Brno University of Technology
Faculty of Information Technology

Information Extraction from HTML Documents Based on Logical Document Structure

by

Radek Burget
M.Sc., Brno University of Technology, 2001

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Supervisor: Jaroslav Zendulka, associate professor
Submitted on: August 27, 2004
State doctoral exam passed on: June 18, 2003
This thesis is available at the library of the Faculty of
Information Technology of the Brno University of Technology

Contents

Abstract

The World Wide Web presents the largest Internet source of information from a broad range of areas. The web documents are mostly written in the Hypertext Markup Language (HTML) that doesn't contain any means for semantic description of the content and thus the contained information cannot be processed directly. Current approaches for the information extraction from HTML are mostly based on wrappers that identify the desired data in the document according to some previously specified properties of the HTML code. The wrappers are limited to a narrow set of documents and they are very sensitive to any changes in the document formatting.

In this thesis, we propose a novel approach to information extraction that is based on modeling the visual appearance of the document. We show that there exist some general rules for the visual presentation of the data in documents and we define formal models of the visual information contained in a document. Furthermore, we propose the way of modeling the logical structure of an HTML document based on the visual information. Finally, we propose methods for using the logical structure model for the information extraction task based on tree matching algorithms. The advantage of this approach is certain independence on the underlying HTML code and better resistance to changes in the documents.

Preface

In 2001, when I finished my master studies at the Faculty of Electrical Engineering and Computer Science at the Brno University of Technology, I was thinking about enrolling the PhD. program. There were multiple topics available but one of them attracted me more than the other ones: "Methods of knowledge discovery in the WWW". Since I was fascinated by everything related to the web, I started working on this topic under the supervision of Jaroslav Zendulka. Very quickly I realized how broad this area is and I decided to focus on processing the web content, which is always the first step of the data mining process.

During the first year, I have been trying to orientate myself in the topic. In addition to reading great amounts of available papers, I attended the EDBT 2002 Summer School on distributed databases on the Internet. In 2002, I also spent three months at a study stay at the University of Valladolid in Spain, which has been also very valuable for me.

This thesis is the result of the last three years' work. In my first papers concerning this topic [8, 9], I formulated the features of current information extraction approaches that in my opinion caused the major problems of the existing methods and I proposed a more abstract look at the HTML documents. During further work, I proposed some suitable models of the documents and the methods of using them for information extraction [10]. The last year I spent on a formal specification of the proposed models and methods [11].

Organization of the Thesis

This thesis is organized as follows. Chapter 1 contains a short introduction to the area of processing the data accessible through the World Wide Web and explains basic concepts of the information extraction from the web documents.

Chapter 2 gives a brief overview of the current World Wide Web technology. Basic concepts of the most important languages are explained and the main technological aspects and forthcoming technologies are discussed.

In Chapter 3, the state of the art in the information extraction from HTML documents and related areas is summarized.

In Chapter 4 we summarize major problems of the current information extraction approaches as the motivation for this thesis and we formulate the goals of this thesis.

Chapter 5 is the theoretical core of the thesis: it contains the formal models of the visual appearance and the logical structure of HTML documents and introduces a novel information extraction method based on these models.

In chapter 6, we describe an experimental system that implements the proposed information extraction method and that has been used for testing the method on real data.

Chapter 7 summarizes the experimental results.

And finally in Chapter 8, we conclude the thesis. We discuss possible uses of the information extraction and of the logical document structure modeling. We summarize the major contributions of the thesis and we propose possible improvements of the method and directions of further investigation in this area.

The appendices contain the XML specifications of the information extraction tasks that have been used for testing the method and formal definitions of the XML formats used within the experimental system.

Acknowledgements

I would like to express my sincere thanks to people who have helped me writing this thesis: Jaroslav Zendulka, my supervisor, for his valuable comments and organizational support, Alexander Meduna for his great help with the formal specification issues. Many thanks also deserve people who supported me while writing this thesis: my parents, my brother and my girlfriend Jana.

Chapter 1

Introduction

The World Wide Web presents currently the largest Internet source of information from a broad range of areas. One of the main reasons of this great expansion is the absence of any strict rules for the information presentation and the relative simplicity of the used technology. The orientation to documents and the HTML language that is mainly used for creating the documents, gives the authors enough freedom for presenting any kind of data with minimal effort and the new technologies such as Cascading Style Sheets (CSS) allow to achieve the desired quality of presentation. Together with the hypertext nature of the documents, these properties make the World Wide Web a distributed and dynamic source of information.

On the other hand, the loose form of the data presentation brings also some drawbacks. With the increasing number of available documents a problem arises, how to efficiently access and utilize all the data they contain. Due to the above properties of the web, related information is often presented at different web sites and in diverse forms. Accessing this data by browsing the documents manually is a time-consuming and complicated task. For this reason, it is desirable to automatically process the documents by a computer. As a first step, there exists an effort to provide the users with centralized views of related data from various sources in the World Wide Web such as the services for comparing the prices of goods in on-line shops. The next step is presented by the data mining techniques that have been developed for the database branch.

For all these tasks, it is necessary to access the data that are contained in the documents. This is however not a trivial problem. As mentioned above, the state-of-the-art web consists mainly of the documents written in the Hypertext Markup Language (HTML) [54]. This language is suitable for the definition of the presentation aspect of the documents but it lacks any means for the definition of the content semantics. The information contained in the documents can be therefore very hardly interpreted and processed by a computer.

Possible answer to this problem is the proposal of *semantic web* [6] that is based on different technology and has quite different nature. While the classical web can be viewed as a distributed document repository, the semantic web has many characteristics of an object database [34]. Although this technology is very promising and it is being developed rapidly, it doesn't solve processing of the great amount of documents that

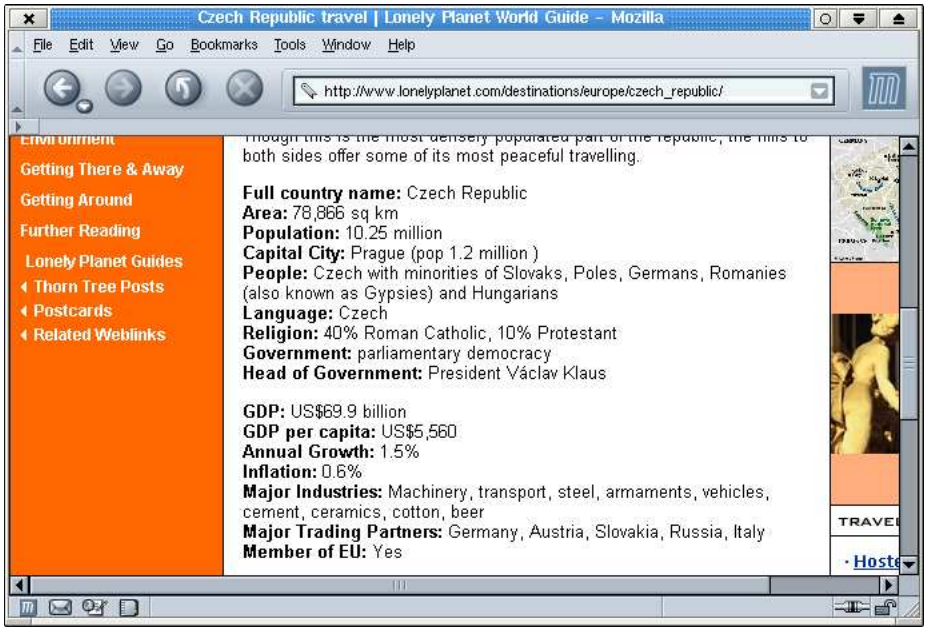


Figure 1.1: An Information Extraction Task

are already available in the "legacy" web. Moreover, simultaneously with the semantic web, the legacy web is still growing and developing. In contrast to the semantic web, the evolution of the technologies has however quite different direction. Currently, the most important issue in the development of the classical web technologies is the flexibility of the web design and effective management of the web content. For the information providers, it is not the aim to allow better automatic processing of the documents on the web and in some cases it is even undesirable. All these facts together with the great amount of information that is virtually available make the automatic HTML document processing an interesting and challenging area of investigation.

1.1 Information Extraction

Our aim in the HTML document processing is to identify a particular information that is explicitly contained in the text of the document and to store it in a structured form; e.g. to a database table or a XML document. This process is usually called *information extraction* from the HTML documents [21, 26, 38].

As an example, let's imagine a set of pages containing information about various countries as shown in the Figure 1.1. The task of information extraction in this case is to identify the values of country name, area, population etc. and to store them in a structured way. The result of processing such a set of documents could be a database table containing the appropriate values for each country.

There are two basic approaches to the information extraction. The first and most common one assumes that it is specified in advance, what data are to be extracted from the documents. The specification for the country information task mentioned above can have for example following format

$$COUNTRY ::=< name, area, population, capital >$$

i.e. for each country we want to extract its name, geographical area, population and the name of the capital. This type of task is often called the *slot filling task*. In contrary, the other approach is to analyze the documents and extract all the available data. This approach requires a set of documents to be compared in order to distinguish the relevant data from the remaining text [18]. It is assumed that the relevant data differs among the documents whereas the other text remains static.

The process of extracting information from the World Wide Web can be split to following phases:

1. Localization of relevant documents
2. Identification of the data in the documents
3. Data storage

The first one is the information retrieval phase where the documents containing the appropriate data must be localized in the World Wide Web. Since each document is identified by its URI, the result of this phase is a list of URIs of the documents to be processed. In the World Wide Web, there are several types of services available for locating the documents, from the Web directories that provide categorized lists of URLs to the automatic search engines such as Google [1] that are based on indexing terms in the documents. The hypertext nature of HTML documents brings another problem: the presented data can be split to several HTML documents that contain links to each other. Then, this set of HTML documents forms a logical entity that is usually called a *logical document* [60]. Since the way in which the information is split to individual documents cannot be predicted, for extracting the information all the physical documents must be processed. As a result, the URIs of all the documents must be discovered.

The identification of the data in documents is the key phase of the information extraction process. During this phase, a model of the logical document is created and the desired data values are identified by certain *extraction rules*. The character of these rules depends on the model of the document and the information extraction method. Alternatively, the data values may be identified based on a statistical model of the document (e.g. the Hidden Markov Models) instead of explicit extraction rules.

The last phase of the process is the data storage. This phase depends completely on the application.

Although the information extraction area had established long before the World Wide Web come into being and its application to the web is almost as old as the web itself, due to the enormous extent and variability of the web it still presents a challenging problem that hasn't been satisfactorily resolved yet. In this thesis, we focus

[1] `http://www.google.com`

on the parts of the information extraction process that are not sufficiently explored yet. In the first phase it is the problem of the logical document discovery. We discuss the existing approaches and we propose certain improvements of current techniques. The main focus of our work is on the data identification phase. We analyze the problems of current methods and we propose a novel approach to this problem based on modeling the visual aspect of the documents and their logical structure.

Chapter 2

The World Wide Web Technology

The World Wide Web is built on a concept of *hypertext*. This notion has been introduced in 1965 by T. H. Nelson [51] and refers to a text that is not constrained to be linear; it contains links to other texts. Hypertext that is not constrained to text only is called *Hypermedia*. It can include graphics, video and sound, for example.

The World Wide Web is document-oriented. Each document is identified by a Uniform Resource Identifier (URI). A notion of Unified Resource Locator (URL) is often used in the same meaning. A document can contain any type of data (text, hypertext, graphics, etc.). For identifying the type of content, the MIME (Multipurpose Internet Mail Extension) standard is used.

The web is based on client-server architecture. The documents are stored on a *web server* that is accessible through the Internet. The client accesses the server and downloads the desired documents using the Hypertext Transfer Protocol (HTTP) [24]. In most cases, the client is represented by a *web browser* that displays the document on the screen.

Most of the documents in the World Wide Web are created using the Hypertext Markup Language (HTML). Additionally, more technologies have been developed for improving the presentation capabilities and to make the maintenance of the content more efficient. There exist also many standard formats for presenting the non-textual data such as images and multimedia files.

2.1 Hypertext Markup Language

Hypertext Markup Language (HTML) [54] is a basic language for publishing hypertext on the web. It is based upon SGML (Standard Generalized Markup Language, ISO 8879). An HTML document is basically a text file enriched by standardized *tags*. HTML contains tags for defining the structure of the document (headings, paragraphs), basic page objects (lists, tables, images), text formatting (font and color selection, special effects) and, of course, hypertext links to other documents. Most of the tags are *pair tags* that consist of an *opening tag* written `<tagname>` and *closing tag* written

`</tagname>`. For example, a portion of text can be written in bold by following code:

```
Normal text <b>bold text</b> normal text.
```

that will be printed out as:

Normal text **bold text** normal text.

Some of the tags are unpaired, for example a line break can be inserted using just a `<br>` tag. Some of the tags allow or require specifying additional *attributes* that define the meaning of the tag more precisely. For example, when inserting an image to the document, an attribute `src` must be specified that identifies the file where the image is stored: `<img src="image.jpg">`.

The SGML language that has been used as a basis for creating HTML is very powerful and flexible. However, this great flexibility brings certain drawback such as, for example, complicated implementation. For this reason, a simplified derivation of SGML has been created that is called XML (eXtensible markup language). The most important simplification is that only the pair tags are allowed and the tags must be properly nested (they may not overlap). For hypertext publishing, a XML variant of HTML has been created that is called XHTML (eXtensible hypertext markup language). In comparison to HTML, many tags have been omitted and replaced by other means (mainly the Cascading Style Sheets that are described below). The unpaired tags have been replaced by pair tags even in the cases that the pair tag gives no sense – e.g. the above mentioned line break must be written as `<br></br>` which can be according to the XML specification also written as `<br/>`. Currently, XHTML is an upcoming standard in web publishing. All the mentioned web standards are being maintained by the World Wide Web Consortium[1].

In this thesis, we focus on hypertext documents written in the HTML language optionally in conjunction with the Cascading Style Sheets. However, all the techniques and methods proposed below apply to the XHTML language as well.

2.2 Cascading Style Sheets

Cascading Style Sheets (CSS) is a simple mechanism for adding style (e.g. fonts, colors, spacing) to Web documents. It allows separating the visual style of the page from the HTML code that facilitates the creation and management of the web pages and allows better flexibility. For XHTML, the Cascading Style Sheets presents a unique mechanism for specifying the visual style of the page.

The style sheets consist of a set of *rules* that define the style for particular (X)HTML tags. For example, a rule can look as follows:

```
h1 { font-weight: bold; color: blue; }
```

This rule says that the headings marked with the `<h1>` tag will be printed in bold and blue color. It consists of a *selector* that determines on which tags this rule should be

[1]http://www.w3.org

applied and a *declaration*. The declaration consists of a name of the CSS *property* (in our case `font-weight:` and `color:`) and its *value* (`bold` and `blue`). The values of the individual properties are inherited as the tags are nested in the HTML code – the values of properties that are not specified in our example are inherited for example from the declaration of the `<body>` that encloses the whole content of an HTML document.

Each element in the HTML code denoted by some tag can be assigned a named class using the `class` attribute (e.g. `<p class="heading">`) and/or assigned an identifier using the `id` attribute that is unique for the whole document. The CSS rule selectors can be based not only on the tag names (as we can see in the example) but also on the element classes and identifiers.

The style sheets are incorporated to the HTML code by one of following ways:

- The style sheet is placed in a separate file that is available through the web and it is referenced in the HTML document header.
- The style sheet is directly inserted into the header of the HTML document.
- The declaration is specified without a selector, directly for a particular tag in the document as a value of its `style` attribute. This is called *inline* style definition. For example, the tag `<p style="font-weight: bold;">` starts a paragraph written in bold.

The use of CSS has grown significantly in the last few years, which has been allowed by sufficient support of CSS by the modern web browsers. It is now practically unimaginable to create a modern, well-designed web presentation without the use of CSS.

2.3 Dynamic Web Content

The modern web technologies allow creating documents that don't only contain a static text but their contents can be generated dynamically upon a client request according to various factors. Basically, the dynamic content can be generated two in ways:

- **On the server side**. There are various technologies that allow generating the HTML code in the time of the client request (e.g. CGI, PHP, ASP, JSP, etc.). In this case, no support for dynamic content is required on the client side, the client always obtains a complete HTML document that can be, however, different on each request.
- **In the client browser**. The HTML document that is sent to the client contains routines written in a scripting language (in most cases, JavaScript language is used) that are interpreted by the client browser. The scripting language must be supported by the browser.

From the document processing point of view, the former case doesn't present any particular problem because the client always obtains a complete document that can be processed directly. The use of JavaScript or similar technology presents a serious problem not only for automatic web content processing tools but also for various alternative browsers such as text-oriented browsers or voice readers for blind people. For this reason, JavaScript should be used as an additional feature only and its support shouldn't be automatically expected by the authors of documents. For the same reason, we don't solve the problems related to the dynamic web content generated on the client side in this thesis.

An additional problem of the dynamically generated pages on the server side is that the generation procedure requires certain data to be supplied by the client when requesting the page from the server. Typically, this is the case of the pages that are generated according to the values filled-in to an interactive form that is available on another page. Without filling some fields of the form – i.e. without providing a particular data along with the page request, the dynamic page is not accessible. The great amount of such pages available through the web is often called a *hidden web* because its content is not easily accessible to any automatic web content processing tools including the search engine indexing robots. Although the hidden web presents a considerable problem for any automatic processing of the web content, it forms a separate and quite large area of investigation that is quite distant from the information extraction as such. Therefore, in this thesis, we only mention this problem without discussing it in greater detail.

2.4 Web Services

Web services present a way of using the Internet for application to application communication. They provide a standardized way of specifying the capabilities and programmatic interfaces of the services that are available over the Internet and the communication protocols that allow using the services by other services or applications. This possibilities originate a distributed service architecture where the services can use each other for performing a particular task and the inter-service communication and the results have a standardized, computer-processable format.

For example, the Google search engine provides a standard web interface where the user uses a web browser for downloading an HTML page with a query form; he fills in the search query and posts the query to the server. The server returns the list of search results, again as an HTML document. The results in this form are intended to be displayed only; it would be very complicated to further process the query results by some application. For this reason, Google provides an alternative interface in a form of a web service. This service receives the query and returns the appropriate results using the standardized web service protocols based on XML so that the results may be further processed.

The roles in web services and the relations among the protocols and the roles are shown in the figure 2.1. The basic roles are the *service provider* and the *service requester* (client). The requester sends a request and the provider sends back the results both using the Simple Object Access Protocol (SOAP). In order to allow the automatic localization of appropriate web services over the Internet, the web service proposal includes a role of a *service broker* that maintains a registry of available web services. Each service publishes information about its purpose and interface to the service broker using the Web Services Description Language (WSDL). The service registry format and the way of querying are is described by the UDDI (Universal Description, Discovery, and Integration) standard.

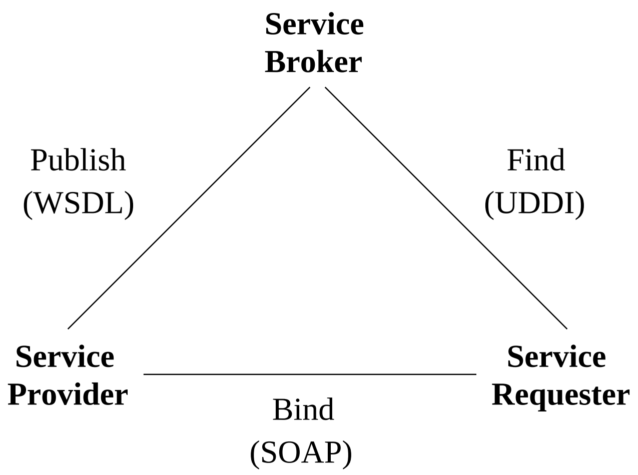


Figure 2.1: The relations among protocols and roles in web services

2.5 Semantic Web

According to [6], the Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation. The semantic web proposal is based on two based technologies: XML and the Resource Description Framework (RDF). RDF allows encoding basic triplets being rather like the subject, verb and object of an elementary sentence. Slightly more formally, it consists of a *subject*, a *predicate* (also known as a *property* of the triplet) and an *object* each of which is defined by its URI. Alternatively, the object may be specified by a literal. This simple architecture allows anyone to define new concepts and verbs by defining new URI for it and to “say anything about anything” by defining new RDF triplets. Each set of RDF definitions creates a *RDF graph* where the nodes correspond to the concepts (subjects and objects) and the edges correspond to the predicates. Figure 2.2 shows an example of a RDF graph that specifies an address of a person. The ovals correspond to the concepts defined by their URIs (a staff member and an address), the rectangles represent the literals.

The last important component of the semantic web is formed by ontologies. Basically, an ontology is a document or file that formally defines the relations among terms. For specifying the ontologies in the semantic web environment, the Web Ontology Language (OWL) has been proposed.

