Given a list of n filtered page instances that are chronologically ordered, find the boundary that encloses the list of records and the growth direction of the list.*

5.5.2 Identifying Insertions and Deletions

The changes between consecutive filtered page instances contain only insertions and deletions but no replacements. This is exactly the opposite of that of record independent changes, which consist of only (relaxed) replacements.

An important property of the changes in records is that the differences between two consecutive filtered page instances are either insertions at the list head, deletions at the

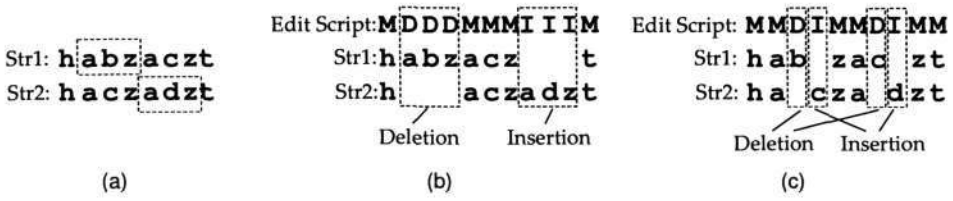

Figure 5.5: An Example of Insertions and Deletions.

tail, or both. This property is derived based on the assumption that new records are added at one end of the list consistently and old records deleted at the other end in the order that they are added. Since insertions and deletions only occur at both ends of the record list, once we have identified these insertions and deletions, we will have clues on where is the boundary of the list and in which direction the list is grown.

Similar to the problem of finding occurrences of RICs, we can first focus on the fundamental problem of comparing two consecutive filtered page instances. The same path-based representation is adopted. However, the weighted edit distance dynamic programming algorithm for finding relaxed replacements is not applicable to this problem. The main reason is that due to the fact that records share the same schema, they will inevitably have some tags and texts in common, even though the core content might change. In the basic algorithm, these common tags and texts might be treated as matching characters, with the core content in-between them as replacements. For example, consider the two strings in Figure 5.5a, where in the second string, "abz" is the deleted record and "adz" is the inserted record. The correct comparison result should be as in Figure 5.5b. However, using the basic algorithm, a mapping shown in Figure 5.5c would be found, where the common characters 'a' and 'z' are matched incorrectly, causing four individual insertions and deletions to be found (with a cost of 4), instead of two chunks of insertion and deletion (with a cost of 6).

In order to arrive in the mapping in Figure 5.5b, instead of that of Figure 5.5c, we need to adjust the costing scheme of edit scripts to favour large consecutive segments of insertions or deletions over smaller chunks. For example, if we consider a simple scheme of assigning the same cost to consecutive insertions/deletions and to the single ones, then the mapping in Figure 5.5b would have a cost of 2, which would allow it to be selected when compared with that of Figure 5.5c, which still has a cost of 4. However, using such costing scheme requires that the algorithm, at step (i, j), know the mapping details, especially the unmatched characters at the end, of the best mappings at steps $(i\text{-}1,j)$, $(i,j\text{-}1)$, and $(i\text{-}1,j\text{-}1)$. This

```
1: function FINDINSERTIONDELETION(Str1, Str2)
2:     Len1 = STRLEN(Str1), Len2 = STRLEN(Str2)
3:     MappingsMatrix = new Mappings[Len1+1][Len2+1]
4:     for i = 0 to Len1 do
5:         MappingsMatrix[i][0].Add( Mapping(i, 0, i, 0) )
6:     end for
7:     for j = 1 to Len2 do
8:         MappingsMatrix[0][j].Add( Mapping(0, j, 0, j) )
9:     end for
10:    for i = 1 to Len1 do
11:        for j = 1 to Len2 do
12:            for each mapping Mapping in MappingsMatrix[i-1][j] do
13:                if Mapping.tail2 == 0 then
14:                    MappingsMatrix[i][j].Add( Mapping(i, j, Mapping.tail1+1, 0) )
15:                else
16:                    for k = Mapping.tail2 to 1 do
17:                        if MATCH(Str1[i], Str2[Mapping.index2+k-1]) then
18:                            MappingsMatrix[i][j].Add( Mapping(i, j, 0, k-1) )
19:                        end if
20:                    end for
21:                end if
22:            end for
23:            for each mapping Mapping in MappingsMatrix[i][j-1] do
24:                if Mapping.tail1 == 0 then
25:                    MappingsMatrix[i][j].Add( Mapping(i, j, 0, Mapping.tail2+1) )
26:                else
27:                    for k = Mapping.tail1 to 1 do
28:                        if MATCH(Str1[Mapping.index1+k-1], Str2[j]) then
29:                            MappingsMatrix[i][j].Add( Mapping(i, j, k-1, 0) )
30:                        end if
31:                    end for
32:                end if
33:            end for
34:            if MATCH(Str1[i], Str2[j]) then
35:                for each mapping Mapping in MappingsMatrix[i-1][j-1] do
36:                    MappingsMatrix[i][j].Add( Mapping(i, j, 0, 0) )
37:                end for
38:            end if
39:            group all mappings in MappingsMatrix[i][j] according to their (tail1, tail2) values and keep only the
               mapping with lowest cost for each distinct pair
40:        end for
41:    end for
42:    for each mapping Mapping in MappingsMatrix[Len1][Len2] do
43:        compute cost by adding insertion cost for tail1 > 0 and deletion cost for tail2 > 0
44:    end for
45:    let BestMapping be the mapping in MappingsMatrix[Len1][Len2] that has the lowest cost
46:    compute insertions and deletions from BestMapping
47:    return the computed insertions and deletions
48: end function
```

Figure 5.6: The FINDINSERTIONDELETION Algorithm

Figure 5.7: Illustrations of the FINDINSERTIONDELETION Algorithm.

violates one of the fundamental assumptions of the basic dynamic programming algorithm – the cost of the best mapping at step (i, j) depends only on the cost of steps $(i\text{-}1,j)$, $(i,j\text{-}1)$, and $(i\text{-}1,j\text{-}1)$ but not their mapping details (see line 13 of Figure 5.2). Therefore, to be able to apply the new costing schemes to the basic algorithm, at each step (i, j), we need to keep not only the costs but also the mapping details required for computing the cost of mappings of future super-strings. In addition, we also need to keep not just one best mapping but all the best mappings whose cost could possibly decrease later.