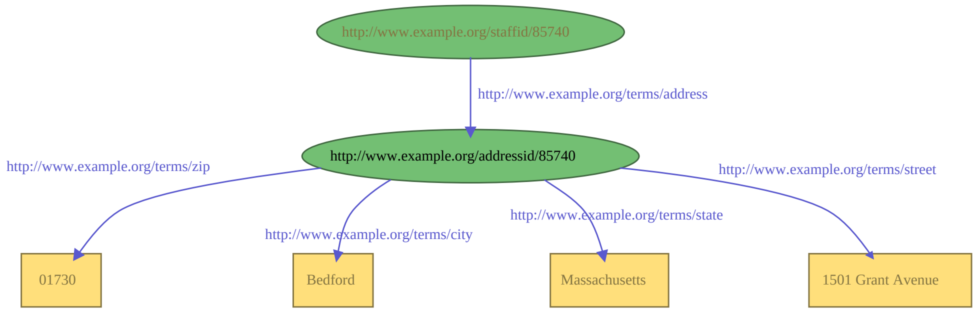


Figure 2.2: An example of an RDF graph (taken from [44])

2.6 Current Information Extraction Alternatives

As stated in the introduction, it is not a trivial task to process the information contained in the HTML documents mainly because the documents don't contain any formal definition of the contained data semantics. The web services and the semantic web are often regarded as possible alternatives, that can be used for giving the content the semantics and allow better processing of the content.

The web services are suitable for applications, where the resulting data are highly structured and have a fixed format that is specified in the service definition. The resulting data are expected to be a product of some process that can be optionally influenced by some input parameters. The drawback of web services is that they require the installation and maintenance of an indispensable amount of quite complicated software on the server that can, in addition, significantly increase the requirements to the server performance. Creating a web service is not a trivial task neither. It requires special tools and creating the software pieces that implement the web service process itself. When comparing this quite complicated technology with the simplicity of the traditional web, in most cases the possible benefits are not worth all the effort.

The semantic web, in comparison to the web services, is much more flexible and its implementation on the server side consists mainly of creating the appropriate XML documents that contain the RDF data. On the other hand, in order to be usable, it requires the implementation of *software agents* that would join together the distributed pieces of information in order to infer the desired results. Unfortunately, the entire semantic web is still in more or less theoretical stage.

The last important issue is not technological. Very often, the information is presented on the web with the only purpose of attracting the users that read the advertisement and bring profit to the service providers. It is not the aim of the information providers to allow better automatic processing of the presented data.

When considering all these facts, the service providers are not motivated enough to adopt the existing or proposed semantic technologies and even the benefits of these technologies for the end-users are questionable. The result is obvious – while the number of available HTML documents grows exponentially, the semantic technologies are practically not used. In this situation, we must accept current state of the web as it is and to state, that extracting information from the unsophisticated HTML documents is currently the only way how to access the information available in the web.

Chapter 3

State of the Art

Our approach to information extraction combines several areas of endeavour that have been intensively investigated recently. Firstly, it is the area of information extraction from HTML documents that mainly focuses on the problems related to *wrapper generation* and *wrapper induction*. And secondly, it is the area of advanced modeling of the structure of documents and the discovery of semantic structures in the documents.

As mentioned in section 1.1, the hypertext capability of HTML allows to create larger information entities consisting of multiple documents linked together that are usually called *logical documents*. A complex approach to information extraction therefore includes the analysis of the logical documents.

First, we give an overview of current approaches to individual, "physical" HTML documents. Next, we discuss the area of modeling the logical organization of documents. This area presents an alternative look to the HTML document modeling and it is applicable to information extraction. And finally, current techniques of the logical document discovery are discussed.

3.1 Information Extraction from HTML Documents

The information extraction has been largely investigated in the plain text context long before the World Wide Web emerged. It has been used for processing the electronic mail messages, network news archives etc. [50]. Many information extraction techniques for various types of electronic messages have been proposed within the frame of the Message Understanding Conferences (MUC) [31] that were in progress from 1987 to 1998. For each MUC, participating groups have been given sample messages and instructions on the type of information to be extracted, and have developed a system to process such messages. These systems have been then evaluated for the conference.

The World Wide Web and the HTML language bring a new look to information extraction. In contrast to the plain text messages, HTML allows to define the visual presentation of the content. This possibility is often used for making the documents clearer and easily understandable. The data in HTML documents is presented in a more or less regular and structured fashion. For this reason, the HTML documents are often regarded as a *semistructured information resource* [38]. The reader is not forced

Capital Cities

France – *Paris*
Japan – *Tokyo*

```
...
<h1>Capital Cities</h1>
<b>France</b> - <i>Paris</i>
<b>Japan</b> - <i>Tokyo</i>
...
```

Figure 3.1: Example of a simple document and its code

to read the whole document. On the contrary, according to many studies, the readers only *scan* the document looking for some interesting parts instead of reading it word by word [47]. Due to this behaviour of the web users, writing text for the World Wide Web has become a specific area for which the term *web copywriting* is frequently used. One of the major requirements to the text on the web is that the organization of the page is clear and the user can easily find the part of the document containing the desired information. For this purpose, the documents contain a system of navigational cues that have mostly visual character. During the years of the World Wide Web development, these techniques of data presentation have been brought almost to perfection so that reading the documents using the web browser becomes relatively efficient.

From the point of view of the automated document processing, the situation is different. HTML doesn't contain sufficient means for the machine-readable semantic description of the document content. The techniques for natural language processing that have been used for the information extraction from plain text are not applicable, because HTML documents usually don't contain many whole sentences or blocks of continuous text. On the other hand the HTML tags inserted to the text of the document provide additional information that can be used for identifying the data. In the figure 3.1, there is an example of a simple document and the relevant part of its HTML code. Let's imagine that we want to extract the names of countries and their capitals from this document. When looking at the HTML code, we can notice that each name of a country is surrounded by the `<b>` and `</b>` tags and accordingly, each name of a capital is surrounded by the `<i>` and `</i>` tags. Thus for extracting the desired information from this document, a simple procedure can be created that reads the document code, detects these tags and stores the text between each pair of the tags. Such a procedure is called a *wrapper*. Apparently, the given example is quite trivial. For more complex documents, more sophisticated wrappers have to be designed.

For the data in the HTML documents, a database terminology is usually used in the information extraction context. We assume that the documents contain one or more *data records* where each record consists of some number of *data fields*. Usually, we admit that some records are incomplete; i.e. that the values of some fields are missing in the document. For example, figure 3.1 shows a document containing two records where each record has two fields: the name of the country and the name of its capital city.

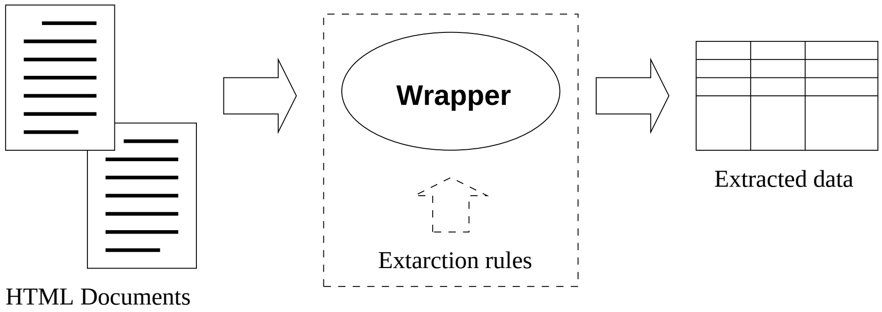


Figure 3.2: Information extraction using a wrapper

Capital Cities

France – *Paris*
Japan – *Tokyo*

```
...
<i>Capital Cities</i>
<b>France</b> - <i>Paris</i>
<b>Japan</b> - <i>Tokyo</i>
...
```

Figure 3.3: Ambiguous fields in a document

3.1.1 Wrappers

According to Kushmerick [38], a *wrapper* is a procedure that provides the extraction of particular data in the HTML document as illustrated in the figure 3.2. For the identification of the particular data in the document, the wrapper uses either a set of *extraction rules* that define the way of the identification of each individual data field or a model of the document that is used for the decision of which part of the document corresponds to the particular data value (for example, the Hidden Markov Models are used in some methods). With *wrapper construction*, we mean the process of formulating the extraction rules or the model of the document for a particular information extraction task.

Kushmerick in [38] defines six classes of wrappers with increasing expressiveness that differ in the way, how the extraction rules are defined. The simplest wrapper class is called LR (left-right). In this class, for each data field to be extracted, one extraction rule is defined. Each rule is a pair of strings that delimit the field in the document code from the left and from the right. Let's consider again an example of a simple document shown in the figure 3.1. An LR wrapper for this task can be defined as a set of two rules:

$$W = \{[< b >, < /b >], [< i >, < /i >]\}$$

where the first rule identifies the values of *country* and the second one identifies the values of *capital*.

This way of defining the rules is apparently not usable for all the tasks; let's consider a slightly modified example in the figure 3.3. When the above LR wrapper were invoked on this document, the caption would be incorrectly identified as a city name. As a solution, more complex wrapper classes have been defined:

- **HLRT** (head-left-right-tail). Two additional delimiters are used; the *head* delimiter is used to skip potentially confusing text in document heading and the *tail* is used to skip potentially confusing text in the bottom of the document.
- **OCLR** (open-close-left-right). The *open* and *close* delimiters identify each record in the document.
- **HOCLRT** (head-open-close-left-right-tail). An analogical combination of the above two classes.
- **N-LR** and **N-HLRT**. The modification of the LR and HLRT classes for handling nested tabular data in documents.

According to Kushmerick, the six wrapper classes were able to handle about 70% of the real web documents. It is obvious that the wrapper works properly for a limited set of documents that correspond to previously defined extraction rules. In literature, such a set of documents is commonly called a *document class*. In most cases, the document class consists of documents of the same topic generated automatically from a back end database by an identical procedure or at least created by the same author. Moreover, the wrapper works until the data presentation changes. As results from a simple comparison of the document in the figures 3.1 and 3.3, even a minor change in the document design can cause the wrapper to stop working properly. Due to the distributed and dynamic nature of the Web, this state cannot be predicted and since no additional information about the extracted data is provided, it is not trivial to detect the malfunction of the wrapper automatically.

When using wrappers for integrating information from many sources, for each source a wrapper or wrappers must be created and when some conditions change, the wrappers must be modified appropriately. From this point of view, the method how the wrappers are being constructed is important. The most obvious method is writing the wrappers *by hand*; i.e. by analyzing a set of documents to be processed and determining the delimiting strings. This method is very time-consuming and error-prone; unfortunately, it is the most used method currently. The companies employ people that work on coding new wrappers and maintaining the old ones. Since this approach presents a serious scalability problem, many approaches have been developed for an automatic inference of wrappers.

Most methods for the automatic wrapper construction are based on *wrapper induction*. This approach is based on machine learning algorithms and the wrapper construction proceeds in following phases:

1. A supervisor provides a set of training samples (i.e. labeled HTML pages)
2. A machine learning algorithm is used to learn the extraction rules
3. A wrapper is generated based on the extraction rules
4. The wrapper is used on the target documents

Capital Cities

Country	Capital
France	Paris
Japan	Tokyo

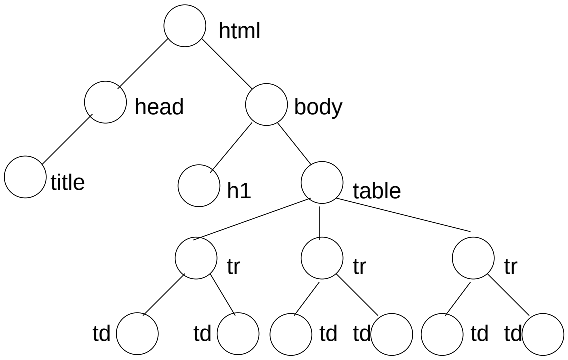


Figure 3.4: An HTML code tree

For the inference of the extraction rules, a model of the document text and the embedded tags must be created. The character of this model depends on the machine learning algorithm used. In following sections, we give an overview of the used models of the document and the existing methods for the wrapper induction.

3.1.2 Document Code Modeling

The most straightforward model is to represent the document code simply as a string of characters. In this representation, the text of the document is not explicitly distinguished from the embedded tags. When processing the documents represented this way, usually the extraction rules based on delimiting substrings [57] or regular expressions [3] are used. For example, the words that end with a colon can introduce an important data value. Such phrases can be found using the regular expression `[A-Za-z0-9_]+[:]`.

For using the machine learning algorithms, it appears that is more suitable to use a different representation where the basic unit is a word instead of a character. The document is represented as a sequence of words [15, 26, 35, 49]. Each word can be assigned various attributes based some of their orthographical or lexical properties. The embedded HTML tags can be either omitted or used for inferring additional attributes of the individual words [26, 49] (e.g. the text is in a caption). A special case of such a model is used by [35]. In this model, on the contrary, the HTML tags are regarded as the symbols of an alphabet Σ; any text string between each pair of subsequent tags is represented with a reserved symbol x.

Most common is a hierarchical model of the HTML code that represents the nesting of the tags in the document [12, 17, 21, 22, 23, 37]. Figure 3.4 shows an example of a simple document and the corresponding tree of the HTML tags. In order to make it possible to create such a model for an HTML, it is necessary to pre-process the document so that we obtain so called *well-formed document* [12] where all opening tags have the corresponding closing tag and they are properly nested. The XHTML documents are always well-formed. The text content of the document is then contained in the leaf nodes of the tree or it is not included in the model at all. It is the advantage of the hierarchical model that it describes the relations among the tags in addition to the observed properties of individual words and tags.

3.1.3 Wrapper Induction Approaches

Current methods of the wrapper induction are based on the knowledge from various areas of the research work. Many approaches are based on grammatical or automata inference [18, 25, 35, 37, 38], other approaches use the relational machine learning algorithms [17, 21, 26, 57]. Quite different approach is presented by the methods based on conceptual modeling [22, 23]. Note that this is a coarse classification only and the different approaches influence each other.

As mentioned in section 3.1.1, the wrapper induction approaches require a set of labeled examples of the documents that are used for inferring the extraction rules. According to the artificial intelligence terminology, we call this set of examples a *training set* and the process of inferring the extraction rules the wrapper *training*.

For evaluating the performance of the individual information extraction approaches, there are two commonly used metrics: the *precision* P and *recall* R [37]. They are defined as follows:

$$P = \frac{c}{i} \tag{3.1}$$

$$R = \frac{c}{n} \tag{3.2}$$

where c is the number of correctly extracted records, i is the total number of extracted records and n is the real number of the records in the document.

Methods Based on the Grammatical Inference

The grammatical inference is a well-known and long studied (early studies in the 60's) problem and its application to information extraction is therefore supported by a large theoretical foundation. The problem is following: We have a finite alphabet Σ and a language $L \subseteq \Sigma^*$ (usually, regular or context-free languages are discussed in this context). Given a set S^+ of the sentences over Σ that belong to L and a (potentially empty) set S^- of the sentences over Σ that do not belong to L we want to infer a grammar G that generates the language L.

The basic idea behind using grammatical inference for information extraction is that generating a wrapper for a set of HTML documents corresponds to a problem of inferring a grammar for the HTML code of the pages and finally using the inferred grammar for the extraction of the data fields. This idea is however not directly applicable to the document processing. The main obstacle is that there are only positive examples available in the web; i.e. the available documents. As it follows from Gold's work [30], regular nor context-free grammars cannot be correctly identified from the positive samples only. This problem can be basically solved by limiting the language class to a subclass of regular languages that is identifiable in the limit (e.g. k-reversible languages) or by changing the computational model (artificial negative samples or supplying an additional information).

One of the approaches to the information extraction is presented in [25]. For locating a particular data field in the document, a combination of a Bayes classifier and grammatical inference is used. The document is modeled as a sequence of words. The

Bayes classifier processes parts of the document determined by a floating window of a fixed length of n words. To each position of the window, a probability is assigned that the particular part of the text matches to the particular data field (e.g. the name of a person). The problem of this method is in determining the exact boundaries of the data field (the window of a fixed size). Moreover, the classification doesn't consider the word order; the Bayes classifier only works with the occurrence of individual words. These problems are solved by the grammatical inference. The words contained in the document are converted to abstract symbols from an alphabet Σ using their orthographical properties. Thus the alphabet Σ contains symbols of the type `word-lower+dr` (an abbreviation "Dr." written arbitrarily in upper-case or lower-case letters), `capitalized-p t` (any word beginning with capital letter) etc. The training set is used for both training the Bayes classifier and for inferring a finite automaton, where each terminating state is assigned a probability that the accepted string corresponds to the particular data field. When the string is not accepted at all, it is assigned a small probability. The result is then a product of the probabilities from the Bayes classifier and from the automaton.

Another approach to the application of grammatical inference is presented by Kosala [37]. This method is based on tree languages. The document is modeled according to its HTML code as a tree where the nodes contain the name of the HTML tag or a text string. Instead of a set of strings over Σ we obtain a set of trees over a new alphabet V where each tree corresponds to a particular document from the training set. In the sample trees, the data field to be extracted is replaced by a special symbol x. Then, the aim is to infer a deterministic tree automaton that accepts the trees in which the desired data value is replaced by x. When using this automaton for information extraction, we replace subsequently the nodes of the document tree by x. Once the resulting tree is accepted by the automaton, the extraction result is the original string that has been replaced by x.

A different way of the grammatical inference application is presented by [35]. This work is based on stochastic context-free grammars. The input alphabet is formed by the HTML tags and an extra symbol `text` that represents any non-empty text string between a pair of tags. During the grammatical inference process, the complexity of the grammar is evaluated and the simplest grammar is chosen. The non-terminals of the inferred grammar correspond to basic parts of the document. For more exact localization of the data fields, regular expressions are used that represents the domain-specific knowledge.

A completely new view of the problem is presented by [18]. The presented approach deals with the problem of schema discovery – given a set of HTML documents we are looking for a common schema of their content and the extraction rules based on the discovered schema. The schema discovery is based on comparing the documents – the parts that are present in all the documents are considered as static content whereas the changing parts correspond to the data values.

Hidden Markov Models

A Hidden Markov Model (HMM) is a finite state automaton with stochastic state transitions and symbol emissions. The automaton models a probabilistic generative processes whereby a sequence of symbols is produced by starting at a designated start state, transitioning to a new state, emitting a new symbol – and so on until a designated final state is reached.

The application of HMMs to information extraction is based on a hypothetical assumption that the text of a document has been produced by a stochastic process and we attempt to find a Markov model of this process. The states of the model are associated to the tokens to be extracted. The model transition and emission probabilities are learned form training data. The information extraction is performed by determining the sequence of states that was most likely to have generated the entire document and extracting the symbols that were associated with designated target states.

This approach is used for example by [27]. For each field to be extracted a separate HMM is used that consists of two types of states – the target states that produce the tokens to be extracted and background states. Each of the HMMs models the entire document so that no pre-processing is needed and the entire text of the documents from the training set is used to train the transition and emission probabilities.

Relational Learning Approaches

General principle of these techniques is similar to the above. Again, we assume that there exists a class of documents with similar properties and that there is a training set of the documents from this class available. In the training set, we describe some properties for each data field to be extracted using logical predicates. Then, we use relational learning algorithms for inducing general rules that identify the data fields in documents.

Freitag [26] assigns each word in the documents certain attributes based on the properties of the given portion of the text such as word length, character type (letters, digits) or orthography and adds some additional attributes that describe the relation between the word and the surrounding HTML tags (e.g. the word forms part of a heading or the word forms a table row). Each data field to be extracted is then described by logical predicates based on these attributes and using an algorithm SRV (based on the FOIL algorithm [53]) a general rule is inferred that identifies the data field in the document. DiPasquo [21] extends this approach by modeling the hierarchical structure of HTML tags in the document, which allows describing the relations among the HTML tags more exactly.

Soderland [57] uses an approach that is based on the methods for natural language processing. Each word of the document a *semantic class* is assigned (e.g. Time, Day, Weather condition) using predefined *concept definitions.* Using the learning algorithms AQ and CN2 [16] a general description of each data field is inferred.

3.1.4 Alternative Wrapper Construction Approaches

Wrapper induction is not the only method for automatic wrapper construction. Following techniques are based on direct analysis of the HTML code of the particular documents to be processed. The aim of these techniques is to avoid the training phase and eliminate the requirement of the training set of document. On the other hand, these techniques are usually based on empirical heuristics and it is often hard to specify exactly for which document the method is suitable. Furthermore, some predefined domain-specific knowledge is often required and in some cases (e.g. [33]) the method is language dependent.

HTML-related Heuristics

These techniques are based on specific empirical heuristics related to the HTML language generally or to some generally accepted ways of its usage.

Ashish [3] locates some important words that introduce an important information in the document (e.g. *Geography*, *Transportation*, etc.) – so called *tokens*. The tokens are identified based on the properties of the text and surrounding HTML tags and all the possible occurrences are firmly defined by regular expressions. For example, the text between the `<b>` and `</b>` tags, words in headings, text that ends with a colon, etc. Each token indicates the start of a section of the document. Next, the hierarchical structure of sections is built by comparing the font size and the indentation of the text that begins each section. The proposed extraction tool contains a graphical user interface for an interactive adjustment of the tokens and the hierarchical structure. Finally, the wrapper is generated using the YACC generator.

Another approach is used by [12, 22]. This approach assumes that there can be found a unified separator of the data records in the document. The document is modeled as a tree of tags and based on various heuristics, a general structure is discovered that is used as a record separator. As the next step, a data field separator is located similar way. The heuristics are based on the statistical analysis of the text in potential sections, repeating patterns, etc. Furthermore, a predefined knowledge about the meaning of some HTML tags is used. Similar approach is used in [43]. The proposed MDR algorithm attempts to locate the regions of the document tag tree that potentially contain data records. In these regions, one or more data records can be identified.

Conceptual Modeling

Conceptual modeling approach is more common in the area of the information extraction from plain text documents; however, it can be used for HTML documents too. For example Embley et al. [22, 23] proposes a method where as the first step, an ontological model of the extracted information is created and based on this model, corresponding data records are discovered in the document. It is possible to combine this approach with the HTML code analysis described above. The main difference is that the structure of the information is not inferred from the document but it is known in advance.

3.1.5 Computer-aided Manual Wrapper Construction

This category is formed by special tools that generate wrappers in collaboration with a human expert. These tools usually provide a graphical user interface that allows the wrapper creator to analyze the documents to be processed and to easily design a wrapper.

The *DoNoSe* tool [1] works mainly with plain text documents. The tool allows hierarchical decomposition of the contained data and mapping selected regions of the text to components of the data model. *LiXto* [5] is a fully visual interactive system for the generation of wrappers based a declarative language for the definition of HTML/XML wrappers. Both tools provide a graphical user interface that allow the user with no programming experience to produce the appropriate wrappers.

3.2 Cascading Style Sheets from the IE Perspective

With the new technologies being introduced to WWW, some critical disadvantages of the wrapper approach appear. For example, according to the recommendations of the WWW Consortium the usage of Cascading Style Sheets (CSS) [7] has grown significantly in the last few years. This technology allows defining the visual layout and formatting of an HTML or XML document independently on its content. This property is particularly useful when the HTML documents are being generated dynamically (e.g. from a database) since it allows modifying the visual appearance of the pages without modifying the HTML generator. On the other hand, this significantly reduces the amount of information that can be used by a wrapper for identifying the information in HTML documents.

Figure 3.5 shows an example of a traditional HTML document formatting. It is obvious that all the names of the countries are denoted by the `<b>` and `</b>` tags. A wrapper for extracting countries from the document simply looks for these tags and extracts the inserted text.

```
...
<h1>Capitals</h1>
<b>France</b>
  - <i>Paris</i>
<b>Japan</b>
  - <i>Tokyo</i>
...
```

Figure 3.5: Document formatting using HTML tags

One of the possible variants of the same document code written using CSS is shown in the figure 3.6. The above method for defining the wrapper fails in this case because all the elements are denoted by the same `<span>` tag. Moreover, there are several ways how to incorporate CSS to the HTML code (in our example, we can see the classes defined by the `class` attribute and inline styles defined by the `style` attribute of the HTML tags).

```
...
<span class="heading">Capitals</span>
<span class="country">France</span>
  - <span style="font-style: italic;">Paris</span>
<span class="country">Japan</span>
  - <span style="font-style: italic;">Tokyo</span>
...
```

Figure 3.6: Document formatting using CSS

As we can see, all the HTML tags have been replaced with a single `<span>` tag that is used for specifying the CSS class of the individual parts of the text. Moreover, as mentioned in section 2.2, the style definitions can be incorporated different ways to the HTML code. In any case, the result is that the "semantic" HTML tags such as headings, emphasis, etc. may be completely removed from HTML and replaced by CSS definitions. This change significantly complicates or even makes unusable most of the wrapper induction methods mentioned above.

From this point of view, it is not reliable to construct wrappers that rely directly on the HTML code. HTML and CSS are only the means for creating documents and they can be used various ways. The fixed point in this variable world is the *final presentation* of the document that has been usually carefully designed by experts from various branches and that must be delivered to the reader in unchanged form regarding especially the *visual appearance* and the *structure* of the presented document.

For this reason, instead of modeling the HTML code directly, some more sophisticated models need to be proposed that describe the documents from the perspective of its final presentation. These models should describe the organization and the visual appearance of the documents as it is expected to be perceived by a human reader.

3.3 Advanced Document Modeling

3.3.1 Logical versus Physical Documents

The World Wide Web consists mainly of HTML documents that may reference each other using hypertext links. In this thesis we will call these documents *physical documents* because each of them corresponds to a physical file stored on a WWW server. However, the hypertext links allow splitting a complex information to multiple physical documents where each of them contains a specific part of the information and these individual parts are interconnected by the hypertext links. This way of presentation is very frequent in the World Wide Web. We will call such a set of physical documents that form a complete information entity a *logical document*. An example of a logical document is given in the figure 3.7. The arrows represent links among the physical

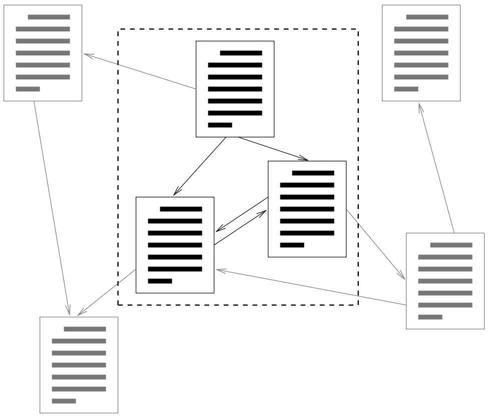


Figure 3.7: Logical document

documents. The three physical documents in the dashed box form a logical document. As we can see, any of the documents that form the logical document can contain links to other external documents that do not belong to the logical documents. Since it is not specified in the documents, which of the links are external, the discovery of logical documents in the web is not a trivial task and it requires further analysis of the documents and the links.

3.3.2 Logical Document Discovery

The task of the logical document discovery consists of locating all the physical HTML documents that form a logical entity called logical documents. The input is the URI of the *main page* (sometimes also called the *top page* or the *index page*), which is intended by the author of the document to be an entry point to the logical document and it is usually directly accessed by the users[1]. The output is the list of the URIs of all the HTML documents that form the logical document.

The primary source of information for discovering the related physical documents are the HTML links. An HTML document that forms part of a logical document, except the main page, must be referenced by at least one other HTML document in the same logical document. The process of logical discovery is therefore quite straightforward:

- We find the URIs of all the documents referenced in the main page
- We select all URIs that point to the documents that belong to the logical document
- We repeat the process recursively for each selected document

The major problem that has to be solved is in the second step: How to distinguish the documents that belong to the logical document from the remaining ones? There exist two different types of information that can be used for resolving this task:

[1]The URI of the main page is usually publicly available, in contrast to the URIs of the remaining documents

- *Document classification.* We assume that the individual physical documents that form the logical document are more similar to each other than the remaining referenced documents. The similarity of documents is usually computed using the methods based on the term frequency in the document such as the $tf \cdot idf$ method [55].
- *Document layout analysis.* We analyze and compare the layout of the documents as mentioned for example in [42]
- *Link topology analysis.* In general, the topology of the links among a set of HTML documents can be represented as a directed graph. By the analysis of this graph using specific heuristics we can detect a subgraph with certain properties. This technique is also used for detecting so-called *communities* in the web [29]. Additionally, some limitations can be put to the format of the URIs (e.g. all documents must be placed on the same web server, etc.)

Usually, these types of information are used together. Tajima et al. [60] shows that most of the logical documents are organized into hierarchies. The approach is based on the assumption that the authors include some hypertext links to the documents that are intended to be used by the users as a standard way of getting to a particular document. These links form so-called *standard navigation paths.* There are, of course other links that either point outside of the logical document or have a lower importance such as links back to the main page. The proposed method of logical document discovery has two steps. First, the hierarchical structure is discovered by identifying paths intended by the authors of the documents to be the *standard navigation routes* from the main page to other pages. Then, the discovered hierarchy is divided into sub-hierarchies corresponding to the logical documents based on comparing the document similarity using the $tf \cdot idf$ method.

3.3.3 Logical Structure of Documents

The information extraction methods described in the section 3.1 have been based on a direct analysis of the code of HTML documents. We can say that these methods work with the *physical* realization of the documents. The bottleneck of this approach is too tight binding of the wrapper to the HTML code. The nature of HTML allows to achieve the desired document design by various ways that can be arbitrarily combined, which makes the wrappers limited to a narrow set of documents and a short time period. As an answer to these drawbacks, there have been several attempts to describe the documents from a *logical* point of view.

The *logical structure* of a document [2, 13, 58] is basically a model of a document that describes the relations among the logical sections of the document as for example sections, paragraphs, figures, captions, etc. There are generally two approaches to the logical document structure discovery:

- Visual document analysis – we analyze the visual aspect of the documents. This approach is applicable to any electronic document format such as PostScript, PDF, HTML, etc. For HTML documents, many approaches to the visual analysis have been published [14, 32, 36, 48, 62].

- Direct analysis of the hierarchy of tags in the document code – we assume that the nesting of the tags corresponds to the logical structure of the document. This approach is applicable to the markup languages only. It is more reliable from the complexity point of view, but it is not usable in all cases (for example when the Cascading Style Sheets are used in certain way) since the hierarchy of tags needn't necessarily correspond to the logical structure.

In this thesis, with the notion of *logical document structure* we understand the logical structure of any document, either the physical or the logical one depending on context. The notion of logical document structure has been introduced by Summers [58, 59] in the context of processing the PDF, PostScript and scanned-in documents and it is defined as a hierarchy of segments of the document, each of which corresponds to a visually distinguished semantic component of the document. Other authors use the notions of *document structure tree* [36] or *document map* [63] in similar sense.

For some time, it has been assumed that in case of HTML documents, there is no need for modeling the logical structure because it is directly present in the document in the form of the HTML tags. However, a closer analysis shows that there is only a very loose binding between the HTML tag tree and the logical structure of an HTML document. The reason is that the HTML provides both structural and presentational capabilities that can be arbitrarily combined. Furthermore, the effort of the document authors aims to the resulting visual presentation rather than the logically correct HTML code so many tags are often misused. Thus creating the logical structure of an HTML document is not trivial neither and it requires more detailed analysis of the document.

In almost all works published on the logical document structure analysis, the resulting model is a tree, where the nodes correspond to individual logical parts of the document. This model is based on the observation that each HTML document consists of elements that specify information carrying objects at different levels of abstraction through object nesting. The logical structure can be viewed as that the objects of a higher level of abstraction are described by objects of finer levels of abstraction [14]. Such a hierarchical conception of a document organization seems to be natural to the document authors as well as the readers. This fact has been observed by many authors, for example [1, 4, 14, 20, 49, 58, 60] without any more detailed reasoning. We believe that the main reasons of the hierarchical document organization are:

- *It is efficient.* Hierarchical organization of a document allows better orientation in the text. It is possible to find the section that deals with a particular topic without having to read the whole document.

- *It is feasible.* A standard document is linear – it has a beginning and the end. The hierarchical organization can be easily reached by using various levels of headings and labels. The organization is then apparent to the reader, especially when the table of contents is included. It is not feasible to achieve some more complex organization such as a general graph without confusing the reader. The situation is different in case of the logical hypertext documents; this problem is discussed separately in section 5.7.

- *It is natural.* The hierarchical organization of a structured text has been widely used in technical and popular articles and books. People are used to it. This is actually a consequence of the above two reasons.

For the above reasons, the authors of structured documents mostly prefer the hierarchical organization and the readers automatically expect it. Therefore, a tree appears to be sufficient for modeling all kinds of documents.

3.3.4 Visual Analysis of HTML Documents

There are several approaches to creating the model of the logical structure for HTML documents. They differ in the granularity of the resulting model that depends on its intended application.

The work of Carchiolo et al. [13] deals with the discovery of the logical schema of a web site that contains multiple documents. For this purpose, basic logical sections are localized in each document such as logical heading, logical footer and logical data where the semantic of the page is mainly placed. The proposed approach is based on a bottom-up HTML code analysis. First, collections of similar code patterns are localized in the document. As the second step, each section is assigned a meaning (e.g. logical header) based on the semantics of the HTML tags (e.g. the `<form>` tag denotes an interactive section) or on some information retrieval techniques (e.g. the header section is the collection that refers to the text in the title of the document or to the URI of the page). Similarly, [62] discovers semantic structures in HTML documents. This approach is based on the observation that in most web pages, layout styles of subtitles or data records of the same category are consistent and there are apparent boundaries between different categories. First, the visual similarity of the HTML content object is measured. Then, a pattern detection algorithm is used to detect frequent patterns of visual similarity. Finally, a hierarchical representation of the document is built. The method described in [48] is based on similar principle. A key observation of this method is that semantically related items in HTML documents exhibit spatial locality. Again, a tree of HTML tags is built and similar patterns of HTML tags are discovered. Finally, a tree of the discovered structures is built.

While the above methods discover the logical structure of the document to the level of basic semantic blocks, the work of Chung et al. [14] is more oriented to information extraction. Based on the visual analysis, it attempts locate data fields in the documents and store them in an XML representation. It is assumed that the documents being processed pertain to a particular, relatively narrow ontology. Furthermore, certain domain knowledge provided by the user is necessary in the form of *topic concepts* and optional *concept constraints*. Each concept is described by a set of *concept instances* that specify the text patterns and keywords as they might occur in topic specific HTML documents. By contrast, the topic constraints describe how concepts as information carrying object can be structured. As the first step, a majority schema of the document is inferred in the form of a document type definition (DTD). Next, the data fields corresponding to the individual concepts are discovered.

The last described approach is quite different. In [32] a method for the web content structure discovery is presented that is based on modeling the resulting page layout and locating basic objects in the page through projections and two basic operations - block dividing and merging. The projection allows to detect visual separators that divide the page into smaller blocks and again, adjacent block can be merged, if they are visually similar. This dividing and merging process continues recursively until the layout structure of the whole page is constructed.

Chapter 4

Motivation and Goals of the Thesis

Currently, wrappers are mainly used for obtaining the data from various web sites in order to create a service that provides a centralized view of the data from certain domain available in the WWW. Such a service allows a user to effectively use the data available in the web with no need of locating the appropriate web sites, browsing the documents and locating the appropriate data in the documents. Such a service is most frequently offered for the shopping domain – several services for comparing prices of goods in on-line shops are available (e.g. AmongPrices[1] or Compare Online[2]). Another good example is the financial domain – services for comparing stock quotes, exchange rates etc.

Although many techniques for automatic wrapper construction have been proposed, the most used approach is still the manual or semi-automatic wrapper construction. The cause of this situation are following major problems that appear in different extent in the above mentioned wrapper induction approaches:

- *The necessity of wrapper training.* A sufficiently large set of annotated training data is required. The training set preparation is time-consuming; moreover, in some cases no training data is available at all (e.g. when there is only one instance of the page available). A partial solution of this problem is bootstrapping [52].
- *Brittleness.* When some change occurs in the documents being processed, the wrapper can stop working properly. The detection of this situation is not a trivial problem [40]. Moreover, new training data must be prepared and the wrapper has to be re-induced.
- *Low precision.* For practical setting of the wrapper, it is necessary that the produced data is reliable – i.e. the values of *precision* and *recall* are close to 100%. The precision of current methods varies between 30 and 80% [15, 21, 26, 37] depending on the used method and the processed documents.

[1] `http://www.amongprices.com/`
[2] `http://www.compare-online.co.uk`

There are several causes of these problems. The most important one is too tight binding of the produced wrapper to the HTML code, which makes the wrapper very sensitive to any minor irregularity in the HTML code. All the above mentioned wrapper induction methods are based on an assumption that there exists some direct correspondence between the HTML code and its informational content. However, the relevance of this assumption is also quite questionable. As stated in the introduction, HTML allows defining a visual appearance of a document, i.e. a presentation and formatting of the contained text. The relation between this formatting and the semantics of the document content is not defined anywhere. Wrappers based on some assumptions regarding this relation that have been defined based on previous analysis of some documents are therefore based on an information whose relevance is not guaranteed anywhere and it can be (and usually it is) limited to a certain set of document and a certain (unpredictable) time period. Moreover, the document can contain various irregularities and special features so that the rules induced for a document needn't to be completely applicable to another document event if it belongs to the same class of documents.

As a solution of the above mentioned disadvantages of the wrapper approach, we propose developing a new method that would fulfill the following requirements:

1. *The documents are analyzed in time of extraction.* The method shouldn't be based on any features of the document or the set of documents that have been discovered in the past. Using information that was relevant at some time in the past can lead to incorrect results because the document may have been changed in the meantime.

2. *Only the features of the logical document being processed are used for information extraction.* Using some knowledge about the features of other documents that are considered to be "similar" to the currently processed one can lead to incorrect results and imprecision.

3. *The method must be independent on the physical realization of the document.* It must be based on the final document appearance that must respect certain rules rather than the underlying HTML/CSS technology that can be chosen arbitrarily. This includes the situations when the presented information is split to several physical documents. Always the whole logical document should be analyzed.

These requirements solve the problems of the necessity of the training set of documents by analyzing each logical document individually. Moreover, this feature should improve the precision of extraction because the method is not based on the extraction rules inferred for other documents. And lastly, the independence on the physical realization of the logical document should significantly reduce the brittleness of inferred wrappers. The goals of our work can be summarized to following points:

1. To find a suitable model that describes the documents on sufficient level of abstraction and to define this model in a formal way.

2. To propose a new method of information extraction based on the defined model that fulfills the above requirements and resolves the above mentioned problems.

3. To evaluate the proposed method experimentally on real WWW documents.

In following sections, we present the results of our work that have been achieved while fulfilling the above specified goals.

Chapter 5

Visual Information Modeling Approach to Information Extraction

As discussed in the previous chapter, most problems of the wrapper approach are caused by the fact that the wrappers interpret the HTML code too literally. This makes the wrapper very sensitive to any irregularity or a slight change of the document code. For removing this strict dependence on the HTML code, it is necessary to create a more general model that would describe the document abstracting from the nonessential implementation details.

The HTML tags embedded in the text of the document are an instrument for achieving certain visual presentation of the document contents. Since the documents in the World Wide Web are designed to be browsed by human users, it seems to be reasonable to create a model describing the resulting rendered document as it is expected to be displayed in the browser window. When designing a document, the authors use the HTML tags for achieving a particular design with following particular aims:

- To make the document good-looking from the aesthetic and typographical point of view.
- To encourage the user to read the document; i.e. to make the document visually "attractive".
- To make the document well arranged; i.e. to allow the user an effective orientation in the document with no need of reading each individual part of the document.

Although from the aesthetic and marketing point of view, the first two points are important, from the point of view of the automatic document processing the last point is the only important one. We are not interesting how attractive the document is for the user but we are interested in what information is the user given in order to be able to quickly understand the organization of the document. There exist several commonly accepted rules in the visual presentation of the documents that are used for giving the reader this information. These rules have been established during the evolution of typography long before first electronic documents appeared. The most common of them are following:

- The parts of the text that deal with different topics or that have a different purposes are visually separated; e.g. the individual articles in a newspaper, footnotes in a book, etc.
- Bold text, italics and underlining are used to stress the significance of a particular part of the text. In modern typography, different colors of the text are sometimes used for highlighting a particular word or a sentence.
- The larger font is used for typing a particular text, the more important this text is. E.g. the most important affairs in newspapers are announced with banner headlines whereas the less important notes are written with a small font size.
- The headings and labels that denote certain information are highlighted using some combination of the above means.

In case of the world wide web documents, these visual instruments are used even more intensively that in the traditional media. Since the Internet is usually used for quickly and effectively obtaining some information, the documents must provide great amount of visual cues that navigate the reader. Most often, these cues have the form of highlighted headings and labels that denote the meaning of each part of the document.

In this chapter, we propose an information extraction method that is based on modeling the visual information in the document that is intended to be used by the readers for quick navigation. First, we define the method for processing a single HTML document. Later, in section 5.7, we extend the method to logical documents formed by multiple HTML documents. As stated in the introduction, in this thesis we focus on the data identification phase of the information extraction process.

5.1 Proposed Approach Overview

The principle of the proposed approach is shown in the figure 5.1. In contrast to the traditional wrapper approach (figure 3.2), the information extraction process consists of multiple steps.