Based on the above ideas, we design a new algorithm that is also based on dynamic programming (See Figure 5.6) to correctly identify insertions and deletions. The main difference from the algorithm for relaxed replacement is that for each step (i, j), instead of keeping only the best from all computed mappings, it distinguishes mappings that differ in the unmatched characters at the end of both substrings. For example, for the two strings in Figure 5.5a, at step $(5, 5)$, the algorithm considers the three mappings in Figure 5.7 and stores the best cost of all three[8], since the unmatched characters in these three mappings (as highlighted in the dotted rectangles) are different. From the first mapping, the best mapping of the two complete strings (Figure 5.5b) could be derived later. On the contrary, the basic algorithm would only keep the mapping in the middle and ignore the first and third mappings in Figure 5.7; thus it will have no way to arrive in the best mapping in Figure 5.5b.

The algorithm makes use of a 2-D array, named MappingsMatrix, to store the intermediate results of the dynamic programming. Each cell in the 2-D array is a vector of a data structure named Mapping. As shown in Figure 5.6, instances of Mapping are created using the constructor Mapping(index1, index2, tail1, tail2), where index1 and index2 are the current

[8]In fact, there are other mappings with different unmatched characters. For simplicity, we show only the three most important ones.

position in each string that are being compared, i.e., i and j in each step (i, j), and tail1 and tail2 are the number of unmatched characters at the end of each string. The Mapping structure also contains other members that are not shown here for simplicity.

The algorithm starts by initializing first row and first column of MappingsMatrix with mappings that have the first string and second string, respectively, as unmatched characters (line 4–10). Next, the dynamic programming portion (line 11–41) is executed to find the best mappings for each substring pair (the first i characters of the first string and first j of the second). For each step (i, j), the new mappings are computed from existing mappings from steps $(i-1, j)$, $(i, j-1)$, and $(i-1, j-1)$. From step $(i-1, j)$ (line 13–23), we are to add the ith character of the first string into the mappings. If there are no unmatched characters in the second substring, i.e., $tail2 == 0$, we construct a new mapping by simply appending the new character to the first substring as an additional unmatched character (line 15). Otherwise, we compare the ith character of the first string with each unmatched character in the second substring, i.e., the last $tail2$ characters (line 17–21). If there is a match, we create a new mapping based on the matching characters (line 19). A similar process is repeated for computing from step $(i, j-1)$ (line 24–34). For computing from step $(i-1, j-1)$, if the ith character of the first string and the jth character of the second matches, we will create one new mapping from each existing mapping in step $(i-1, j-1)$ (line 36–38). At each step, we keep for each distinct $(tail1, tail2)$ pair the mapping with the lowest cost (line 40).

After finding all mappings, the candidates of the best mapping for the two complete strings are in MappingsMatrix[Len1][Len2]. Since the cost of mappings at each step do not include the cost resulted from the unmatched characters at the end of both substrings, we need to update the cost by considering these unmatched characters (line 43–45). This is done by adding insertion cost if $tail2 > 0$ and adding deletion cost if $tail1 > 0$ (line 44). The best mapping can then be picked by choosing the mapping with lowest cost. Form the best mapping, we can compute the insertions and deletions by considering the all unmatched substring segments in the second string as insertions and all unmatched substring segments in the first string as deletions. The algorithm terminates after computing all insertions and deletions.

5.5.3 Deriving List Boundary and Growth Direction

By applying the algorithm to every consecutive pair of filtered page instances, i.e., instances 1 and 2, 2 and 3, ..., $n-1$ and n, we can obtain $n-1$ comparison results, with $n-1$ sets of insertions and deletions. Similar to the mapping of RIC occurrences to RICs in Section 5.4.2, we can propagate and map all these $n-1$ sets of insertions and deletions to all the n filtered page instances.

We will first determine the growth direction of the list. Based on the kinds of changes that we found, there could be three different scenarios: (1) The simplest case is when we find both insertion and deletion in one comparison. From the relative position of the insertion and deletion, we are able to tell the growth direction immediately. For example, if the insertion appears before the deletion, we know that the list grows *upwards*. (2) However, since we deliberately try retrieving page instances that only change slightly in the records, it is very likely that the changes between two consecutive filtered page instances only contain one insertion or one deletion. In such situation, we are not able to determine the growth direction of the list from one insertion or one deletion alone, since the list could be growing in either way. To make full use of all $n-1$ comparisons, we map all $n-1$ sets of insertions and deletions found onto any one of the filtered page instance, say the first one. If among these $n-1$ comparisons, there are at least one insertion and one deletion, these insertions and deletions will be mapped onto the same page instance; hence we are able to find out the growth direction as in the simplest case. (3) A slightly more complicated situation occurs when we only found one type of changes, either all insertions or all deletions. In this case, we still map all insertions or deletions onto one page instance. If the insertion (deletion) obtained from comparing page instances 2 and 3 appears before that of 1 and 2, then the list grows upwards (downwards) and vice versa.

Once we know the growth direction of the record list, we can easily figure out the list boundary on each filtered page instance. We take the first deletion and the last insertion found in the $n-1$ comparisons and map them onto each page instance. If the list grows upwards (downwards), the beginning (ending) position of the last insertion and ending (beginning) position of the first deletion are the two positions marking the list boundary. However, for case (3) described above, we would only be able to identify one side of the boundary of the list. We take the beginning or ending of the string as the other side, depending on the growth direction and whether we have found insertions or deletions.

5.5.4 Training Record List Extractors

By combining the FindInsertionDeletion algorithm with the techniques described in the previous section, we can obtain boundaries of record list in all filtered page instances. The task in this section is similar to that of Section 5.4.3, i.e., it could be considered as another instance of a wrapper generation problem. The only difference is that now we only need to generate one wrapper (or extractor), instead of k wrappers, one for each RIC. We use the same simple extraction pattern to represent the record list extractor and the same wrapper generation process.

5.6 Experiments

We have implemented prototypes of the two algorithms proposed in the previous two sections. In this section, we conduct experiments to test their performance.

5.6.1 Dataset

The dataset used in the experiment consists of 9 distinct pages from various domains. For each page, the sampling technique is applied to retrieve n sets of page instances, with m page instances per set. In addition, for each page, k new page instances are also fetched at random time points within the next t period, where t is the time taken to fetch the $n \times m$ page instances. The $n \times m$ page instances will be used to generate record stream wrappers; whereas the k page instances will be used to test the generated wrappers. The expected results for all the $9 \times k$ page instances are manually identified. In this experiment, we set $n = 5$, $m = 5$ and $k = 5$, yielding a total of 270 page instances.

5.6.2 Methodology and Performance Metrics

In this experiment, we focus on the performance of the first two steps in the proposed framework, especially the two proposed algorithms. For the first step in the framework, we measure the performance in two aspects. Firstly, we want to know whether the algorithm for finding relaxed replacements is able to find out all RICs correctly from the training examples, i.e., the $n \times m$ page instances. For each page, we compute the precision p_{rrn} and recall r_{rrn} in terms of number of RICs:

$$p_{rrn} = \frac{no.\ of\ correctly\ found\ RICs}{no.\ of\ relaxed\ replacements\ found}$$

$$r_{rrn} = \frac{no.\ of\ correctly\ found\ RICs}{no.\ of\ actual\ RICs}$$

Secondly, we also want to see how effective the generated RIC extractors are in terms of extracting occurrences of RICs in new page instances. For each page, we apply the generated RIC extractors to find occurrences of RICs in the k new page instances and compute the precision p_{rre} and recall r_{rre} as:

$$p_{rre} = \frac{no.\ of\ correctly\ extracted\ RIC\ occurrences}{no.\ of\ extracted\ RIC\ occurrences}$$

$$r_{rre} = \frac{no.\ of\ correctly\ extracted\ RIC\ occurrences}{no.\ of\ actual\ RIC\ occurrences}$$

For the second step, we measure the performance of the three sub-steps. Firstly, for the algorithm for finding insertions and deletions, we compute, for each page, the precision p_{idn} and recall r_{idn} from the n filtered page instances as:

$$p_{idn} = \frac{no.\ of\ correctly\ found\ insertions\ and\ deletions}{no.\ of\ insertions\ and\ deletions\ found}$$

$$r_{idn} = \frac{no.\ of\ correctly\ found\ insertions\ and\ deletions}{no.\ of\ actual\ insertions\ and\ deletions}$$

Secondly, we also want to see how effective our method of identifying record list boundary and growth direction is. We compute the precision of record list boundary discovery p_{idb} and growth direction identification p_{idg} as:

$$p_{idb} = \frac{no.\ of\ pages\ with\ correctly\ identified\ boundary}{no.\ of\ pages}$$

$$p_{idg} = \frac{no.\ of\ pages\ with\ correctly\ identified\ growth\ direction}{no.\ of\ pages}$$

Thirdly, we measure the effectiveness of the generated record list extractors by computing, for each page, the precision p_{ide} and recall r_{ide} of extracting record list boundary from the k new filtered page instances:

$$p_{ide} = \frac{no.\ of\ correctly\ extracted\ record\ lists}{no.\ of\ record\ lists\ successfully\ extracted}$$

$$r_{ide} = \frac{no.\ of\ correctly\ extracted\ record\ lists}{k}$$

For each p and r pair, the F1-measure f is also calculated:

$$f = \frac{2pr}{p+r}$$

Table 5.1: Performance of the First Step

	Site	p_{rrn}	r_{rrn}	f_{rrn}	p_{rre}	r_{rre}	f_{rre}
1	Zaobao	0	0	-	0	0	-
2	Reuters Business News	1	0.667	0.8	1	0.667	0.8
3	NYTimes Technology	1	0.5	0.667	1	0.5	0.667
4	Slashdot	0.2	1	0.333	0.2	1	0.333
5	TheServerSide	0.3	1	0.462	0.3	1	0.462
6	DevX News	1	1	1	1	1	1
7	DevX Java Articles	1	0.75	0.857	1	0.75	0.857
8	CodeGuru	1	1	1	1	1	1
9	DBWorld	-	-	-	-	-	-
	Average	0.688	0.74	0.731	0.688	0.74	0.731

5.6.3 Results and Discussion

The results of step 1 are shown in Table 5.1. Except the DBWorld page, all other pages contain some record independent changes (RIC). For the performance of the algorithm (3rd to 5th columns), 5 out of 8 achieve 100% precision and 4 out of 8 have 100% recall. For page 1, 2 and 3, the reason of having a low recall is because some or all RICs change less frequently than the records, which makes them appear more like static data. This is a violation of one of the core assumption made by the algorithm; hence the algorithm failed to find out these RICs. For page 7, there is one RIC that is generated by a JavaScript function which in turn displays random advertisement. Since the JavaScript function call appears to be the same in the HTML source, the algorithm does not consider it as an RIC. For precisions, the 0 precision of page 1 is due to the same reason as its 0 recall. For page 4 and 5, it is because each record in these two pages contains one attribute for the number of comments about the news or article. For some page instances within the same set, the number of comments changed, causing them to be identified as relaxed replacements by the algorithm. This effect leads to a large number of incorrect relaxed replacements being found, which in turns leads to low precision values. This is in fact another violation of our assumptions, i.e., record attribute values do not change.

For the performance of RIC extractor (6th to 9th columns), it is interesting to note that they have the same values as that of 3rd to 5th columns. This indicates that for those RICs correctly identified by the algorithm, the extraction did not produce any error, i.e., all errors are inherited from the inefficiency in the algorithm for finding RIC. In other

Table 5.2: Performance of the Second Step

	Site	p_{idn}	r_{idn}	f_{idn}	p_{idb}	p_{idg}	p_{ide}	r_{ide}	f_{ide}
1	Zaobao	1	1	1	1	1	1	1	1
2	Reuters Business News	1	1	1	1	1	1	1	1
3	NYTimes Technology	1	1	1	1	1	1	1	1
4	Slashdot	1	0.818	0.9	1	1	1	1	1
5	TheServerSide	1	0.8	0.889	1	1	1	1	1
6	DevX News	1	0.833	0.909	1	1	1	1	1
7	DevX Java Articles	1	1	1	1	1	1	1	1
8	CodeGuru	1	1	1	1	1	1	1	1
9	DBWorld	1	1	1	1	1	1	1	1
	Average	1	0.939	0.966	1	1	1	1	1

words, the simple extraction language and wrapper generation process indeed work very well. This is mainly because all pages are generated by server-side programs and their HTML templates are rather well structured.