As a first step, we analyze the HTML document that contains the data to be extracted and we create a model of the visual information as it is expected to be presented in a standard web browser. This model consists of two separate components – the model of the page layout and the model of the visual features of the text.

As the next step, we transform these two components to an unified model of the document logical structure. This model describes the document content on significantly more abstract level. As defined in section 3.3.3, the logical structure only describes hierarchical relations among individual parts of the document content. When we represent the text content of the document as a text string, (omitting the embedded HTML tags), the resulting model of the logical structure is a tree where each node contains a substring of the text of the document and the edges represent relations between a superior

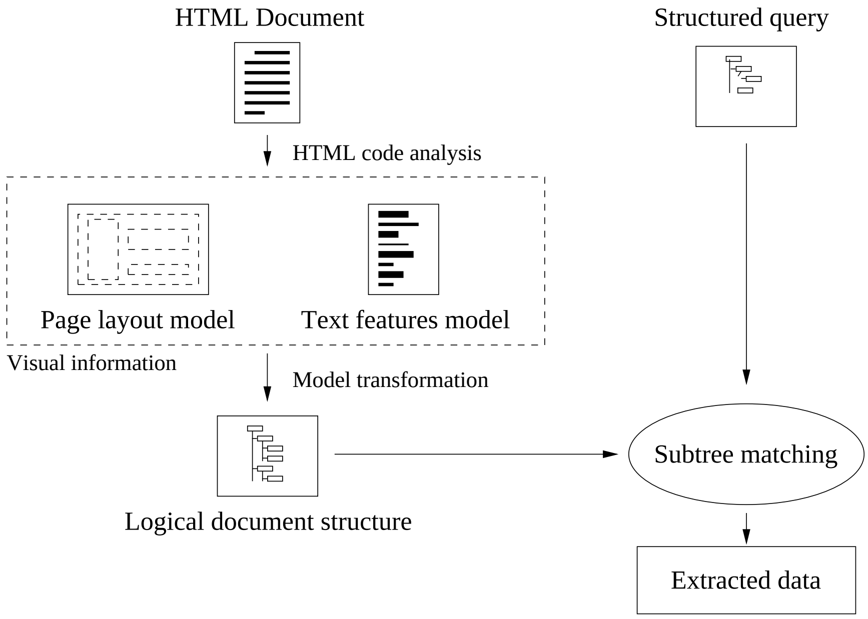


Figure 5.1: Visual modeling of an HTML document

or more general part of the text (e.g. a heading) and the inferior, more concrete part (e.g. the chapter contents or a data value). More detailed specification can be found in section 5.5.

After creating the logical structure model, the next step is the information extraction itself. Since the logical structure model is a tree of text strings, the information extraction task can be formulated as a problem of locating a particular tree node that contains the desired information. In our method, we propose defining the information extraction task as a template of a subtree of the logical document structure. After this, the information extraction task consists of locating all the subtrees of the logical structure model that correspond to the specified template. Each subtree found corresponds to a data record in the extracted data.

In following sections we give a detailed description of the individual steps.

5.2 Visual Information Modeling

As discussed in previous sections, the documents in the World Wide Web have more or less hierarchical organization so that the user can effectively locate the desired information in the document. This hierarchical organization is expressed by two basic means:

1. By splitting the document to several visually separated parts that can be arbitrarily nested; i.e. the *page layout*.

2. By providing a hierarchy of headings and labels of different level of abstraction

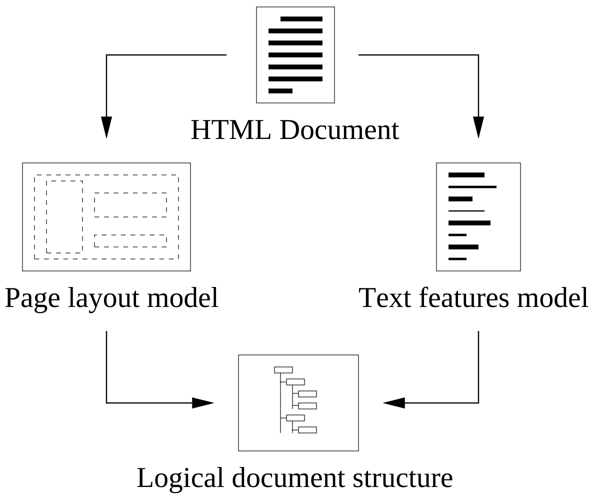


Figure 5.2: Visual modeling of an HTML document

that describe the contents of a part of the document or the meaning of a particular data presented in the document. This hierarchy is expressed by various *typographical attributes* of the text; e.g. the more important is the heading, the larger font size is used, etc.

In order to obtain a logical structure of a document we have to create the models of both the components of the visual information: the page layout and the typographical attributes and to transform these models to the model of the logical document structure as shown in Figure 5.2.

5.2.1 Modeling the Page Layout

In order to clearly distinguish different kinds of information in the document, the web pages are usually split to multiple areas. Figure 5.3 shows a typical example of a document split to several visual areas. We can notice three basic visual areas – the header on the top of the page, the left column with a navigation menu and the main part that carries the informational contents of the page. The main part is further split by a horizontal separator in two parts – the main contents and the footer. We can see that some areas are nested and thus there is a hierarchy of visual area present in the document.

Generally, the visual areas in a document can be visually expressed using various visual separators (horizontal rules, boxes, etc.) or by different visual properties; the most usual one is different background color. In HTML, it is not difficult to enumerate all the possible means that can be used for creating a visually separated are in a document. As follows from the HTML specification [54], the set of available means is quite limited. To be specific, such an area can be created using following HTML constructions:

Figure 5.3: Visual areas in a document

- *Document.* The whole document can be considered as a visual area that always forms the root node of the visual area hierarchy.
- *Tables and table cells.* Each cell of a table forms an area that can have its own visual attributes. The table forms an area that holds all the cells and optionally a table caption.
- *Lists and list items.* A list item itself forms a visually separated area, a list can contain multiple items.
- *Paragraphs.* Paragraphs are usually not used for creating separated areas but they can be interpreted that way when used together with CSS and the visual attributes of a paragraph are sufficiently different from the attributes of preceding and following text.
- *Generic areas.* HTML provides a generic area `<div>` that has no influence to the visual attributes itself. However, it is often used together with CSS that allows to specify the style and position of the area.
- *Frames.* HTML frames allow to combine multiple HTML documents in a page. Each of the documents forms an area.
- *Horizontal rule.* The `<hr>` element cannot be used for creating a standalone visual area but it allows to insert a horizontal line that splits an area in two parts.

A complete list of the related page objects with the appropriate HTML tags is given in Table 5.1.

Page object	**HTML tags**
Document	`<html>`
Table, table cell	`<table>` `<th>` `<td>`
List, list item	`<ul>` `<ol>` `<dl>` `<li>` `<dt>` `<dd>`
Paragraph	`<p>`
Generic area	`<div>`
Frames	`<frameset>` `<frame>`
Horizontal rule	`<hr>`

Table 5.1: Visual areas in HTML and corresponding tags

For the purpose of modeling the page layout we define the notion of *visual area* as any area in the document that is formed by one of the mentioned means independently on its visual attributes. For example, we consider each table cell a separate visual area independently on whether the surrounding table cells have different background color or whether they are separated by a bounding box. The visual areas in a document form a hierarchy where the root represents the whole document and the remaining nodes represent the visual areas in the document that can be possibly nested. For modeling the page layout, we assign each visual area an unique numeric identifier $v_i \in I$, where the whole document has $v_0 = 0$ and the remaining areas have $v_i = v_{i-1} + 1$. Then, the layout of the page can be modeled as a tree of area identifiers v_i as shown in the figure 5.4.

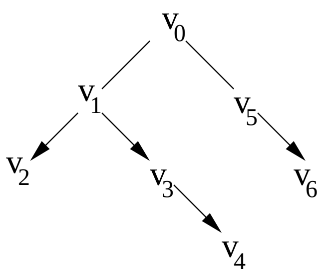


Figure 5.4: Example of the page layout model

Formally, the model of the page layout can be denoted as a graph:

$$M_l = (V_l, E_l) \tag{5.1}$$

where $V_l = \{0, 1, \ldots n - 1\}$ is the set of all area identifiers that form the nodes of the tree and $(v_i, v_j) \in E_l$ iff v_i and v_j are the visual area identifiers and the area identified by v_j is nested in the area identified by v_i.

This model doesn't contain any information about the visual attributes of the areas, it only represents the way in which the visual areas are nested. However, by assigning individual parts of the document text to the visual areas we can obtain the information about what parts of the text are related. The model thus represents the most important information the page layout gives to the reader as discussed above.

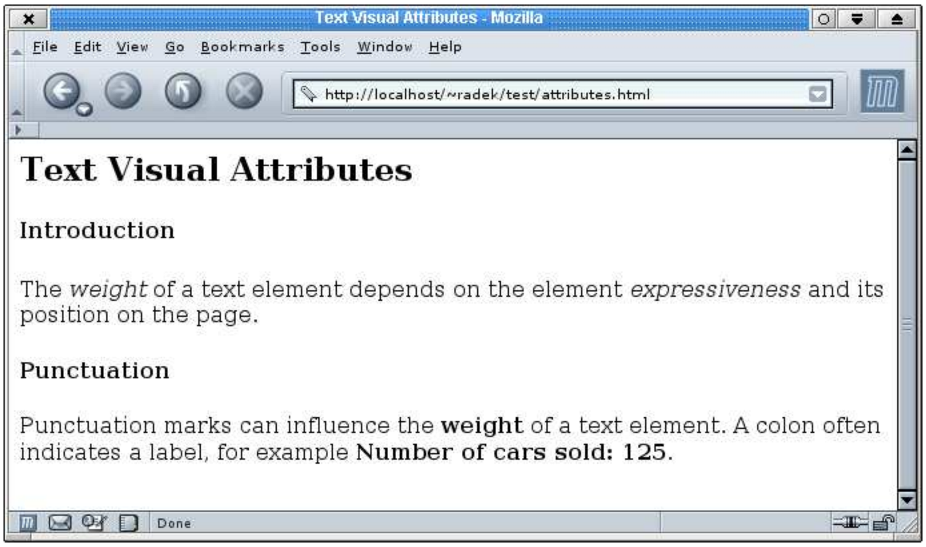


Figure 5.5: Visual attributes of the text

5.2.2 Representing the Text Features

Various visual attributes of certain portions of the text are often used for emphasizing the importance of the particular text portion or, on the contrary, to suppress its importance. Another common use of the visual attributes is to distinguish the headers and labels in a document from the remaining text. An illustration of this fact is given in the figure 5.5. In the text of the first section, the words "weight" and "markedness" are emphasized by using italics. Similarly, the word "weight" in the second section is emphasized by using the bold font. There are two levels of headings in the document that are distinguished by the font size. The first-level heading gives the title of the document whereas the two second-level headings are used for the titles of the sections. Furthermore, there is a label "Number of cars sold" that introduces the value of "125". The function of the label is indicated by the trailing colon.

For the purpose of automatic document processing, we need to create a model of the documents describing the mentioned features of the text. This model has to describe these features on an abstract level. For example, we want to describe the fact, that the word "weight" is emphasized in both occurrences. The used mean, i.e. if the emphasizing is achieved using the bold font or italics is not significant from our point of view. The same effect can be achieved for example by typing this word in red color. Similarly, the font size used for the word "Introduction" is not important, we want to describe that this heading has a lower level than the main document heading but it is at higher level that the remaining text.

An HTML document consists of the text content and the embedded HTML tags. Some of the tags can specify the typographical attributes of the text. Let's denote T_{html} a set of all possible HTML tags and S an infinite set of all possible text strings between each pair of subsequent tags in a document. Then, an HTML document D

can be represented as a string of the form

$$D = T_1 s_1 T_2 s_2 T_3 s_3 \ldots T_n s_n T_{n+1} \tag{5.2}$$

where $s_i \in S$ is a text string with the length $|s_i| > 0$ that doesn't contain any embedded HTML tags and n, $n \geq 0$ is the number of such strings in the document. T_j, $1 \leq j \leq n+1$ is a string of HTML tags of the form

$$T_j = t_{j,1}\, t_{j,2}\, t_{j,3}\, \ldots\, t_{j,m_j} \tag{5.3}$$

where $t_{j,k} \in T_{html}$ and $|T_j| \geq 1$.

Since the visual attributes of the text can only be modified by the HTML tags, each text string s_i has constant values of all the attributes. Let's define the notion of *text element* as a text string with visual attributes. Each text element $e_i \in E$, where $E = S \times I \times I \times I$. As usual, we write e_i as

$$e_i = (s_i, v_i, x_i, w_i) \tag{5.4}$$

and we define

$$e_i < e_j, 1 \leq i \leq n-1, i+1 \leq j \leq n \tag{5.5}$$

where $s_i \in S$, $v_i, x_i, w_i \in I$. s_i is a text string that represents the content of the element, v_i is the identifier of the visual area the element belongs to and x_i and w_i are the element *markedness* and *weight* which present a generalization of the visual attributes of the text string as defined further.

From this point of view, the whole text of the document with visual attributes can be expressed as a string of the form

$$M_t = e_1 e_2 e_3 \ldots e_n \tag{5.6}$$

where $e_i = (s_i, v_i, x_i, w_i)$, $1 \leq i \leq n$ are the text elements. s_i corresponds to the appropriate text string in (5.2). v_i is determined during the tree of visual areas is being built. The element e_1 always contains the document title as discussed further in section 5.4.

The Markedness of a Text Element

The visual appearance of a text element e_i can be determined by interpreting the HTML tags in T_j, $j \leq i$ and corresponding CSS styles (this is basically what the web browsers do). The *markedness* of an element determines how much the element is highlighted in the document. In order to determine the element markedness, we analyze and interpret following visual attributes of the text element:

- *Font size.* The font size is used to distinguish important parts of the text especially headers. The relation between the font size and the visual markedness is straightforward – the greater is the font size the more expressive is the text element. The normal text of a document is usually written in some default font size. Small font size can be used for writing some additional information (e.g. copyright notes etc.)

- *Font weight.* Although more degrees of the font weight can be distinguished, from the point of view of a reader we can distinguish two of them – normal and **bold**. The text element written in bold is always more visually expressive as a normal text with the same attributes.

- *Font style.* The font style can be normal or *italic*. It is also possible to use the font style called *oblique* or *slanted*. This style is however very similar to italic and it is usually not distinguished by the readers even nor by some web browsers. Writing the text element in italics is often used to reach its higher visual markedness.

- *Text decoration.* The text element can be underlined in order to increase its markedness or, on the contrary, striked-out to decrease its markedness. Overlined elements are usually not used and there is no clear interpretation of overlining.

- *Text color.* The document text is usually written in one color. Different colors can be however used for highlighting some parts of the text. According to this, any text element written in other than default color higher visual markedness.

Based on the above visual properties, we define following heuristic for computing the value of markedness of a text element:

$$x = (F \cdot \Delta f + b + o + u + c) \cdot (1 - z) \tag{5.7}$$

where b, o, u, c and z have the value 1 when the text element is bold, oblique, underlined, color highlighted or striked-out respectively and 0 if they are not. Δf is the difference $f - f_d$ where f is the font size of the element and f_d is the default font size for the document. The constant F defines the relation between the text size and its markedness. For $F > 4$ the element with greater font size is always more important than the element with lower font size. This corresponds to the usual interpretation.

Element Weight

Some of the text elements are used as headings or labels that describe the contents of a corresponding section of the document on a higher level of abstraction. The weight of the element describes the position of the element in the hierarchy of headings. The main title of the document has the highest weight while the normal text has the weight of zero. For determining the element weight we analyze following factors:

- *The markedness of the element.* The headings are usually written in larger font or at least in bold. The more expressive is the text element the higher should be its weight.

- *Element position.* The position of the element is critical for deciding if the element is a heading or it is not. An element that lies in the continuous block of the text cannot be considered as a heading. For example the word "weight" in the figure 5.5 cannot be considered as a heading even if it has greater markedness that the surrounding text. Generally, we can say that the must be placed at least at the beginning of a line. An element e_n is placed at the beginning of a line iff any of the tags in T_n causes a line break. Such tags are defined in see [54].

- *Punctuation.* The punctuation can be used for denoting the term-description or property-value pairs of elements. The elements that contain the text ending with the colon should have higher weight than the same element that doesn't end with the colon.

Considering these factors we can define the weight of a text element based on a heuristic similar to the definition of markedness:

$$w = \Big[(F \cdot \Delta f) + (b + o + u + c) \cdot l + W \cdot p\Big] \cdot (1 - z) \tag{5.8}$$

where F, Δf, b, o, u, c and z have the same meaning as in the markedness definition (5.7), l and p have the value 1 when the element follows a line break and when the element text ends with a colon respectively and W is the weight of the final colon. An element that ends with a colon should have higher weight that possibly following element written in bold, underlined or highlighted by color. This condition holds for $W > 4$.

5.3 Representing the Hypertext Links

As it follows from the hypertext nature of the world wide web, each document can contain links to another document. These links don't form part of the text content of the document but they can be specified using a special HTML tag `<a>`. Therefore, the links are not directly included in the visual information model[1], they may be however useful for the discovery of the logical documents that is going to be discussed below in section 5.7. For this purpose, it is necessary to represent the included hypertext links in some way.

In HTML, the link is assigned to a continuous portion of the HTML text usually called *anchor*. This text is visually distinguished in the document and it is assigned a target URI that points to another document. In our document representation, the text of the document consists of text elements. Thus, we can define a representation of a link as

$$l = (uri, T) \tag{5.9}$$

where uri is the target URI of the link and T is a non-empty set of text elements that form the anchor. Since the anchor must be continuous, T always contains one or more subsequent text elements. For each document, we can create a set

$$L = \{l_1, l_2, \dots, l_n\} \tag{5.10}$$

that contains all the links in the document.

5.4 HTML Code Analysis for Creating the Models

The information necessary for creating both components of the visual information model – the page layout model and the text features model must be obtained from

[1] Only the visual aspect of the links is considered as discussed below in section 5.4

the HTML code of the document. The process of obtaining the necessary data can be split to following steps:

1. Obtaining of the document(s)
2. Tag to CSS conversion
3. Visual area and text element identification
4. Unit unification
5. Computation of the *markedness* and *weight*

The way in which the documents are obtained is not significant for the document modeling. Usually, the documents are retrieved through a network using the HTTP protocol. For the analysis, we have to retrieve the HTML document itself and all the eventual style sheets referenced in the document. When the frames are used, it is also necessary to obtain the code of all the frames.

As the next step, we go through the HTML code and we convert all the tags that modify the values of the visual attributes to CSS specifications. For example, we remove all the and pairs from the document code and we replace it with <span style="font-weight: bold">. Similarly, some visual attribute can be influenced by multiple CSS properties or by different specification methods (absolute or relative value, etc.). In this phase, we convert the specification using HTML tags and CSS properties to a single CSS property for each visual attribute. For each tag encountered, we check also the document-wide style specification that may be also influenced by the id or class attributes of the tag. The resulting CSS style definitions are inserted back to the document using the style attribute of the generic <span> and <div> tags. Various methods and units of the value specification are unified later, in the unit unification phase. Table 5.2 shows a list of all HTML tags and the corresponding CSS properties that influence the values of individual visual attributes of the text. We convert the specifications as follows:

- *Headings* are converted to a text style with the property font-weight: set to bold and the font-size: property is set according to the level of the heading. According to the standard behaviour of the web browsers, the headings of level 1 to 6 have the font size set to xx-large, x-large, large, medium, small and x-small.
- *Table headers* specified using the <th> tag have the property font-weight: set to bold.
- *Links*, i.e. the text that is used as a hypertext link to another documents have the property font-decoration: set to underline.
- *Font specifications* using the <font> tag can be used for modifying the font face, size and color for a part of the text. The font size definitions is converted to the font-size: property specification in CSS and the color attribute is converted to the color: property. We don't deal with the font face in our method.

Attribute	**CSS properties**	**HTML tags**
Size	`font-size:`, `line-height:`	`<big>`, `<font>`, `<h1> - <h6>`, `<small>`
Weight	`font-weight:`	`<b>`, `<h1> - <h6>`, `<th>`, `<strong>`
Style	`font-style:`	`<em> <i>`
Decoration	`text-decoration:`	`<s>`, `<strike>`, `<u>`
Color	`color:`	`<font>`

Table 5.2: Text Visual Attributes

- *Bold* text specified using the `<b>` and `<strong>` tags is converted to `font-weight: bold` specification.
- *Italics* specified using the `<i>` and `<em>` tags are converted to `font-style: italic`.
- *Underlined and overlined text* is specified using the `text-decoration:` property set to `underline` or `line-through` respectively.