For the performance of the algorithm for finding insertions and deletions in the second step, most pages achieve 100% results, except the recall values of page 4, 5 and 6, as shown in Table 5.2. Further investigation revealed two reasons. The first one (for page 4 and 5) is the same as that in the first step, i.e., the attribute for the number of comments changes even within the same set of page instances, making it difficult to match the same records in two consecutive page instances. The second reason (for page 6) is that some old records are updated and moved to the top again as if they are new records. This violates the assumption that new records are added consistently from one direction in certain order and old records are deleted in the order that they are added.

For record list boundary (6th column) and growth direction discovery (7th column), since the methods that we use are rather straightforward, it is not unexpected to observe 100% precision for both. It should be noted that the failures in identifying some insertions and deletions in some filtered page instances do not affect the discovery of record list boundary and growth direction. This is because the rest of the correctly found insertions and deletions still provide sufficient information for us to derive the record list boundary and growth direction correctly.

For record list extraction performance (8th to 10th columns), the good performance could be attributed to the same reason as in the first step, i.e., the simple extraction language and wrapper generation process are good enough for these pages.

It should also be pointed out that even though the averaged performance of the first step is not very good for some pages, the degradation is not propagated to the second step. This is because a majority of the failures in the first step is due to RICs changing less frequently than records. If these RICs do not change in all $n \times m$ page instances, then the second step would still be able to derive a correct record list extractor.

5.7 Related Work

As discussed in Section 5.1, the problem that we study in this chapter is related to several research areas. *Wrapper generation* deals with the problem of creating software programs, called wrappers, that could automatically extract structured data from a set of Web pages. Wrappers could be created manually [24, 42]; but most recent research focuses on semi-automatic [7, 50, 57] and automated approaches [3, 21] to learn wrappers from (selected) examples. These works do not consider the dynamic aspect of Web pages. Although it might be possible to apply wrappers to extract data from the entire page instances, much post-processing is still required in order to produce a stream of records. In addition, core content and non-core content are not distinguished by these systems. Another important difference lies in sample selection: training samples of wrapper generation systems are usually manually selected; whereas our proposed framework automatically samples the page with varying intervals to obtain page instances. Nevertheless, wrapper generation systems are complementary to our approach and are used in several components within the proposed framework.

Record boundary discovery, which aims to identify boundary of records, has also been studied as part of the general wrapper generation problem [13, 25]. The general approach is to apply some heuristics to find out the DOM subtree that contains all records and then to compute some confidence scores on which tag or pair of tags are the most likely to be the record boundary separator [13, 25]. The boundary of record list discovered (the DOM subtree) usually only indicates the rough area that covers the actual list boundary. The discovery of record boundary relies on several heuristics derived from observations and domain knowledge [25]. Unlike above approaches, the list boundary that we obtain from the movement of insertions and deletions is a tight boundary. Moreover, identifying records (insertions and deletions) by comparisons does not depend on heuristics and domain knowledge.

Another area of closely related research is *change detection* of Web data. Early works in change detection on semi-structured and Web data focus on defining appropriate change operators and finding efficient ways of computing changes [17, 49]. These techniques fail to distinguish the core content from record independent changes. Subsequent works [11, 27, 52, 65] on Web monitoring and continuous query detect changes in a Web site or Web pages at a rather coarse granularity – they consider either one page or a page fragment as the basic unit of change. Changes are detected as they are and no further semantic processing, such as grouping records into a list and discovering record schema, is applied.

Other change detection and monitoring algorithms [18, 63] focus on scheduling page fetches to discover changes under certain constraints, typically limited network bandwidth. In [63], a general-purpose algorithm is proposed to monitor Web information sources with the goal of balancing the completeness and timeliness of the changes detected. As mentioned in Section 5.3, scheduling page fetches is not the focus of this chapter, even though it is in fact a sub-problem under the general framework that we proposed. These techniques are complementary to our proposed algorithms for the first and second steps of the framework. They could be integrated into our framework as solutions to the fourth step.

5.8 Summary

In this chapter, we study the problem of extracting streams of structured data from Web pages that are frequently updated. We propose a framework for solving this problem that permits a divide-and-conquer methodology to be adopted. The framework consists of five steps with each step being an almost independent problem to be studied. We focus on the first two steps and propose algorithms for sub-problems encountered in these steps. The key ideas underlying our approach are repeated sampling with varying intervals to obtain training page instances and two types of comparisons between page instances for finding out relaxed replacements and insertions and deletions. Experimental results show that as long as the assumptions that we make are not violated, the algorithms perform very well. Even when some assumptions are violated, the effect of such violation is minimized and the final discovery of record list boundary still work quite well.

Chapter 6

Conclusions

Information extraction from the Web is widely considered as one of the solutions to the *information overloading* problem that most home users are facing. Since the requirements come from the home users, the ultimate judgement of whether a Web information extraction system is successful still lies in the hands of these users who do not really care about the internal technologies being used. What matters most to them is how well the problem of information overloading has been solved. In this thesis, we focus on one of the important criteria – how easy it is to generate a wrapper. This criterion, when quantified, becomes the amount of time taken by a user to generate a wrapper for a given Web site, which is indirectly affected by the degree of automation and the easy-to-useness of the system.

In this thesis, we adopted a semi-automatic wrapper generation approach to solve the problem. The end result is a software tool (Mapping Wizard) that assists users in creating a wrapper for a given Web site. With the conceptual separation of logical and physical views of Web sites and a clear process for wrapper creation, it is much easier for users to use the tool. Our decision to develop a collection of small tools each addressing one sub-issue was proven to be an adequate approach. One important advantage of this approach is that it is extensible and flexible – as new issues are discovered, new tools could be added to handle them. We demonstrated the feasibility of this approach by studying two new problems of quickly building Web site skeleton and handling frequently updated Web pages. The proposed algorithms to these two problems could be easily incorporated into the Mapping Wizard and made part of the wrapper creation process. Therefore, we believe that semi-automatic wrapper generation is a viable approach to Web information extraction, hence to the problem of information overloading.

6.1 Future Work

In this section, we discuss some potential improvements to our work and list out some new challenges to be met as part of the future work.