The visual properties of some page objects such as headings and table headers are not exactly defined in the HTML or CSS specification. The resulting visual appearance of these elements depends to certain extent on the used web browser and its configuration. The conversion rules listed above have been observed on popular web browsers Mozilla[2] version 1.6 and Konqueror[3] version 3.2.2. Especially the first one is commonly considered to implement all the web standards and recommendations the most precisely. Anyway, since certain variety in interpreting the HTML code by different web browsers shouldn't influence the document understanding by the user, the mentioned rules present one of the possible variants that is however acceptable and de facto standard.

The next phase consists of discovering the visual areas and the text elements in the document. These two tasks are done simultaneously during single-pass analysis of the HTML code. The output of this phase is

1. The tree M_l of visual area identifiers v_n that corresponds to the definition (5.1).
2. A string of *styled elements* $M_s = e_{s1}, e_{s2}, \ldots e_{sn}$.

A *styled element* is basically a text string with an assigned visual area identifier and the style. The style can be represented as a tuple

$$style = (fsize, fweight, fstyle, fdecoration, color) \tag{5.11}$$

where the individual elements correspond to the appropriate CSS properties. Then, a styled element can be defined as

$$e_{si} = (s_i, v_i, style_i) \tag{5.12}$$

[2]`http://www.mozilla.org`
[3]`http://konqueror.kde.org`

where s_i is the text contents of the element, v_i is the identifier of the assigned visual area and $style_i$ is the style of the element.

During the HTML code analysis, we read subsequently the HTML tags from the code of the document. For creating the tree of visual area identifiers M_l, we maintain the stack S_v of currently open visual areas. At the beginning, S_v contains the identifier v_0 of the root area and M_l contains only v_0 as the root element. When a tag is read that implies the start of a new area (any tag from the table 5.1), a new area is open, it is assigned a new identifier v_i. This identifier is added to M_l as a son node of the identifier being currently on the top of S_v and afterwards, v_i is added to the top of S_v. When the corresponding closing tag is encountered, the area identifier on the top of the stack is removed from the stack.

For determining the visual attributes of the elements we maintain a stack S_s of the styles. At the beginning, the stack contains a default style. When any opening tag is encountered in the code, the top of the stack is duplicated. The style on the top of the stack is updated according to the available style definitions for the tag being processed. When a corresponding closing tag is encountered, the top of the stack S_s is removed. The style of the element can be specified by any CSS specification (see the overview in section 2.2). This specification is particularly important for the hypertext links, that are usually underlined by default but any other visual effect can be specified using CSS definitions for the `<a>` tags.

Any non-empty text string encountered between two subsequent tags forms a new styled element. The first styled element e_{s1} is always formed by the document title specified by the `<title>` tag. If no title is specified the element contains an empty string s_1. The weight of the title element should be always greater than the weight of any other text element in the page. Each new element is assigned the identifier of the visual area that is currently on the top of the stack S_v of visual areas and the style that is currently on the top of the stack S_s. Then, the new styled element is appended to the resulting string of styled elements M_s.

In the unit unification phase, the values of `size` and the `color` of the style of all the styled elements from M_s are converted to a unified form. In CSS, the font size can be specified by an absolute value or relatively or by pre-defined keywords (e.g. `small` or `x-large`) and various units can be used. The text color can be specified by pre-defined keywords or by the percentage of red, green and blue color that can be written various ways. During the unification, all sizes are converted to an absolute value in points (`pt`) and the colors are converted to the string *#rrggbb* where *rr*, *gg* and *bb* mean the value of the red, green and blue in hexadecimal from 00 to FF.

In the final phase we create the model M_t of text visual features as defined in 5.6 from M_s by computing the values of *markedness* and *weight* for all the styled elements. For each styled element $e_s i \in M_s$ we create a new text element e_i by copying the values of s_i and v_i and computing the values of the markedness x_i according to the definition (5.7) and the weight w_i according to the definition (5.8). Then, the new element e_i is appended to M_t.

5.4.1 Tables in HTML

In HTML documents, the tables can be used in two basic roles:

- As a standard instrument for presenting the structured, tabular data
- For achieving desired page layout

The use of the tables for achieving some layout is deprecated by the HTML specification [54], the cascading style sheets should be used instead. Using tables for this purpose causes many problems such as slower page rendering and display problems on non-visual media. However, it is very frequent in current world wide web to use the tables this way. For the HTML code analysis, this use of the tables corresponds to the analysis method presented above. Each table cell is interpreted as a separate visual area as well as the whole table itself. The text content is processed independently on the table or cell tags; only the visual area identifier is assigned to each text element.

Structured tables that are used for presenting some tabular data must be handled as a special case in the HTML code analysis. As in a HTML document always holds that the order of the text elements in the document code corresponds to the order of the elements in the displayed document, tables introduce a two-dimensional structure where the logical order of the elements depends on the organization of the table and it can be interpreted various ways. Moreover, the tables can contain a hierarchy that results from the relation between the table header and the rest of the table [62].

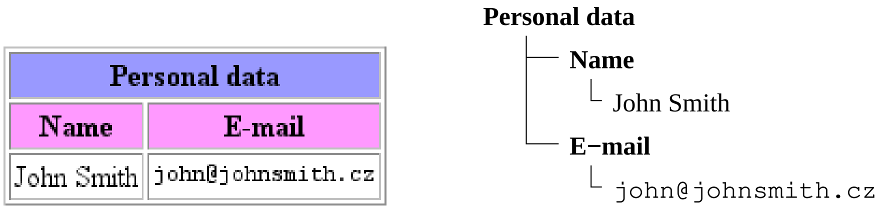


Figure 5.6: An example of a structured table

An example of such a table is in the figure 5.6. The contents of table cells in HTML are always defined row by row so the order of the elements in the HTML code is “Personal Data”, “Name”, “E-mail”, etc. However, a reader interprets such a table as a hierarchy shown in the right part of the figure. It this case, the hierarchy of visual areas that correspond to the table cells is not created by the HTML code only, but the relations among the header and non-header cells must also be considered. Let’s assign numbers from **1** to **5** to the visual areas that correspond to the table cells as shown in the figure 5.7 and let’s consider that the whole table forms a visual area 0. Then, using the HTML processing method specified above, the hierarchy of the visual areas corresponds to the left tree in the figure 5.7. For the reader, however, the hierarchy of the visual areas appears the way corresponding to the right tree in the figure 5.7. For this reason, a special algorithm must be used for processing the structured tables.

When a table is encountered in the HTML code, the first step is to decide which are the header cells. HTML allows to distinguish between the header cells and the data

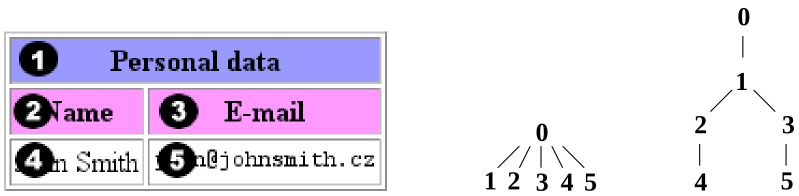


Figure 5.7: Visual areas in a structured table

cells using the <th> or <thead> tags, this feature is however not commonly used and in some cases it is used incorrectly. Therefore, we propose determining the headers by comparing the visual appearance of the table cells:

- We assume that the header is formed by the top row or the leftmost column of the table.
- All the cells in the header must have consistent visual style: font, color and background.
- When a header row or column is encountered, we add each header cell to the tree of visual areas as child nodes of the table visual area. After this, we consider each part of the table that is covered by a single header cell as a separate subtable, that is formed by one or more rows (when the header cells are in the first column) or columns (headers in the first row) and we repeat the header identification process recursively on all the subtables. The process ends when no headers can be identified in the subtables. In this case, we assume that the remaining cells only contain the data values.

At the end of this process we obtain a hierarchy of headers. In the next step, we add the all the cells of the remaining subtables to the visual model as the child nodes of the respective header cell they belong to. An illustration of this process is in the figure 5.8. In steps 1 and 2 we detect the headers of the (sub)tables. In step 3, no more headers can be detected and the data cells are just included to the tree.

5.4.2 Example of Visual Models

In order to demonstrate the HTML parsing process, let's consider the simple document shown in the figure 5.9. The HTML tags that influence the visual appearance of the text elements are shown in the shaded boxes.

The root visual area is formed by the document. The only object that introduces sub-areas to the document is a simple table. In this table, a structure can be detected. The resulting tree of visual areas is shown in the figure 5.10.

Table 5.3 shows the visual attributes of all the text elements in the page and the values of *markedness* and *weight* computed from the definitions (5.7) and (5.8) for $F = 5$ and $W = 5$. As stated in the section 5.2.2, the first element e_1 is always the title of the document specified using the <title> tag.

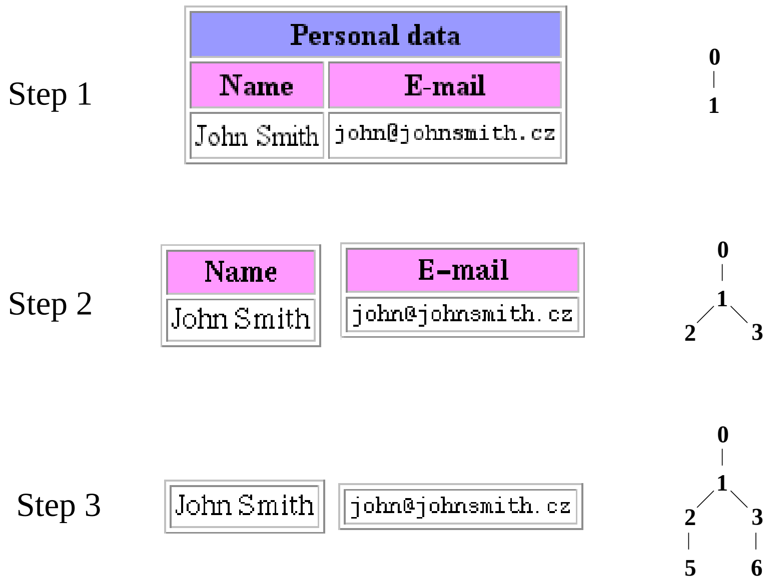


Figure 5.8: Example of the table structure detection

The resulting model of text visual attributes is the string of tuples (s, v, x, w) of the corresponding values in each line of the table. Note that the visual area identifier v of each text element corresponds to visual areas in the figure 5.10.

5.5 Logical Structure of a Document

The next step of the proposed information extraction method consists of creating the model of logical document structure based on the visual information gathered and modeled in previous steps. As defined in the introduction, logical document structure is a hierarchy of the text elements in a document as it is interpreted by a reader. This hierarchy should respect the logical relations among the text elements as the follow from the visual appearance of the text elements independently on the particular HTML code. As an illustration, figure 5.11 shows the logical structure of our sample document from the previous section.

Formally, the logical document structure is an ordered tree of text elements. It can be denoted as a graph

$$S = (V_S, E_S) \tag{5.13}$$

where $V_S = \{e_1, e_2, \ldots e_n\}$ is an ordered set of all the text elements in the document where e_{i-1} precedes $e_i \forall 1 < i <= n$. The set E_S of edges of the tree contains the tuples (e_i, e_j) such that $e_i \in V_S$ and $e_j \in V_S$ and e_i is directly superior to e_j according to the interpretation of their visual attributes and their position in the document. In order to ensure that S is a tree, it must hold that each text element except e_1 which is the page title has exactly one directly superior element (its parent element in the tree). The element e_1 always forms the root node of the tree.

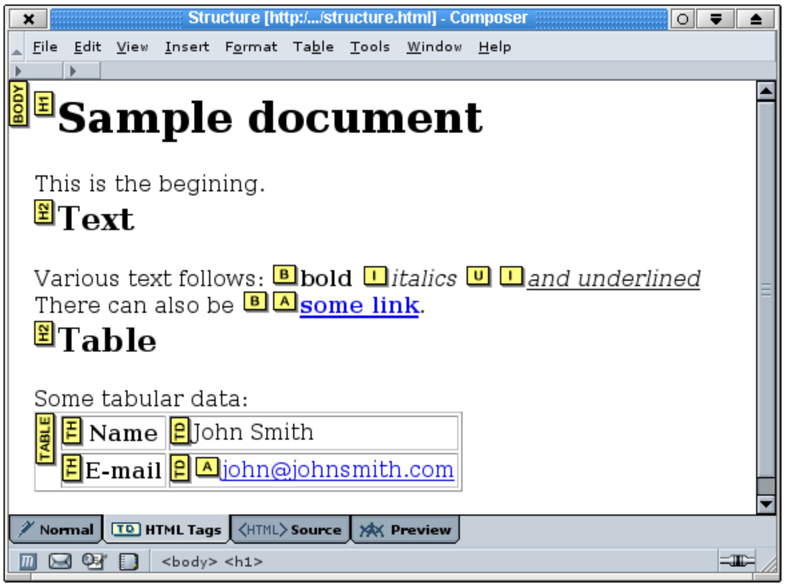


Figure 5.9: Sample document

The set of edges E_S of the tree is derived from the page layout model M_l (5.1) and the model of typographical attributes of the text M_t (5.6) by an algorithm that consists of following two phases:

1. Creating the set of graph edges E_V such that $S_V = (V_S, E_V)$ is a tree of text elements and for any $e_i, e_j \in V_S$ e_i is an ancestor of e_j iff the corresponding visual area identifier v_i is an ancestor of v_j in the tree of visual areas M_l. We call S_V a *frame* of the logical document structure.

2. Creating E_S by copying all the edges (e_i, e_j) from E_V and replacing e_i by one of its descendants if needed, so that the element of a higher weight is always an

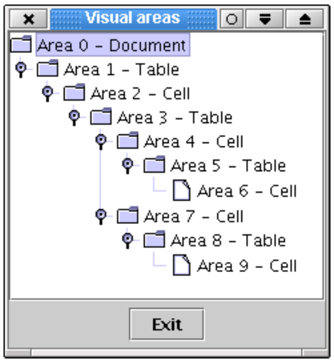


Figure 5.10: Visual areas in the sample document

s	Δf	b	o	u	c	z	l	p	v	x	w
Structure	0	0	0	0	0	0	0	0	0	0	0
Sample document	8	1	0	0	0	0	1	0	0	41	41
This is the beginning.	0	0	0	0	0	0	1	0	0	0	0
Text	5	1	0	0	0	0	1	0	0	26	26
Various text follows:	0	0	0	0	0	0	1	1	0	0	5
bold	0	1	0	0	0	0	0	0	0	1	0
italics	0	0	1	0	0	0	0	0	0	1	0
and underlined	0	0	1	1	0	0	0	0	0	2	0
There can also be	0	0	0	0	0	0	1	0	0	0	0
some link	0	1	0	0	0	0	0	0	0	1	0
.	0	0	0	0	0	0	0	0	0	0	0
Table	5	1	0	0	0	0	1	0	0	26	26
Some tabular data:	0	0	0	0	0	0	1	1	0	0	5
Name	0	1	0	0	0	0	1	0	4	1	1
John Smith	0	0	0	0	0	0	1	0	6	0	0
E-mail	0	1	0	0	0	0	1	0	7	1	1
john@johnsmith.com	0	0	0	1	0	0	1	0	9	1	1

Table 5.3: Text elements in the sample document

ancestor of all the elements with a lower weight within the visual area.

The algorithms for both steps follow.

Algorithm 1 Creating a frame of the logical structure

Input: $V_S = \{e_1, e_2, \dots e_n\}$ – the set of text elements
M_l – page layout model
Output: $S_V = (V_S, E_V)$ – a frame of the logical structure
$current = e_1$
For each $e_i = e_2, e_3, \dots e_n$; $e_i = (s_i, v_i, x_i, w_i)$ **do**
    if $v_i \neq v_{i-1}$ **then**
        if v_i **descendant of** v_{i-1} **then**
            $current = e_{i-1}$
        else
            $current$ is the nearest e_j such that
                e_j is ancestor of e_{i-1} in S_V and
                v_j is ancestor of v_i in M_l
Add $(current, e_i)$ to E_V

Algorithm 2 Applying the element weight

Input: $S_V = (V_S, E_V)$ – a frame of the logical structure where

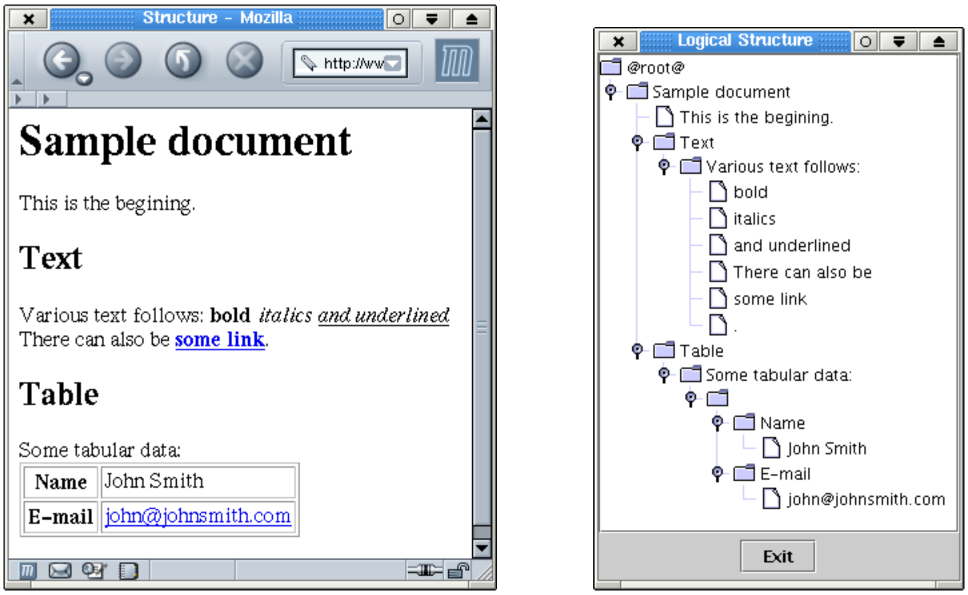


Figure 5.11: Sample document with the corresponding logical structure

$e_p \in V_S$ is a root of S_V
$e_{c1}, e_{c2}, \ldots e_{cn} \in V_S$ are child nodes of e_p;
$e_{c1} < e_{c2} < \ldots < e_{cn}$

Output: $S = (V_S, E_S)$ – the logical structure tree

if $n > 0$ **then**
  $current = e_p$
  Add (e_p, e_{c1}) to E_S
  For each $e_{ci} = e_{c2}, e_{c3}, \ldots e_{cn}$; $e_{ci} = (s_i, v_i, x_i, w_i)$ **do**
    Recursively apply algorithm 2 for $e_p = e_{ci}$
    if $w_i < w_{i-1}$ **then**
      $current = e_{ci-1}$
    if $w_i > w_{i-1}$ **then**
      $current =$ nearest e_{cj} such that
        e_{cj} is ancestor of e_{ci-1} and
        e_p is ancestor of e_{cj} and
        $w_{cj} > w_i$
      when such e_{cj} doesn't exist then $current = e_p$
    Add $(current, e_{ci})$ to E_S

The resulting model of the logical structure exhibits an important feature that can be described as follows. Let *text* be the whole text of an HTML document that can be obtained by taking the contents of the `<body>` tag of the document and discarding all the embedded HTML tags. Let $e_1, e_2, \ldots e_n$ be a string of text elements obtained by a pre-order traversal of the resulting tree S and let s_i be the text string contained in the text element e_i. Then, it holds that

$$text = \prod_{i=2}^{n} s_i \tag{5.14}$$

In other words, when concatenating all the text strings except the first one (the title) in the logical document structure during the pre-order traversal of the tree, we obtain the complete text of the document.

5.6 Information Extraction from the Logical Model

As defined in the previous section, the logical structure of a document is a tree whose nodes are formed by all the text elements contained in the document. When using the logical structure model for information extraction, the task is to locate the nodes of the tree that contain the desired information. We assume that the document is designed to be understandable by a user and we analyze the way in which the visual information is used by the user when looking for a particular data in the document.

The first assumption is that the user usually knows an approximate form of the information he is looking for. For example, when looking for a price, we are trying to find a number preceded or followed by a code of currency. The second assumption is that there is a hierarchy of headings and labels provided by the author of the document that allows the user to interpret the document contents effectively. The user is thus provided with three types of navigational cues:

1. The *relations* among the text elements in the document that correspond to the logical document structure and that are expressed using the visual design of the document.

2. The *labels* provided by the author of the document.

3. The expected *format* of the desired information.

When looking for a particular information, the user typically navigates through the logical structure of the document as suggested by the visual means and tries to choose the path leading to the desired information by classifying the headings and labels that in our case correspond to the nodes of the structure tree. Normal user can use the natural language understanding for choosing the best path. Since the natural language processing is a non-trivial task, we propose a simplified solution based on regular expressions and approximate tree matching algorithms.