- The data model that we proposed in Chapter 3 does not allow structural disjunction. The extraction rules allowed in this model is relatively simple. We would like to see how WDM can be extended to include more expressive and flexible features while maintaining the degree of automation of the current wrapper generation tools. Another interesting extension would be to investigate how the extraction rules and their induction algorithms from other systems can be incorporated into the WICCAP Data Model.

- The low precision obtained in the experiments in Chapter 4 indicateed that there is still much room to improve the SEW algorithm. In particular, techniques are required to prevent the algorithm from retrieving too many incorrect pages. One possible way is to have better features that distinguish navigation links from others so that incorrect link sets would be more likely to be rejected.

- In Chapter 5, the basic assumption of one page having one record list might be violated in more complex Web pages. We would like to study how the algorithm could be extended to handle pages with more than one record list.

- Web sites from the same domain often share similarity in terms of site structure and data presentation style. It is possible to build wrappers that can remember data structure models and extraction rules derived previously for sites of the same genre. Such knowledge could be accumulated incrementally so that as more and more knowledge is collected, less and less effort is required to generate a wrapper for a new source of a known category.

- A much larger amount of information can only be retrieved by filling in HTML forms. This huge collection of information is often referred to as the *Hidden Web*. The mechanism provided in WICCAP Data Model allows access to information from the Hidden Web. However, the creation of WDM elements for HTML forms is still not very easy. We would like to investigate how the process of extracting information from the Hidden Web can be further automated.

- As one of the important issues of Web information extraction, wrapper maintenance is certainly an area that we would like to explore. Current research has been mainly focusing on dealing with simple layout change while ignoring structural changes. We would like to look into issues involved in automatic detection of structural changes of Web sites, perhaps by extending the SEW algorithm, and reparation of wrappers when changes are detected.

Appendix A

A Mapping Rule of BBC Online News

```
<!-- WiccapNewsBBC.xml -->
<?xml version="1.0" encoding="UTF-8"?>
 <Wiccap Name="BBC" Group="News"
          xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="WiccapNews.xsd">
   <Mapping><Link Type="Static">http://news.bbc.co.uk</Link></Mapping>
   <Section Name="World">
       <Mapping>
           <Link Type="Dynamic"><Locator Type="ByPattern">
               <LocatorPattern>
                   <BeginPattern>
                       <![CDATA[<TD CLASS="rootsection" BGCOLOR="#FFFFFF"><A HREF="]]>
                   </BeginPattern>
                   <EndPattern><![CDATA[" CLASS="index">]]></EndPattern>
               </LocatorPattern>
           </Locator></Link>
       </Mapping>
       <Region Name="ArticleList">
           <Mapping><Locator Type="ByPattern">
               <LocatorPattern>
                   <BeginPattern><![CDATA[<SPAN CLASS="sectiontitle">]]></BeginPattern>
                   <EndPattern><![CDATA[<SCRIPT LANGUAGE=]]></EndPattern>
               </LocatorPattern>
           </Locator></Mapping>
           <Record Name="Article" NumOfItem="3">
               <Mapping><Locator Type="ByPattern"><LocatorPattern>
                   <BeginPattern><![CDATA[<DIV CLASS="]]></BeginPattern>
                   <EndPattern/>
               </LocatorPattern></Locator></Mapping>
               <Item Type="Link" TagFilter="Off" Description="This is the link to the news">
                   <Mapping><Locator Type="ByPattern"><LocatorPattern>
                       <BeginPattern><![CDATA[<a href="]]></BeginPattern>
                       <EndPattern><![CDATA[">]]></EndPattern>
                   </LocatorPattern></Locator></Mapping>
               </Item>
               <Item Type="Title" Description="This is the title of the news">
                   <Mapping><Locator Type="ByPattern"><LocatorPattern>
                       <BeginPattern><![CDATA[<B class="h***]]></BeginPattern>
                       <EndPattern><![CDATA[</B>]]></EndPattern>
                   </LocatorPattern></Locator></Mapping>
               </Item>
               <Item Type="Description" Description="This is the description of the news">
                   <Mapping><Locator Type="ByPattern"><LocatorPattern>
                       <BeginPattern><![CDATA[</A>]]></BeginPattern>
```

```
                    <EndPattern><![CDATA[<BR]]></EndPattern>
                </LocatorPattern></Locator></Mapping>
            </Item>
        </Record>
    </Region>
    <Section Name="Asia-Pacific">
     : : :
    </Section>
    <Section Name="Americas">
     : : :
    </Section>
  </Section>
  <Section Name="Business">
   : : :
</Wiccap>
```

Appendix B

An Example of Extracted Data of BBC Online News

```
<!-- WiccapNewsContentBBC.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<Wiccap Name="BBC" Group="News" Link="http://news.bbc.co.uk"
         xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="WiccapContentNews.xsd">
  <Section Name="World">
    <Region Name="ArticleList" NumOfRecord="3">
      <Record Name="Article" NumOfItem="3">
        <Item Type="Link">/hi/english/world/middle_east/newsid_1860000/1860273.stm</Item>
        <Item Type="Title">Israel targets refugee camps</Item>
        <Item Type="Description">Israel occupies two refugee camps in its biggest assault
            since the Palestinian uprising began 17 months ago.</Item>
      </Record>
      <Record Name="Article" NumOfItem="3">
        <Item Type="Link">/hi/english/world/middle_east/newsid_1859000/1859710.stm</Item>
        <Item Type="Title">Iraq has 'positive' talks with UN</Item>
        <Item Type="Description">Iraq's foreign minister meets UN head Kofi Annan for the
            first time in a year as US pressure mounts for action against Baghdad.</Item>
      </Record>
      <Record Name="Article" NumOfItem="3">
        <Item Type="Link">/hi/english/world/europe/newsid_1859000/1859287.stm</Item>
        <Item Type="Title">Irish PM concedes abortion defeat</Item>
        <Item Type="Description">Irish Prime Minister Bertie Ahern announces that voters
            have narrowly rejected an amendment to the country's strict abortion laws.</Item>
      </Record>
    </Region>
    <Section Name="Asia-Pacific">
        ⋮ ⋮ ⋮
    </Section>
    <Section Name="Americas">
        ⋮ ⋮ ⋮
    </Section>
  </Section>
  <Section Name="Business">
    ⋮ ⋮ ⋮
</Wiccap>
```

Appendix C

List of Publications

The work in this thesis contributed a total of 9 international publications (8 published and 1 under review) as listed below. The publications that result indirectly from the work done in this thesis are marked with a '⋆'.

Book Chapters:

- Wee-Keong Ng, Zehua Liu, Zhao Li and Ee-Peng Lim. Personalized Web Information Extraction via Web Views. In Web Information Systems, edited by David Taniar and Wenny Rahayu, Idea Group Inc, 2004.

Journals and Articles:

- Zehua Liu, Wee Keong Ng, Ee-Peng Lim, and Feifei Li. Towards Building Logical Views of Websites. Data & Knowledge Engineering, 49(2):197–222, Elsevier, May 2004.
- Zehua Liu, Wee Keong Ng and Ee-Peng Lim. Automatic Extraction of Website Skeletons. Submitted for publication. Under review after revision. 2005.
- Zehua Liu, Hai Yu, Ee-Peng Lim, Ming Yin, Dion Hoe-Lian Goh, Yin-Leng Theng and Wee-Keong Ng. Java-based Digital Library Portal for Geography Education. Science of Computer Programming, 53(1):87–105, Elsevier, October, 2004. ⋆ [1]
- Zehua Liu, Wee Keong Ng and Ee-Peng Lim. Personalized Web Views for Multilingual Web Sources. IEEE Internet Computing, 8(4):16–22, IEEE CS, Jul 2004. ⋆ [2]

[1] The information extraction techniques developed in this thesis have been used in this paper to implement remote spatial queries

[2] A brief description of how the work developed in this thesis is integrated into the WICCAP system architecture for Personalized Web Views and its application in multilingual environment.

Conferences:

- Zehua Liu, Wee Keong Ng and Ee-Peng Lim. An Automated Algorithm for Extracting Website Skeleton. In Proceedings of the 9th International Conference on Database Systems for Advanced Applications (DASFAA 2004), pp. 799-811, Jeju Island, Korea, March 2004.

- Zehua Liu, Feifei Li, and Wee Keong Ng. Wiccap Data Model: Mapping Physical Websites to Logical Views. In Proceedings of the 21st International Conference on Conceptual Modelling (ER 2002), pp. 120–134, Tempere, Finland, October 2002.

- Zehua Liu, Feifei Li, Wee Keong Ng, and Ee-Peng Lim. A Visual Tool for Building Logical Data Models of Websites. In Proceedings of Fourth ACM CIKM International Workshop on Web Information and Data Management (WIDM 2002), in conjunction with the Eleventh International Conference on Information and Knowledge Management (CIKM2002), pp. 92–95, LcLean, Virginia, USA, November 2002.

- Zehua Liu, Wee Keong Ng, Ee-Peng Lim, Yangfeng Huang, and Feifei Li. Unloading Unwanted Information: From Physical Websites to Personalized Web Views. In Proceedings of Sixth Asia Pacific Web Conference (APWeb 2004), pp. 111–121, Hangzhou, China, April 2004. $\star$ [3]

[3]This paper describes the WICCAP framework in general and how the work developed in this thesis is integrated.

References

[1] Brad Adelberg. NoDoSE – A Tool for Semi-Automatically Extracting Semi-Structured Data from Text Documents. In *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD 1998)*, pages 283–294, Seattle, Washington, USA, June 2-4 1998.

[2] Brad Adelberg and Matthew Denny. NoDoSE Version 2.0. In *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD 1999)*, pages 283–294, Philadelphia, Pennsylvania, USA, June 1-3 1999. ACM Press.

[3] Arvind Arasu and Hector Garcia-Molina. Extracting Structured Data from Web Pages. In *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD 2003)*, San Diego, California, USA, June 9–12 2003. ACM Press.

[4] Gustavo O. Arocena and Alberto O. Mendelzon. Weboql: Restructuring documents, databases, and webs. In *Proceedings of the Fourteenth International Conference on Data Engineering (ICDE 98)*, pages 24–33, Orlando, Florida, USA, February 23–27 1998. IEEE Computer Society.

[5] Naveen Ashish and Craig A. Knoblock. Semi-Automatic Wrapper Generation for Internet Information Sources. *ACM SIGMOD Record*, 26(4):8–15, December 1997.

[6] Paolo Atzeni, Giansalvatore Mecca, and Paolo Merialdo. To Weave the Web. In *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB 97)*, pages 206–215, Athens, Greece, August 25–29 1997.

[7] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual Web Information Extraction with Lixto. In *Proceedings of 27th International Conference on Very Large Data Bases (VLDB 2001)*, pages 119–128, Roma, Italy, September 11–14 2001.

[8] BBC Online News. http://news.bbc.co.uk/, 2002.

[9] Michael K. Bergman. BrightPlanet LLC. The Deep Web: Surfacing Hidden Value. http://www.completeplanet.com/Tutorials/DeepWeb/, 2000.

[10] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001.

[11] Vijay Boyapati, Kristie Chevrier, Avi Finkel, Natalie S. Glance, Tom Pierce, Robert Stockton, and Chip Whitmer. ChangeDetector^{TM}: A Site-Level Monitoring Tool for the WWW. In *Proceedings of the 12th International Conference on Database and Expert Systems Applications (DEXA 2001)*, pages 587–598, Munich, Germany, September 3-5 2001.

[12] Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Proceedings of the Seventh International World Wide Web Conference (WWW 7)*, pages 107–117, Brisbane, Australia, April 14–18 1998.

[13] David Buttler, Ling Liu, and Calton Pu. A Fully Automated Object Extraction System for the World Wide Web. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS 2001)*, pages 361–370, Phoenix, Arizona, USA, April 16–19 2001.

[14] Mary Elaine Califf and Raymond J. Mooney. Relational learning of pattern-match rules for information extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI 99)*, pages 328–334, Orlando, Florida, USA, July 18–22 1999. The AAAI Press / The MIT Press.

[15] R. G. G. Cattell and Tom Atwood, editors. *The Object Database Standard: ODMG-93: Release 1.2.* Morgan Kaufmann, March 1996.

[16] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom. The TSIMMIS project: Integration of Heterogeneous Information Sources. In *Proceedings of the 10th Anniversary Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, Japan, October 1994.