Let's return back to the original idea of information extraction from data-intensive documents that is based on the extraction of the complete data records rather than on extracting the single data values. Further we will assume that each data value, when it is present in the document, forms exactly one text element. This assumption is somewhat simplifying because it doesn't allow the situations when the information is split to several text elements or, on the contrary, when there is an additional text present in the same text element. However, as follows from the text element definition (5.4) on the page 38, this assumption corresponds to the following situation:

- The data value is visually consistent (thus it forms a single text element)
- The data value is visually separated from the surrounding text by any HTML tag so that the surrounding text doesn't form part of the text element

In the data intensive documents, the acceptance of these rules can be expected. The impact of this simplification to the performance of the information extraction method is discussed in the evaluation part of this thesis.

Let's assume that each data record r to be extracted consists of $|r|$ data values (an example of such a data record is given on the page 3). When we admit that some data values can be missing in the document, when extracting the data record, the task is to locate m text elements in the logical document structure that correspond to the appropriate data values; $N \leq m \leq |r|$ where N is the minimal number of values that must be located for considering the data record to be found. All the located text elements that form a single data record are located in certain subtree of the logical structure tree. Aside from the data values, this subtree contains the text elements that correspond to the labels that denote the value in the documents and some remaining text elements that can contain additional notes etc.

From this perspective, the information extraction task can be viewed as a task of locating the subtrees of logical structure tree that meet following requirements:

- The subtree contains the data values of the expected form and the expected labels.
- The data values are logically related to the appropriate labels; i.e. the text element containing the potential data value is a descendant of an element containing the appropriate label.

For locating the appropriate subtrees of the logical structure tree, we propose an approach based on tree matching algorithms.

5.6.1 Using Tree Matching

The proposed approach is based on the specification of a template of a subtree that is to be located. This specification consists of two steps:

1. We specify the expected logical structure of the extracted data record
2. We add the information about the expected labels and the format of the data values. Regular expressions are used for this specification.

The result of this specification is a template of a tree that can be interpreted as a structured query to the database of all the subtrees of the logical structure tree.

Let's consider a simple example in the Figure 5.12. From a personal page, we want to extract the name, department and the e-mail address of that person. The left tree defines an expected logical structure of the extracted information. In case of the personal pages, the name of the person is usually presented as a superior text (heading) and the remaining data is placed further in the document. We extend the tree by replacing the fields with the regular expressions that denote their expected format

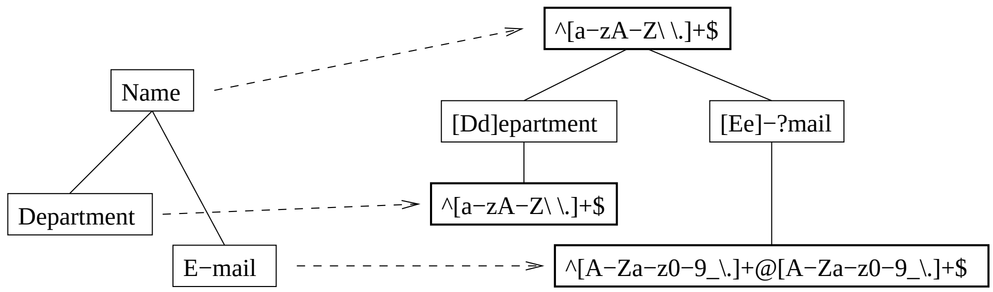


Figure 5.12: An example extraction query with regular expressions

and we add the expressions that denote expected labels of these data. The resulting tree can be viewed as a structured query and tree matching algorithms can be used for identifying all matching subtrees of the logical document structure.

For simple extraction tasks, we can simply estimate the logical structure of the data records as shown on the previous example. For more complex task, it is necessary to obtain at least one sample document in order to determine the logical structure of the contained data records. Furthermore, there usually exist several variants of structuring the data records. For example, the previous example shown in the figure 5.12 could be changed so that the name is presented at the same level as the remaining data. For this reason, it may be necessary to modify the expected data structure when processing a new set of documents. By adapting the extraction task for further sets of documents, we obtain several variants of the template subtree that cover all possible variants of presenting the extracted data. However, in comparison to wrappers, where the number of possible variants is extremely high and therefore the wrappers must be re-generated for each set of documents, in our case, there are only a few variants of the logical structure and with the increasing number of processed data sets, the frequency of the modifications decreases rapidly. For example, for the sample task shown in the figure 5.12, two variants of the task specification are sufficient as shown in the method evaluation part in section 7.1.1.

5.6.2 Approximate Unordered Tree Matching Algorithm

For tree matching, we use a modification of the **pathfix** algorithm published by Shasha et al. [56]. This algorithm solves approximate searching in unordered trees based on the root-to-leaf path matching. The original problem is to find all data trees D from a database $\mathcal{D}$ that contain a substructure D' which approximately matches a query tree Q within distance $DIFF$. The distance measure is defined as the total number of root-to-leaf paths from Q that do not appear in D'. Let's consider an example of the query tree Q in Figure 5.13, which has two paths. The query tee matches data tree D_1 with distance of 0 (both paths from Q are present in D_1). For the tree D_2, there are two possible matches with distance of 1 (the path **A-D** is not present in the tree but the path **A-B** has two occurrences in D_2).

In order to use this algorithm for our purpose, we introduced following modifica-

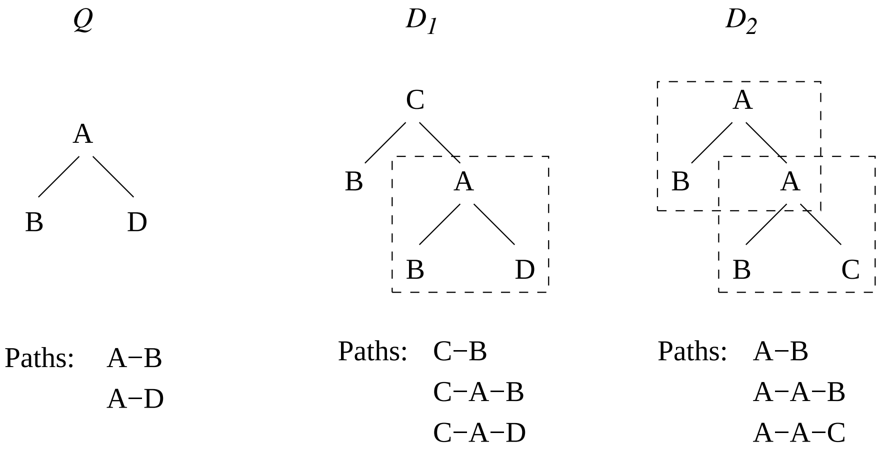


Figure 5.13: Pathfix algorithm illustration

tions:

- Instead of the database of trees we have only one tree (the logical structure tree) and we want to locate the subtrees (data records) that match the query tree. Thus in our case, the database $\mathcal{D}$ is formed by all the subtrees of the logical document structure S with the root node that matches the root node of the query tree.
- The values in the nodes of the trees don't have to match exactly since the query tree Q contains regular expressions in its nodes. We define, that a node in the logical structure tree S matches a node in the query tree Q iff the appropriate text element matches the regular expression in the query tree node.
- Since the exact form of the logical document structure may differ in some range among the web sites, we have extended the path matching algorithm by introducing two more parameters:
  - $QSKIP$ – number of nodes from the query path that don't match any node from the matched path in S. This corresponds to the situation that some labels or data values that are expected by the tree template but they are missing in the document.
  - $MSKIP$ – number of nodes in the matched path from S that don't match any node in the query path. This corresponds to the situation that there are some extra nodes in the logical structure tree that are not expected by the tree template.

  These parameters allow to consider the paths as they were matching event if there is certain number of missing or excessive nodes.

The accuracy of the results depends on the values of the $QSKIP$ and $MSKIP$ parameters and on the allowed number of missing paths $DIFF$. More detailed analysis of the influence of the parameters to the precision and recall of the method is given in Chapter 7.

5.7 Information Extraction from Logical Documents

Analogous to the physical documents, the logical documents also exhibit a hierarchical organization for similar reasons (see the discussion at page 25, section 3.3.3). In contrast to the physical documents, the hypertext allows to give the logical document any organization that can be modeled using a directed graph. On the other hand, the organization must be easily understandable to the document users. From the practical point of view, considering this requirement, it is not advisable to use a more complex organization than the hierarchical one. As observed by [60], when we extract only the links intended to be the routes through which the readers go forward within a document, in most cases, we obtain a sequence or a hierarchy of pages.

Let's represent the logical document D as a tuple

$$D = (D_p, I) \tag{5.15}$$

where D_p is a set

$$D_p = \{d_1, d_2, \ldots, d_n\} \tag{5.16}$$

where $d_i; 1 \leq i \leq n$ are the physical documents and $I \in D_p$ is the main (index) page. In the context of using the information extraction method described in the above chapters, the physical document d_i can be defined as a tuple

$$d_i = (uri_i, S_i, L_i) \tag{5.17}$$

where uri_i is the unified resource identifier of the physical document, S_i is the model of the logical structure as defined on page 46 and L_i is the set of links defined on the page 40.

The extension of our information extraction method from physical to logical documents is straightforward. We assume, that the user specifies the URI of a document when invoking the information extraction process. For discovering the logical documents, we adopt the method published in [60], which is also discussed in the section 3.3.2. The information extraction process consists of following steps:

- We consider the document whose URI has been specified by the user the index page I of the logical document. We use the method [60] for obtaining the URIs of the remaining physical documents $d_i \in D_p$.

- We create the models of the logical structure S_i for each physical document d_i separately using our method proposed in the above sections. Simultaneously, we create the set of links L_i for each d_i.

- We create a single model of the logical structure S for the whole logical document D by joining the logical structure trees S_i of the physical documents to a single tree.

When joining the logical structure trees S_i to the global model of the logical structure S, we proceed following way:

- Let the root node of I (the logical structure tree of the index page) be the root node of S.
- Let's assume that the index page I corresponds to a physical document $d_j = (uri_j, S_j, L_j)$. For each link $l = (url, T)$; $l \in L_j$ we select the text element $e \in T$ that precedes all remaining elements contained in T (see the definition (5.4) on the page 38). If the uri of the link corresponds to the URI of any document $d_k \in D_p$, then we add the logical structure tree S_k to S as a subtree of the selected element e.
- We repeat the process recursively for each added subtree.

After finishing this process, we have created the model of the logical structure for the whole logical document. This model can be used for information extraction as specified in section 5.6. We can see, that this approach abstracts from the physical organization of the information to documents by creating an unified model for all the physical documents.

Chapter 6

Experimental System Implementation

In order to evaluate the proposed information extraction method in the real world wide web environment, we have implemented an experimental system for extracting information from the data intensive documents in the web.

6.1 System Architecture Overview

General architecture of our information extraction system is shown in the figure 6.1. The system consists of four basic modules:

- *Interface module* implements an interface between the system and the Internet. It is responsible to download the necessary HTML documents and to store them for further analysis.
- *Logical document module* implements the discovery of logical documents. Given a URI of the main page, it analyzes the hypertext links and creates a list of URIs of the documents that form the logical document.
- *Analysis module* provides the analysis of the HTML code in order to create the model of the visual information. Further, it implements the transformation of the visual information model to the logical structure model. Finally, it provides unification of the logical structure models to the unified tree of the logical structure.
- *Extraction module* implements the information extraction from the logical structure model using the tree matching algorithms. The input is the user-specified extraction template and the output is the extracted data.

In order to reach maximal flexibility of the system and in order to be able to evaluate the individual parts separately, the modules are implemented as standalone pieces of software that communicate witch each other. The communication with the interface module is simple – the module works as an HTTP proxy, so that the communication is implemented using the HTTP protocol. The communication among the remaining

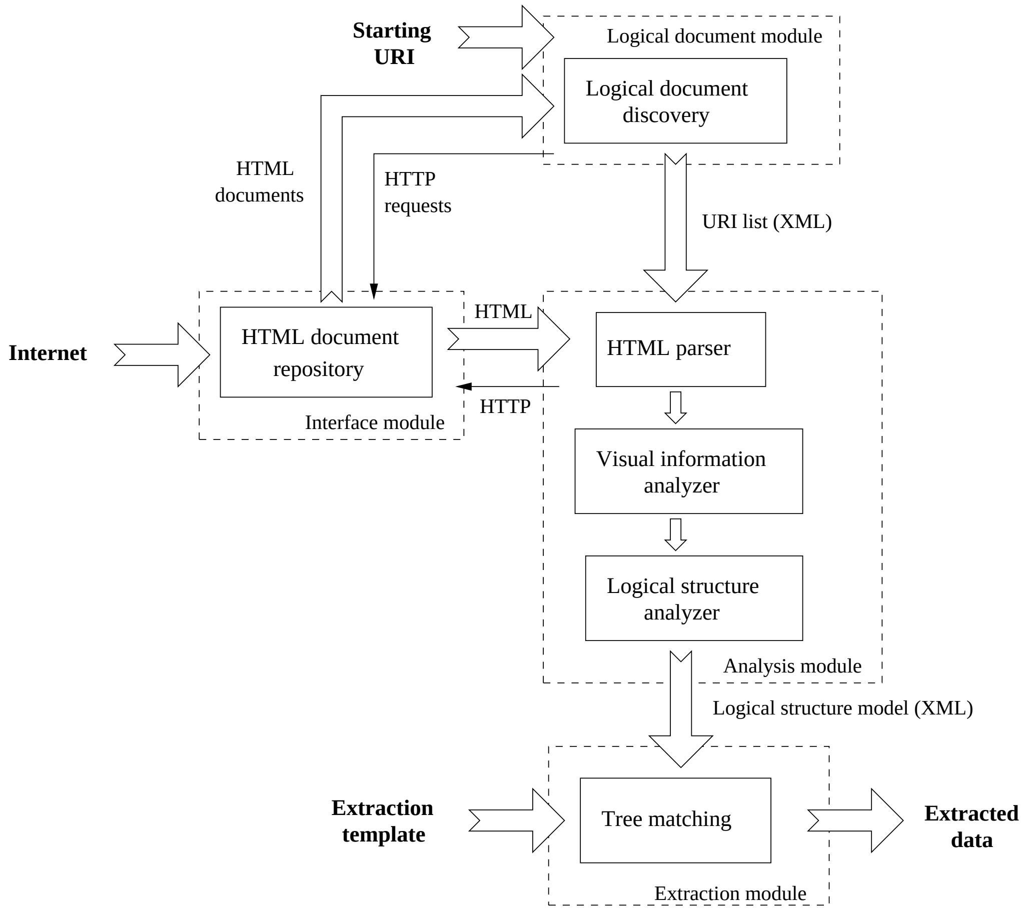


Figure 6.1: System architecture overview

```
<?xml version="1.0" encoding="iso-8859-2"?>
<!DOCTYPE logical_documents SYSTEM "ld.dtd">
<logical_document url="http://www.xyz.cz/">
    <document url="http://www.xyz.cz/index.php" />
    <document url="http://www.xyz.cz/aboutus.php" />
    <document url="http://www.xyz.cz/contact.php" />
</logical_document>
```

Figure 6.2: XML representation of a logical document

modules is based on the use of XML. This solution allows to achieve maximal interoperability of the modules – each module can be used separately and the produced output can be used for any task. This is particularly useful for representing the logical document structure, which can be used not only for information extraction but also for other tasks where the existing XML processing technologies such as XSLT or XQuery can be used. For this reason, we now describe the use of XML in our system in more detail.

6.2 Using XML for Module Communication

The XML based communication is used in two points of the system. First, XML is used for transferring lists from the logical document module to the analysis module. Secondly, the most important use of XML is the representation of the global model of the logical structure. Furthermore, XML is also used for the extraction task specification. This application of XML is discussed below in the section 6.3.4. Formal definitions of the used XML formats are given in appendix B.

6.2.1 Representing the Logical Documents

The data to be represented is formed by a simple list of URIs of the physical documents that form the logical document. We represent each physical document by a `<document>` tag with an attribute `uri` that contains the URI of the physical document. The root tag `<logical_document>` encloses the list of physical documents. An example XML representation of a logical document is shown in the figure 6.2. The full DTD (Document Type Definition) of the logical document XML representation can be found in appendix B.2.

6.2.2 Logical Structure Representation

Since the logical document model is a tree of text elements, the XML representation is straightforward. Each text element $e = (s, v, x, w)$ is represented by a single XML tag `<text>` with the attributes that represent individual values of the tuple: the `content` attribute carries the text content of the elements and the attributes `visual`, `expr` and

```
<document title="Sample" url="http://sample.org">
    <text content="Personal data" visual="2"
                         expr="3" weight="3">
        <text content="Name" ...>
            <text content="John Smith" .../>
        </text>
        <text content="E-mail" ...>
            <text content="john@johnsmith.org" .../>
        </text>
    </text>
</document>
```

Figure 6.3: XML representation of the logical structure model

`weight` contain the values of v, x and w respectively. Furthermore, a `<document>` tag is used as a root element. Figure 6.3 shows an example representation of a logical structure of the document shown in the figure 5.6. The attributes of some elements have been omitted for greater clarity. The full DTD of the logical structure XML representation can be found in appendix B.3.

6.3 Implementation

All the system modules have been implemented on the Java platform. The main reasons for this choice were:

- Portability of the resulting product
- HTTP protocol support
- HTML parser available
- Large set of suitable data structures, mainly trees

On the other hand, Java brings a drawback in the form of slightly worse performance of the running application. However, this point is not critical for the prototype.

Each module is implemented as a standalone application. Each of the modules except the interface module can run either separately as an application with a graphical user interface or together with other modules. We will now briefly describe the implementation of each module.

6.3.1 Interface Module

This module provides the interface between the system and the Internet. It works as an HTTP proxy, the communication with the remaining modules is implemented using the HTTP protocol [24]. Each module sends a request on a document with a certain

URI. The interface module downloads the document and stores it locally for later use. Then it generates an HTTP reply containing the code of the document. The stored documents are valid for a single information extraction task only. For improving the reliability of the downloading process, any publicly available general HTTP cache can be used as for example Squid[1].

6.3.2 Logical Document Module

When provided with a URI, this module analyzes the structure of links leading from this document and discovers the boundaries of the documents. As the result, it produces the list of the URIs of physical documents.

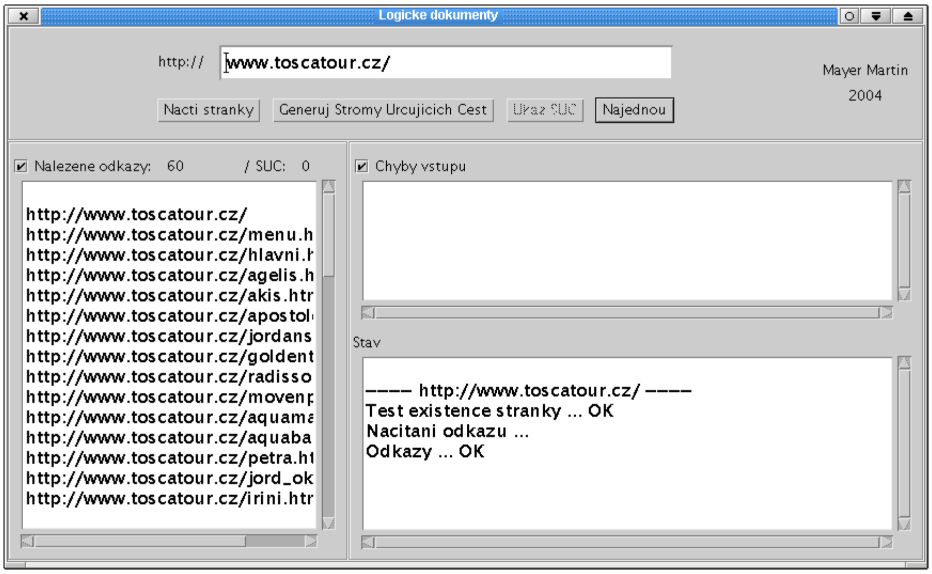


Figure 6.4: Logical document module

6.3.3 Analysis Module

This module implements the methods of HTML code parsing proposed in the section 5.4, visual information modeling in documents that has been proposed in section 5.2 and transforming this model to the logical structure model as proposed in section 5.5. The resulting model of the logical structure is represented using an XML document as described in section 6.2.2.