[17] Sudarshan S. Chawathe and Hector Garcia-Molina. Meaningful Change Detection in Structured Data. In *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD 1997)*, pages 26–37, Tucson, Arizona, USA, May 13–15 1997.

[18] Junghoo Cho and Alexandros Ntoulas. Effective Change Detection Using Sampling. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB 2002)*, pages 514–525, Hong Kong, China, August 20–23 2002.

[19] Jim Cowie and Wendy Lehnert. Information Extraction. *Communications of The ACM*, 39(1):80–91, 1996.

[20] Valter Crescenzi and Giansalvatore Mecca. Grammars Have Exceptions. *Information Systems*, 23(8):539–565, 1998.

[21] Valter Crescenzi and Giansalvatore Mecca. Automatic Information Extraction from Large Websites. *Journal of the ACM*, 51(5):731–779, September 2004.

[22] 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 (VLDB 2001)*, pages 109–118, Roma, Italy, September 11–14 2001.

[23] Line Eikvil. Information Extraction from World Wide Web - A Survey. Technical Report 945, Norwegian Computing Center, July 1999.

[24] David W. Embley, Douglas M. Campbell, Yuan Stephen Jiang, Stephen W. Liddle, Deryle W. Lonsdale, Yiu-Kai Ng, and Randy D. Smith. Conceptual-model-based data extraction from multiple-record Web pages. *Data & Knowledge Engineering*, 31(3):227–251, 1999.

[25] David W. Embley, Y. S. Jiang, and Yiu-Kai Ng. Record-Boundary Discovery in Web Documents. In *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD 1999)*, pages 467–478, Philadephia, Pennsylvania, USA, June 1–3 1999.

[26] Martin Ester, Hans-Peter Kriegel, and Matthias Schubert. Web Site Mining: A new way to spot Competitors, Customers and Suppliers in the World Wide Web. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD 2002)*, pages 249–258, Edmonton, Alberta, Canada, July 23–26 2002. ACM.

[27] Sergio Flesca, Filippo Furfaro, and Elio Masciari. Meaningful Change Detection on the Web. In *Proceedings of the 12th International Conference on Database and Expert Systems Applications (DEXA 2001)*, pages 22–31, Munich, Germany, September 3-5 2001.

[28] Dayne Freitag. Machine Learning for Information Extraction in Informal Domains. *Machine Learning*, 39(2/3):169–202, 2000.

[29] Robert Gaizauskas and Yorick Wilks. Information extraction: Beyond document retrieval. *Journal of Documentation*, 54(1):70–105, 1998.

[30] Georg Gottlob and Christoph Koch. Monadic Datalog and the Expressive Power of Languages for Web Information Extraction. In *Proceedings of 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2002)*, pages 17–28, Madison, Wisconsin, USA, June 3–5 2002. ACM.

[31] Stéphane Grumbach and Giansalvatore Mecca. In Search of the Lost Schema. In *Proceedings of the 7th International Conference on Database Theory (ICDT 99)*, volume 1540 of *Lecture Notes in Computer Science*, pages 314–331, Jerusalem, Israel, January 10–12 1999. Springer.

[32] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences – Computer Science and Computational Biology.* Cambridge University Press, New York, USA, 1997.

[33] Joachim Hammer, Héctor García-Molina, Junghoo Cho, Arturo Crespo, and Rohan Aranha. Extracting Semistructured Information from the Web. In *Proceedings of the Workshop on Management of Semistructured Data*, pages 18–25, Tucson, Arizona, USA, May 16 1997.

[34] Joachim Hammer, Héctor García-Molina, Svetlozar Nestorov, Ramana Yerneni, Marcus Breunig, and Vasilis Vassalos. Template-based Wrappers in the TSIMMIS System. In *Proceedings of 1997 ACM SIGMOD International Conference on Management of Data*, pages 532–535, Tucson, Arizona, May 1997.

[35] Chun-Nan Hsu and Ming-Tzung Dung. Generating Finite-Sate Transducers for semistructured Data Extraction From the Web. *Information Systems*, 23(8):521–538, 1998.

[36] Gerald Huck, Peter Fankhauser, Karl Aberer, and Erich J. Neuhold. Jedi: Extracting and synthesizing information from the web. In *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems (CoopIS 98)*, pages 32–43, New York City, New York, USA, August 20–22 1998. IEEE Computer Society.

[37] Scott B. Huffman. Learning information extraction patterns from examples. In Stefan Wermter, Ellen Riloff, and Gabriele Scheler, editors, *Proceedings of the IJCAI'95 Workshop on Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, volume 1040 of *Lecture Notes in Computer Science*, pages 246–260. Springer, 1996.

[38] Hung-Yu Kao, Shian-Hua Lin, Jan-Ming Ho, and Ming-Syan Chen. Entropy-based link analysis for mining web informative structures. In *Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management (CIKM 2002)*, pages 574–581, McLean, VA, USA, November 4–9 2002. ACM.

[39] Gerald Keller, Brian Warrack, and Henry Bartel. *Statistics for Management and Economics*. Duxbury Press, 1994.

[40] Jon M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM*, 46(5):604–632, 1999.

[41] Stefan Kuhlins and Ross Tredwell. Toolkits for Generating Wrappers – A Survey of Software Toolkits for Automated Data Extraction from Websites. In *Proceedings of Net.ObjectsDays 2002*, Erfurt, Germany, October 7 – 10 2002.

[42] Nicholas Kushmerick. Wrapper Induction: Efficiency and Expressiveness. *Artificial Intelligence*, 118(1–2):15–68, 2000.

[43] Nicholas Kushmerick. Wrapper verification. *World Wide Web Journal*, 3(2):79–94, 2000.

[44] Alberto H.F. Laender, Berthier A. Ribeiro-Neto, and Altigran S. da Silva. DEByE–Data Extraction By Example. *Data & Knowledge Engineering*, 40(2):121–154, 2002.

[45] Alberto H.F. Laender, Berthier A. Ribeiro-Neto, Altigran S. da Silva, and Juliana S. Teixeira. A Brief Survey of Web Data Extraction Tools. *ACM SIGMOD Record*, 31(2):84–93, 2002.

[46] Kristina Lerman and Steve Minton. Learning the Common Structure of Data. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000)*, pages 609–614, Austin, Texas, USA, July 30 – August 3 2000. AAAI Press / The MIT Press.