6.3.4 Extraction Module

The extraction module implements the tree matching algorithm defined in section 5.6. Given the logical structure of the document obtained from the analysis module, it

[1] http://www.squid-cache.org

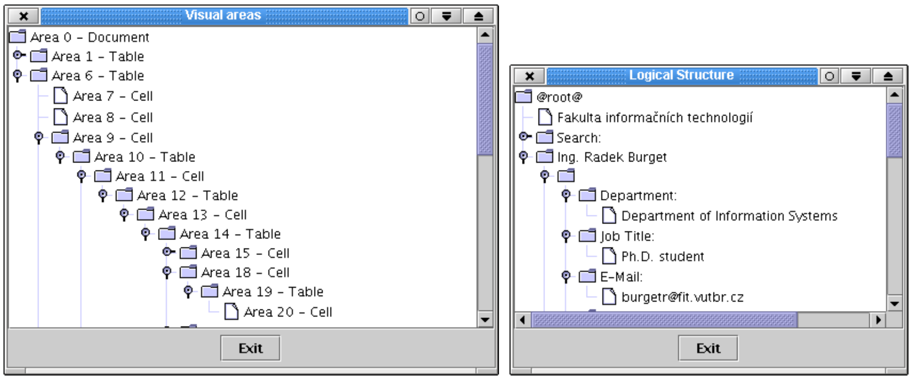


Figure 6.5: Analysis module

attempts to locate subtrees of the logical structure that correspond to the extraction task specification.

The task specification corresponds to a template of a subtree as shown on an example in the figure 5.12 (page 52). Again, we use XML for defining the information extraction task. The XML specification has following format:

```
<task name="...">
    <input>
        <!-- Element format specifications -->
    </input>
    <model name="...">
        <!-- data and label hierarchy spec. -->
    </model>
</task>
```

The whole specification is enclosed in the `<task>` tag and it is assigned an unique name. The specification consists of two parts. In the `<input>` part, the formats of text elements are specified. Each format specification consists of a name that identifies the format and a regular expression that specifies the format itself. Additionally, it can be specified if the regular expression is compared in a case sensitive or case insensitive manner. For example, the format for a name of a department can be specified as

```
    <spec case="lower" name="dept">^[a-z0-9\ $]+</spec>
```

The `<model>` part of the task specification defines the template tree. In this definition, two tags can be used: the `<label>` tags corresponds to a label in the template, `<data>` tag corresponds to a text field containing a data value. These tags can be arbitrarily nested in order to create the corresponding hierarchy. Both these tags specify an expected format of a logical structure tree node by using one or more format specification from the `<input>` section, for example

```
    <label spec="poslabel|joblabel"/>
```

means that the node matches to a node in the logical document structure that has either the "poslabel" or the "joblabel" format that have been previously specified using regular expressions in the `<input>` sections. The difference between `<label>` and `<data>` is that the label is just matched to a structure tree node whereas the data forms part of the extracted output data. Therefore, a data node must have a `name` attribute set, e.g.

```
<data spec="posval" name="position"/>
```

that corresponds to the name of the field in the output data. An example of a complete extraction task specification is in Appendix A.

The module provides a graphical user interface (see the figure 6.6) that allows to browse the information extraction task and the tree of the logical structure and to browse the paths in the trees that correspond to each other.

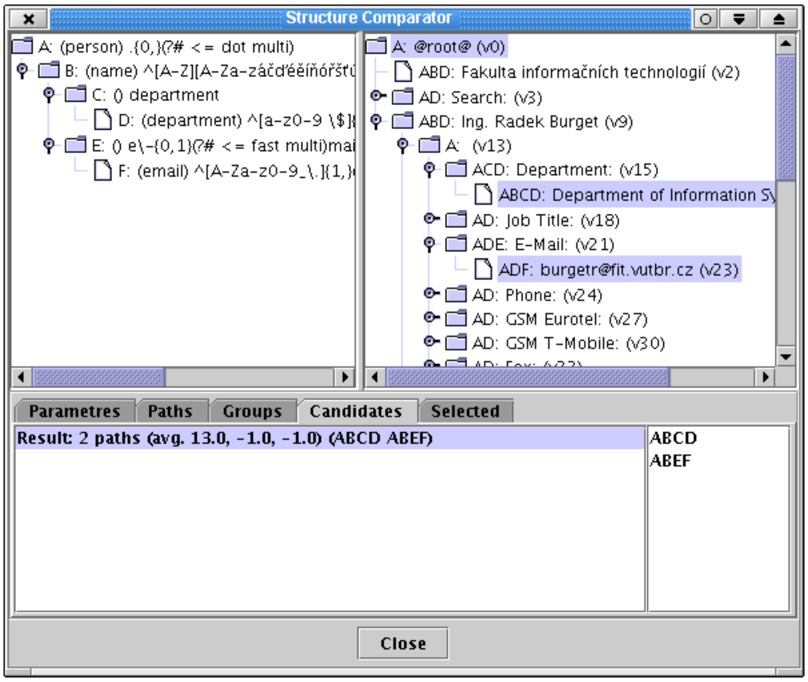


Figure 6.6: Extraction module

6.3.5 Control Panel

Control panel is a simple application that provides an unified user interface for all the modules. It allows to specify the target URL and the extraction task specification file. Then, it calls subsequently the individual modules and finally, it displays the information extraction results.

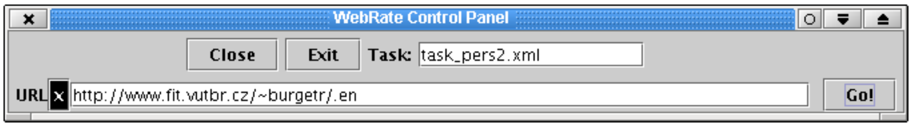


Figure 6.7: Control panel

6.4 Information Extraction Output

The output of the information extraction task is a sequence of data records where each data record consists of the values of named data fields that have been defined in the extraction task specification. In our systems, two output formats are supported: a XML file and SQL script for storing the data to a relational database.

6.4.1 Extracted Data as an XML Document

The format of the resulting XML file is derived from the extraction task specification:

- The sequence of all the output records is enclosed in a root tag whose name corresponds to the specified extraction task name.
- Each output record is enclosed in a tag whose name corresponds to the name of the extraction model.
- Each record consists of data fields that are enclosed in tags whose names correspond to the specified data field names in the model section of the task specification.

An output document for the example task shown in Appendix A would have following structure:

```
<?xml version="1.0" encoding="iso-8859-2"?>
<staff>
    <person>
        <name>...</name>
        <department>...</department>
        <email>...</email>
    </person>
    <person>
    ...
    </person>
    ...
</staff>
```

6.4.2 Extracted Data as an SQL Script

The resulting SQL script consists of a sequence of `INSERT` commands that store the extracted records to a database table. The commands have following format:

- The name of the target table corresponds to the name of the name of the extraction model specified using the <model> tag in the extraction task specification.
- The names of the table columns correspond to the specified data field names in the model section of the task specification.

An output script for the example task in Appendix A would have following format:

```
INSERT INTO person (name, department, email) VALUES (..., ..., ...);
INSERT INTO person ...
...
```

We assume that the corresponding database table has been created in advance by the user. In order to create the appropriate SQL commands automatically, it would be necessary to extend the extraction task specification by a posibility of specifying the data types of individual fields.

Chapter 7

Method Evaluation

In order to evaluate the functionality of the proposed method, we have first tested the proposed information extraction method on selected sets of physical HTML documents available in the World Wide Web. As the next step, we have run the tests on larger logical documents on the same web sites.

7.1 Experiments on Physical Documents

We have tested the method on two different cases. The first experiment was extracting the personal information from various university staff member pages. These pages are usually very strictly formatted and regular. The second experiment deals with stock quote servers. In this case the documents are significantly more complex, contain various kinds of data and their structure exhibits various irregularities.

7.1.1 Experiment 1 - Personal Information

We have tested the proposed method on a simple task that consists of processing the directory pages of various universities and extracting the name, e-mail and department of the persons listed in these pages. We have defined two extraction models that differ in the expected logical structure of the presented information in the page. The two possibilities are shown in the figure 7.1.

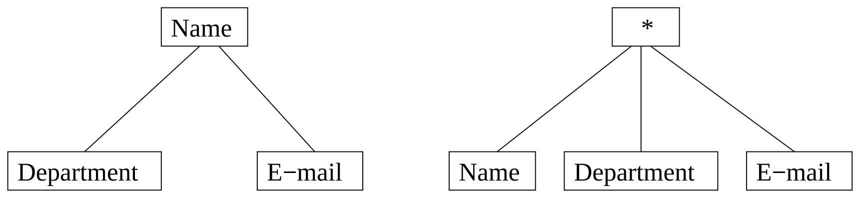


Figure 7.1: Two variants of the expected logical structure

The first variant corresponds to the situation where the name of the person is used as a page title or a heading. The second variant corresponds to the situation where

#	URL	Precision %	Recall %	Variant
1	`fit.vutbr.cz`	91	100	a
2	`mff.cuni.cz`	100	100	a
3	`is.muni.cz`	100	52	a
4	`mit.edu`	-	-	a
5	`cornell.edu`	100	100	a
6	`yale.edu`	100	100	a
7	`stanford.edu`	100	100	b
8	`harvard.edu`	100	100	b
9	`usc.edu`	-	-	b
10	`psu.edu`	100	100	b

Table 7.1: Sample extraction task results

the name is presented on the same level as the remaining data fields. By adding the expected field formats and the expected labels as discussed in section 5.6 we obtain a template tree for each variant (we will call these variants A and B) as shown in the figure 7.2. The XML extraction task specification of both variants is included in Appendix A. For simplicity, all the regular expressions except the name value have the `case=lower` attribute set in the specification so that the text is always converted to lowercase before being matched with the regular expression. The format of name is matched in the case-sensitive manner since the name should start with a capital letter.

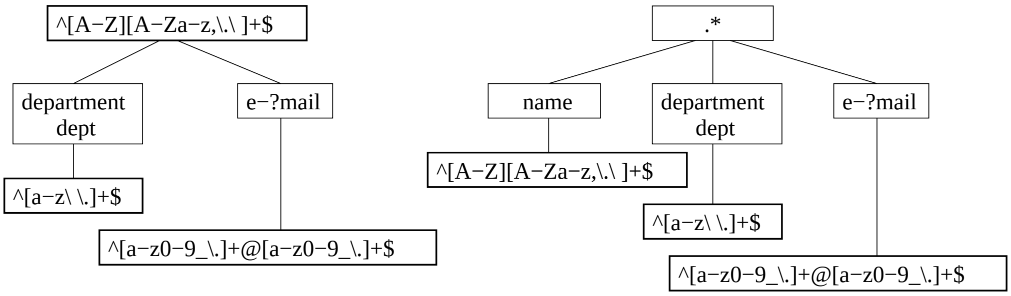


Figure 7.2: Template trees (variants A and B)

As the data source we have used sets of staff personal pages from various universities. Table 7.1 shows values of *precision* (3.1) and *recall* (3.2) as defined in section 3.1.3. From each listed site, we have taken 30 random personal pages. The only input for the information extraction is the URI of the document and the appropriate tree template A or B.

There are three basic reasons that may cause the extraction process to fail:

- The extracted data is not labeled as expected; e.g. in the testing set 3 the e-mail addresses are not denoted by any label

- The data to be extracted is contained inside of larger text elements; e.g. the appropriate data at `mit.edu` (4) is presented as unformatted text only and at `usc.edu` the labels are not distinguished from the values
- Various anti-spam techniques used in the documents such as writing e-mail addresses in some non-standard form or presenting is as an image

During the tests, the parameters have been set to $QSKIP = 0$ and $MSKIP = 1$. Increasing $QSKIP$ and $MSKIP$ can improve the recall in case that the format of the data fields specific enough; e.g. the e-mail address can be discovered by its format only so that it is not necessary to require any label. Generally, increasing these values causes a significant loss of precision.

7.1.2 Experiment 2 - Stock Quotes

As a second experiment we have extracted some quote data from publicly available quote servers. The template tree with the regular expressions is given in the figure 7.3. The task consisted of extracting the last price, the change of the last price, the opening price, the last bid and the traded volume. Examples of some of the documents are shown in the figure 7.4.

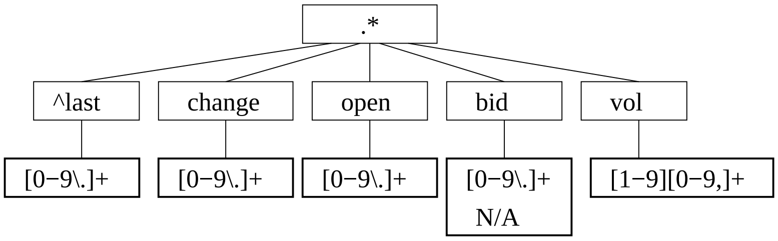
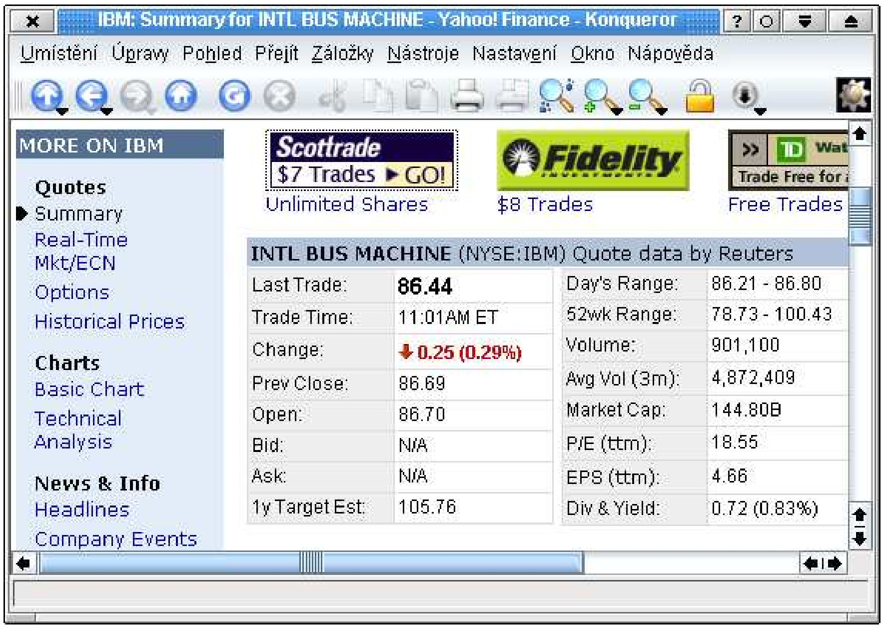


Figure 7.3: Template tree for extracting the quote data

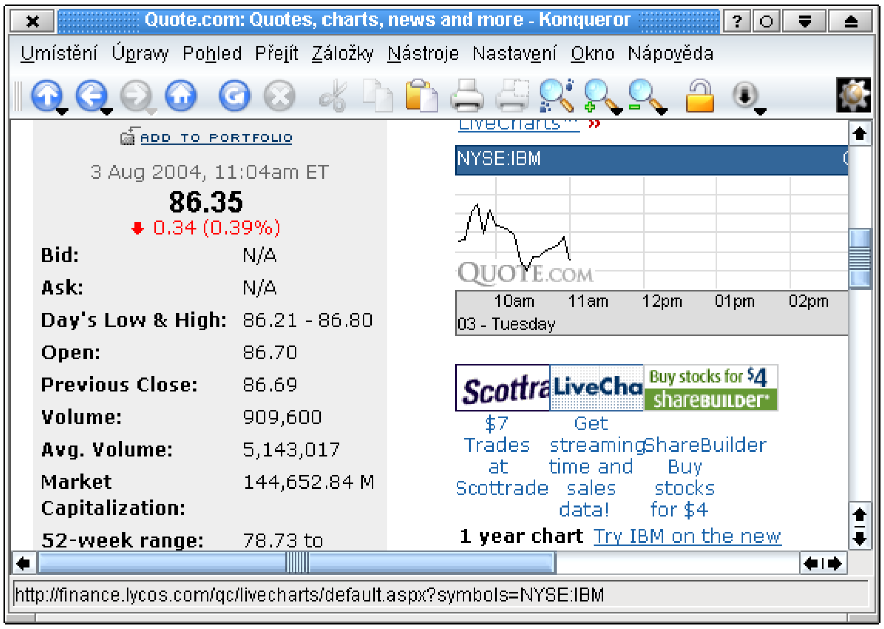
The results are summarized in the table 7.2. The testing sets 2 and 6 both use tables for presenting the data in quite complicated way so that the logical structure of the table is not detected properly. It would be necessary to further improve the algorithms for the table analysis in order to obtain the proper results. The lower recall for the testing sets 3, 4 and 5 has the usual reason – the data is not labeled as expected by the extraction task specification. The solution is to create more variants of the expected logical structure as discussed in section 5.6.1.

7.2 Independence on Physical Realization

In the figure 7.5 we can see a comparison of the web directories of two different universities that correspond to the `yale.edu` and `cornell.edu` testing sets. The data from both these sites have been correctly extracted using a single template tree definition although each of them uses a distinct way of data presentation and the HTML codes



`finance.yahoo.com` quote server



`quote.com` server

Figure 7.4: The designs of two different quote servers

#	URL	Precision %	Recall %
1	`finance.yahoo.com`	100	100
2	`dbc.com`	0	0
3	`quote.com`	100	63
4	`pcquote.com`	100	59
5	`money.cnn.com`	100	44
6	`uk.finance.yahoo.com`	0	0

Table 7.2: Sample extraction task results

of both documents are significantly different. Thus, the proposed extraction method is to a great extent independent on the physical realization of the HTML documents.

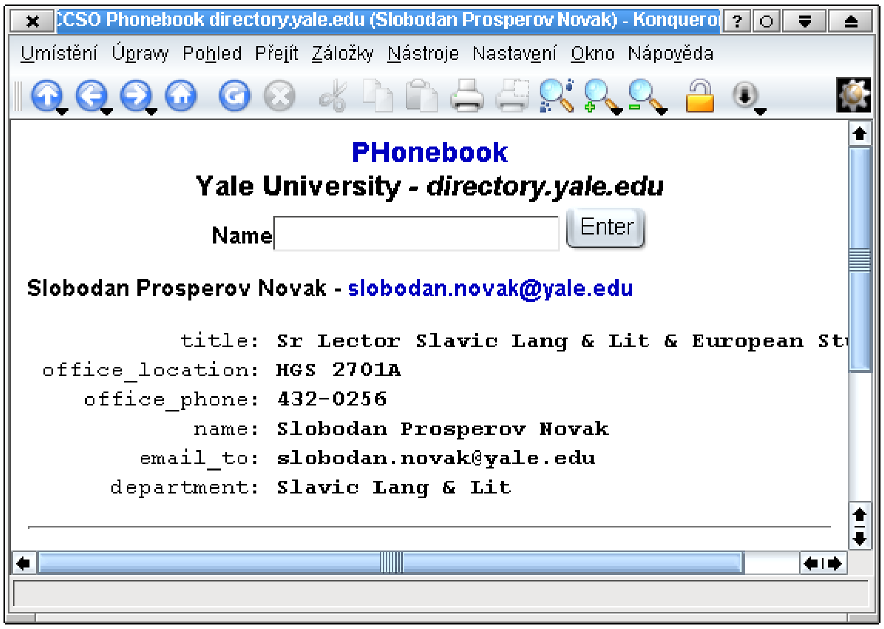
However, the method still depends on the way of the data presentation; i.e. on the logical structure of the documents. Considering the experiments described in the previous section, this difference causes the necessity of two variants of the template tree in the "personal data" experiment and, when only a single template tree is used, it causes lower recall of the results as show the results of the second, "stock quote" experiment. The difference between the logical structures of the documents is apparent from a visual comparison of the samples. Let's compare the examples of the variant A of the first experiment that are shown in the figure 7.5 with the example of the variant B that is in the figure 7.6. We can notice that in the former case, the name of the person is presented as being superior to the remaining data (it is used as a title) whereas in the later case, the name of the person is listed in the same way as the remaining data.

The results show that in the proposed method, the independence on the HTML code is partially replaced by the dependence on the logical document structure. However, comparing with the number of various wrappers that would be necessary for extracting the same data from the listed web sites (typically, one for each site), our method brings a significant improvement.