[47] Feifei Li, Zehua Liu, Yangfeng Huang, and Wee Keong Ng. An Information Concierge for the Web. In *Proceedings of the First International Workshop on Internet Bots: Systems and Applications (INBOSA 2001), in conjunction with the 12th International Conference on Database and Expert System Applications (DEXA 2001)*, pages 672–676, Munich, Germany, September 3-7 2001.

[48] Wen-Syan Li, Necip Fazil Ayan, Okan Kolak, Quoc Vu, Hajime Takano, and Hisashi Shimamura. Constructing Multi-Granular and Topic-Focused Web Site Maps. In *Proceedings of the Tenth International World Wide Web Conference (WWW 10)*, pages 343–354, Hong Kong, China, May 1–5 2001. ACM.

[49] Seung Jin Lim and Yiu-Kai Ng. An Automated Change Detection Algorithm for HTML Documents Based on Semantic Hierarchies. In *Proceedings of the 17th International Conference on Data Engineering (ICDE 2001)*, pages 303–312, Heidelberg, Germany, April 2-6 2001.

[50] Ling Liu, Calton Pu, and Wei Han. XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. In *Proceedings of the 16th International Conference on Data Engineering (ICDE 2000)*, pages 611–621, San Diego, California, USA, February 28 – March 3 2000.

[51] Ling Liu, Calton Pu, and Wei Han. An XML-Enabled Data Extraction Toolkit for Web Sources. *Information Systems*, 26(8):563–583, 2001.

[52] Ling Liu, Calton Pu, and Wei Tang. WebCQ: Detecting and Delivering Information Changes on the Web. In *Proceedings of the 2000 ACM CIKM International Conference on Information and Knowledge Management (CIKM 2000)*, pages 512–519, Washington, DC, USA, November 6–11 2000. ACM.

[53] Zehua Liu, Feifei Li, and Wee Keong Ng. Wiccap Data Model: Mapping Physical Websites to Logical Views. In *Proceedings of the 21st International Conference on Conceptual Modelling (ER 2002)*, pages 120–134, Tempere, Finland, October 7-11 2002.

[54] Zehua Liu, Feifei Li, Wee Keong Ng, and Ee-Peng Lim. A Visual Tool for Building Logical Data Models of Websites. In *Proceedings of Fourth ACM CIKM International Workshop on Web Information and Data Management (WIDM 2002), in conjunction with the Eleventh International Conference on Information and Knowledge Management (CIKM2002)*, pages 92–95, LcLean, Virginia, USA, November 8 2002.

[55] Zehua Liu, Wee Keong Ng, and Ee-Peng Lim. Personalized Web Views for Multilingual Web Sources. *IEEE Internet Computing*, 8(4):16–22, Jul 2004.

[56] Zehua Liu, Wee Keong Ng, Ee-Peng Lim, Yangfeng Huang, and Feifei Li. Unloading Unwanted Information: From Physical Websites to Personalized Web Views. In *Proceedings of Sixth Asia Pacific Web Conference (APWeb 2004)*, pages 111–121, Hangzhou, China, April 14–17 2004.

[57] Zehua Liu, Wee Keong Ng, Ee-Peng Lim, and Feifei Li. Towards Building Logical Views of Websites. *Data & Knowledge Engineering*, 49(2):197–222, 2004.

[58] Giansalvatore Mecca and Paolo Atzeni. Cut and Paste. *Journal of Computer and System Sciences*, 58(3):453–482, 1999.

[59] George A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[60] Ion Muslea. Extraction Patterns for Information Extraction Tasks: A Survey. In *Proceedings of AAAI-99 Workshop on Machine Learning for Information Extraction*, pages 1–6, Orlando, Florida, USA, July 19 1999.

[61] Ion Muslea, Steve Minton, and Craig Knoblock. Selective Sampling with Redundant Views. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000)*, pages 621–626, Austin, Texas, USA, July 30 – August 3 2000. AAAI Press / The MIT Press.

[62] Ion Muslea, Steve Minton, and Craig Knoblock. Hierarchical Wrapper Induction for Semistructured Information Sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001.

[63] Sandeep Pandey, Kedar Dhamdhere, and Christopher Olston. WIC: A General-Purpose Algorithm for Monitoring Web Information Sources. In *Proceedings of 30th International Conference on Very Large Data Bases (VLDB 2004)*, pages 360–371, Toronto, Canada, August 31 – September 3 2004.

[64] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object Exchange across Heterogeneous Information Sources. In *Proceedings of the Eleventh International Conference on Data Engineering (ICDE 95)*, pages 251–260, Taipei, Taiwan, March 1995.

[65] Ma Qiang, Shinya Miyazaki, and Katsumi Tanaka. WebSCAN: Discovering and Notifying Important Changes of Web Sites. In *Proceedings of the 12th International Conference on Database and Expert Systems Applications (DEXA 2001)*, pages 587–598, Munich, Germany, September 3-5 2001.

[66] J. Ross Quinlan. Learning Logical Definitions from Relations. *Machine Learning*, 5(3):239–266, 1990.

[67] Ellen Riloff. Automatically constructing a dictionary for information extraction tasks. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI 93)*, pages 811–816, Washington, DC, USA, July 11–15 1993. The AAAI Press/The MIT Press.

[68] Arnaud Sahuguet and Fabien Azavant. Building intelligent Web applications using lightweight wrappers. *Data & Knowledge Engineering*, 36(3):283–316, 2001.

[69] Hichan Snoussi, Laurent Magnin, and Jian-Yun Nie. Heterogeneous Web Data Extraction using Ontology. In *Proceedings of the Third International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS 2001)*, Montreal, Canada, May 2001.

[70] Stephen Soderland. Learning to extract text-based information from the world wide web. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD 97)*, pages 251–254, Newport Beach, California, USA, August 14–17 1997. AAAI Press.

[71] Stephen Soderland. Learning Information Extraction Rules from Semi-Structured and Free Text. *Machine Learning*, 34(1–3):233–272, 1999.

[72] Stephen Soderland, David Fisher, Jonathan Aseltine, and Wendy G. Lehnert. Crystal: Inducing a conceptual dictionary. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 1314–1321, Montréal, Québec, Canada, August 20–25 1995. Morgan Kaufmann.

[73] XML Schema. http://www.w3.org/XML/Schema/, 2003.