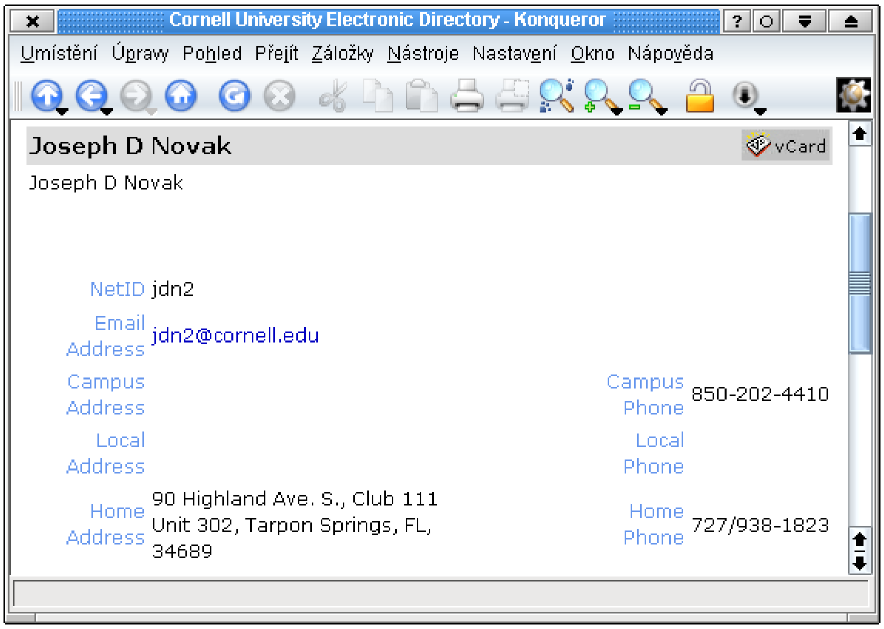
7.3 Information Extraction from Logical Documents

The proposed method for extracting data from logical documents has been developed for processing the logical documents that consist of static web pages. However, in the real World Wide Web, it is not easy to find static logical documents containing large amounts of data. In both cases – the university staff directories and the quote servers, it is impossible to present all the data in static document due to its great amount. Instead, the data is stored in a database and the documents are generated dynamically upon a user query. For example, the staff directories typically require entering at least the last name of a person and return a document containing the data of all the staff members of that last name. Therefore, the entire directory content cannot be accessed directly through the web interface. At this point, we face the problem of the hidden web as mentioned in section 2.3.

For the mentioned reasons, we have used following method for obtaining the logical



Yale University Directory



Cornell University Directory

Figure 7.5: Different designs of a university directory (variant A)

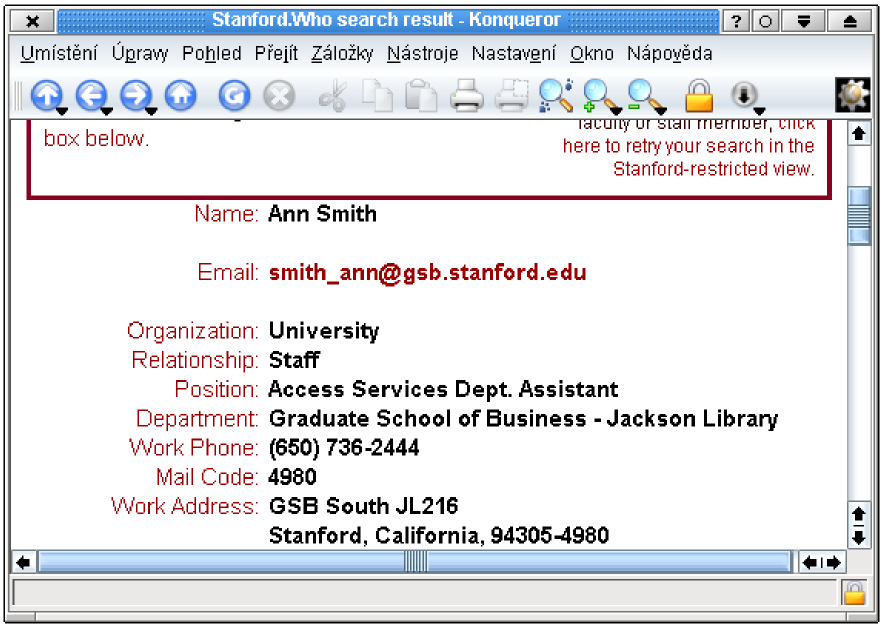


Figure 7.6: An example of the variant B – Stanford University Directory

documents from the staff directories in order to evaluate the proposed method.

We have chosen some frequent last names, namely "Novak", "Dvorak", "Smith" and "Johnson". With the standard web browser we used these names for querying the web directories of the above listed universities. In all the tested directories, the query results in a dynamic document, that contains the list of matching staff members and the searched name is contained in the URI of this document. For example, the list of all the staff members of Stanford University with the name containing "Novak" is available in a document with the following URI:

`https://stanfordwho.stanford.edu/lookup?search=Novak&submit=Search`

This example shows an URI of a dynamically generated document with two arguments: `search` and `submit`. From this point, no user queries are necessary and we can regard this document as the main page of a logical document that contains the data of all the listed persons that match the query. Note that this procedure is not required by the proposed information extraction method as such; it is only a way of obtaining a sufficient amount of logical documents for the method evaluation.

Once we have determined the URI of the main page of the logical document, we can run the information extraction task on this URI. Since this logical document contains the personal pages of all the matching persons and the logical structures of the individual personal pages will appear as subtrees in the logical structure tree of the whole logical document, the results of infirmation extraction should be the same as if the individual personal pages of the listed persons were processed individually. This fact has been confirmed by the practical tests – the information extraction results correspond to the results for the individual physical documents that have been processed during the tests described in section 7.1.1 and that are summarized in the table 7.1. The used logical document discovery algorithm (section 5.7) however tends to include more additional pages to the logical document that do not influence the information extraction result but the download and processing of these excessive pages is time-consuming. For example, in case of the Harvard University, 62 documents have been downloaded during the logical document discovery whereas the data to be extracted was contained in 4 of them.

In case of the quote servers, no summary pages are available that could be used as the main pages of some logical documents. For obtaining the data, the quote symbols must be entered exactly and therefore, the generated documents must be analyzed separately.

Chapter 8

Conclusions

Proposed method of information extraction is usable for real data-intensive documents available through the World Wide Web. With data-intensive documents we mean the documents that are primarily intended for presenting data in a relatively regular and structured form where the visual information plays an important role for the readers' understanding of the document. These documents often contain up-to-date data that are worth extracting and typically, the documents are automatically generated from a back end database. On the other hand, the method is not suitable for processing the documents where the desired information is buried in large blocks of unformatted or poorly formatted text. For such documents, the traditional text document processing and natural language processing methods are more applicable.

There is one more important issue in information extraction as such that has not a technical nature. Quite often, the provider of the information that is presenting it on the web prefers browsing the documents by people rather than the automatic tools, mainly for marketing reasons (advertisement etc.). Such subjects often use various techniques that complicate automatic processing of the documents such as detection and blocking of the client or "hiding" the information in the document. There are, however, still many areas where the information extraction is justifiable and useful.

8.1 Summary of Contributions

Proposed method presents a novel approach to information extraction from HTML documents. In contrast to current methods based mainly on the direct analysis of the HTML code, our method has following important features:

- *Independence on the underlying HTML code of the document.* The document is described by a abstract model, which is then used for extracting information. This abstraction avoids the dependence on particular HTML tags, which is the bottleneck of the wrapper approach.

- *Resistance to the changes of documents.* The use of abstract model ensures that the method is resistant to changes in the data presentation in the document unless the logical structure of the document changes.

- *No training phase required.* The information extraction process can start as soon as the extraction task specification is finished. There is no training set of example documents needed. The method allows processing new, previously unknown documents that correspond to the extraction task specification.

Aside from this main contribution, there are some issues in the method proposal that we consider a significant or novel contributions:

- *Formal models of the visual information in the document.* To the author's best knowledge, this is a **first attempt to formally describe the information that is given to the user by visual means**. We propose formal models of two components of this information – the page layout and the visual attributes of the text.

- *Modeling the logical structure on the basis of the visual model.* This approach is unique for the processing of HTML documents although similar idea has been proposed for other types of documents. The model of the logical structure presents an important information about the document that can be used (aside from information extraction) in many other areas such as *information retrieval* (searching documents based on structured queries instead of single keywords) or *alternative document presentation* (e.g. structure-aware voice readers for blind people).

- *Application of the tree matching algorithms.* Although the hierarchical organization of HTML documents is commonly accepted, the use of tree matching algorithms for this task can be considered a novel contribution to this area.

Finally, the contribution of the thesis is an experimental information extraction system that implements the proposed techniques. This system has been implemented in the Java environment and has been used for verifying the method in the real world.

8.2 Possible Improvements and Future Work

From the point of view of further improvements of the method, the most important point seems to be the proposed algorithm for analyzing the logical structure of HTML tables. Although these algorithm works satisfactorily for most of the documents, there exist more complex ways of presenting data in a table that are not recognized properly as results for example from the test results given in the section 7.1.2. For being able to process a larger set of possible variants, a more sophisticated analysis method should be developed.

The second issue is the way of using the logical document structure for information extraction. In our method, we use the tree matching algorithms and we avoid any machine learning phase. However, for some applications, it would be interesting to use some machine learning algorithms for inferring the extraction task specification automatically.

Bibliography

[1] Adelberg, B. NoDoSe – A Tool for Semi-Automatically Extracting Structured and Semistructured Data from Text Documents. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data* Seattle, Washington, United States, 1998

[2] Anjewierden, A. AIDAS: Incremental Logical Structure Discovery in PDF Documents In *6th International Conference on Document Analysis and Recognition (ICDAR).* Seattle, USA, 2001

[3] Ashish, N., Knoblock, C. Wrapper Generation for Semi-structured Internet Sources. In *Workshop on Management of Semistructured Data.* Tucson, Arizona, 1997

[4] Atzeni, P., Mecca, G., Merialdo, P. Semistructured and Structured Data in the Web: Going Back and Forth. In *Proceedings of ACM SIGMOD Workshop on Management of Semi-structured Data.* 1997

[5] Baumgartner, R., Flesca, S., Gottlob, G. Visual Web Information Extraction with Lixto. In *Proceedings of the 27th International Conference on Very Large Data Bases.* Roma, Italy, 2001

[6] Berners-Lee, T. The Semantic Web. *Scientific American.* May 2001

[7] Bos, B., Lie, H.W., Lilley, C., Jacobs, I. (editors). *Cascading Style Sheets, level 2, CSS2 Specification.* W3C Recommendation 12 May 1998. `http://www.w3.org/TR/1998/REC-CSS2-19980512`

[8] Burget, R. Analyzing Logical Structure of a Web Site. In *Proceedings of 5th International Conference ISM '02 - Information Systems Modelling.* Ostrava, CZ, MARQ, 2002, p. 29-35, ISBN 80-85988-70-4

[9] Burget, R. HTML Document Analysis for Information Extraction. In *Proceedings of 8th EEICT conference.* Brno, CZ, FIT VUT, 2002, p. 426-430, ISBN 80-214-2116-9

[10] Burget, R. Information Extraction from WWW Based on the Data Structure Knowledge (in czech language). In *Proceedings of the 2nd conference Znalosti 2003.* Ostrava, CZ, FEI VSB, 2003, p. 271-280, ISBN 80-248-0229-5

[11] Burget, R. Hierarchies in HTML Documents: Linking Text to Concepts. Accepted for *3rd International Workshop on Web Semantics - WebS '04.* Zaragoza, Spain, 2004

[12] Buttler, D., Liu, L., Pu, C. A Fully Automated Object Extraction System for the World Wide Web. In *Proc. of IEEE International Conference on Distributed Computing Systems.* 2001

[13] Carchiolo, V., Longheu, A., Malgeri, M. Extracting Logical Schema from the Web", In *PRICAI Workshop on Text and Web Mining.* Melbourne, Australia, 2000

[14] Chung, C.Y., Gertz. M., Sundaresan, N. Reverse Engineering for Web Data: From Visual to Semantic Structures. In *18th International Conference on Data Engineering (ICDE 2002).* IEEE Computer Society, 2002.

[15] Ciravegna, F. $(LP)^2$, an Adaptive Algorithm for Information Extraction from Web-related Texts. In *Proceedings of the IJCAI-2001 Workshop on Adaptive Text Extraction and Mining.* Seattle, USA, 2001

[16] Clark, P., Niblett, T. The CN2 Induction Algorithm. *Machine Learning*, 3:261-283, 1989

[17] Cohen, W.W., Hurst, M., Jensen, L.S. A Flexible Learning System for Wrapping Tables and Lists in HTML Documents. In *Proceedings of the Eleventh International World Wide Web Conference.* Honolulu, Hawaii, USA, 2002

[18] Crescenzi, V., Mecca, G., Merialdo, P. *RoadRunner: Towards automatic data extraction from large web sites.* Technical Report n. RT-DIA-64-2001, D.I.A. Università di Roma Tre, 2001

[19] Crescenzi, V., Mecca, G., Merialdo, P. Automatic Web information Extraction in the RoadRunner System. In *International Workshop on Data Semantics in Web Information Systems.* Yokohama, Japan, 2001

[20] Deogun, J.S., Sever, H., Raghavan, V.V. Structural Abstractions of Hypertext Documents for Web-based Retrieval In *Proceedings of DEXA 98 - 9th International Conference on Database and Expert Systems Applications.* Vienna, Austria, 1998

[21] DiPasquo, D. *Using HTML Formatting to Aid in Natural Language Processing on the World Wide Web.* School of Computer Science, Carnegie Mellon University, Pittsburgh, 1998

[22] Embley, D.W., Campbell, D.M., Jiang, Y.S., Ng, Y.-K., Smith, R.D., Liddle, S.W., Quass, D.W. A conceptual-modeling approach to extracting data from the web. In *Proc. of the 17th International Conference on Conceptual Modeling (ER'98).* Singapore, 1998

[23] Embley, D.W., Jiang, Y.S., Ng, Y.-K. Record-boundary discovery in Web documents. In *Proc. of the 1999 ACM SIGMOD International Conference on Management of Data* 1999

[24] Fielding, R., et al. *Hypertext Transfer Protocol – HTTP/1.1.* RFC2616, The Computer Society, 1999. `http://rfc.net/rfc2616.html`

[25] Freitag, D. Using Grammatical Inference to Improve Precision in Information Extraction. In *ICML-97 Workshop on Automata Induction, Grammatical Inference, and Language Acquisition.* 1997

[26] Freitag, D. Information extraction from HTML: Application of a general learning approach. In *Proc. of the Fifteenth Conference on Artificial Intelligence AAAI-98.* 1998

[27] Freitag, D., McCallum, A. Information Extraction with HMMs and Shrinkage. In *Proceedings of the AAAI-99 Workshop on Machine Learning for Information Extraction.* 1999

[28] Fuhr, N., Grossjohann, K. XIRQL: An XML Query Language Based on Information Retrieval Concepts. In *Proc. of the 24th annual international ACM SIGIR conference on Research and development in information retrieval.* New Orleans, USA, 2001

[29] Gibson, D., Kleinberg, J., Raghavan, P. Infering Web Communities from Link Topology. In *Proc. of 6th Intl. Conference on Database Systems for Advanced Applications (DASFAA'99).* IEEE Computer Society, 1999

[30] Gold, E.M. Language Identification in the Limit. *Information and Control*, 10(5):447-474. 1967

[31] Grishman, R., Sundheim, B. Message Understanding Conference – 6: A Brief History. In *Proceedings of the 16th International Conference on Computational Linguistics.* Copenhagen, Denmark, 1996

[32] Gu, X.-D., Chen, J., Ma, W.-Y., Chen, G.-L. Visual Based Content Understanding towards Web Adaptation. In *Proc. Adaptive Hypermedia and Adaptive Web-Based Systems.* Malaga, Spain, 2002, pp. 164-173

[33] Guan, T., Wong, K.F. KPS – a Web Information Mining Algorithm. In *The 8th International World Wide Web Conference.* Toronto, Canada, 1999

[34] Güttner, J. Object Database on Top of the Semantic Web. In *Proceedings of the WI/IAT 2003 Workshop on Applications, Products and Services of Web-based Support systems.* Halifax, CA, 2003, pp. 97-102

[35] Hong, T.W., Clark, K.L. Using Grammatical Inference to Automate Information Exraction from the Web. In *Principles of Data Mining and Knowledge Discovery.* 2001

[36] Kan, M.-Y. *Combining visual layout and lexical cohesion features for text segmentation.* Columbia University Computer Science Technical Report, CUCS-002-01. 2001

[37] Kosala, R., Van den Bussche, J., Bruynooghe, M., Blockeel, H. Information Extraction in Structured Documents using Tree Automata Induction, In *Principles of Data Mining and Knowledge Discovery, Proceedings of the 6th International Conference (PKDD-2002).* 2002

[38] Kushmerick, N., Weld, D.S., Doorenbos, R.B. Wrapper Induction for Information Extraction, In *International Joint Conference on Artificial Intelligence.* 1997

[39] Kushmerick, N. Wrapper Induction: Efficiency and Expressiveness. *Arfiticial Intelligence* vol. 118, no. 1-2, pp. 15-68, 2000

[40] Kushmerick, N. Wrapper Verification. *World Wide Web Journal* vol. 3, no. 2, pp. 79-94, 2000

[41] Kushmerick, N., Thomas, B. Adaptive information extraction: Core technologies for information agents In *Intelligent Information Agents R&D in Europe: An AgentLink perspective* Lecture Notes in Computer Science 2586, Springer 2002.

[42] Lin, S.-H., Ho, J.-M. Discovering Informative Content Blocks from Web Documents. In *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD'02).* 2002

[43] Liu, B., Grossman, R., Zhai, Y. Mining Data Records in Web Pages In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003).* Washington, DC, USA, 2003

[44] Manola, F., Miller, E. (editors) *RDF Primer.* W3C Working Draft 11 November 2002
`http://www.w3.org/TR/2002/WD-rdf-primer-20021111/`

[45] May, W. Modeling and Querying Structure and Contents of the Web. In *Proceedings of the 10th International Workshop on Database and Expert Systems Applications* Florence, Italy, 1999

[46] Mello, R., Heuser, C.A. A Bottom-Up approach for Integration of XML Sources In *International Workshop on Information Integration on the Web.* Rio de Janeiro, Brasil, 2001

[47] Morkes, J., Nielsen, J. *Concise, SCANNABLE, and Objective: How to Write for the Web,* 1997
`http://www.useit.com/papers/webwriting/writing.html`

[48] Mukherjee, S., Yang, G., Tan, W., Ramakrishnan, I.V. Automatic discovery of Semantic Structures in HTML Documents. In *International Conference on Document Analysis and Recognition (ICDAR).* 2003

[49] Muslea, I., Minton, S., Knoblock, C.A. Hierarchical Wrapper Induction for Semistructured Information Sources. *Journal of Autonomous Agents and Multi-Agent Systems* 4:93-114, 2001

[50] Nahm, U.Y., Mooney, R.J. Text Mining with Information Extraction. In *Proceedings of the AAAI 2002 Spring Symposium on Mining Answers from Texts and Knowledge Bases.* 2002.

[51] Nelson, T.H. Complex information processing: a file structure for the complex, the changing and the indeterminate. In *ACM/CSC-ER, Proceedings of the 1965 20th national conference.* Cleveland, USA, 1965

[52] Niu, C., Li, W., Ding., J., Srihari, R. A Bootstrapping Approach to Named Entity Classification Using Successive Learners. In *Proceedings of the ACL-2003 Conference.* Sapporo, Japan, 2003

[53] Quinlan, J.R., Cameron-Jones, R.M. FOIL: A Midterm Report, *Machine Learning: ECML-93*, Vienna, Austria, 1993

[54] Raggett, D., Le Hors, A., Jacobs, I. (editors). *HTML 4.01 Specification.* W3C Recommendation 24 December 1999. http://www.w3.org/TR/1999/REC-html401-19991224

[55] Salton, G. Recent Studies in Automatic Text Analysis and Document Retrieval. *JACM*, 20(2):258-278, Apr. 1973

[56] Shasha, D., Wang, J.T.L, Shan, H., Zhang, K. ATreeGrep: Approximate Searching in Unordered Trees. In *14th International Conference on Scientific and Statistical Database Management.* Edinburgh, Scotland, 2002

[57] Soderland, S. Learning to Extract Text-based Information from the World Wide Web, In *Proceedings of Third International Conference on Knowledge Discovery and Data Mining (KDD-97).* 1997

[58] Summers, K. Toward a Taxonomy of Logical Document Structures. In *Electronic Publishing and the Information Superhighway: Proceedings of the Dartmouth Institute for Advanced Graduate Studies (DAGS '95).* Boston, USA, 1995, pp. 124-133

[59] Summers, K. Automatic Discovery of Logical Document Structure. *PhD thesis.* Cornell Computer Science Department Technical Report TR98-1698, 1998

[60] Tajima, K., Tanaka, K. New Techniques for the Discovery of Logical Documents in Web. In *International Symposium on Database Applications in Non-Traditional Environments (DANTE'99).* 1999

[61] World Wide Web Consortium (W3C) pages. http://www.w3.org/

[62] Yang, Y., Zhang, H. HTML Page Analysis Based on Visual Cues. In *Proc. of 6th International Conference on Document and Analysis.* Seattle, USA, 2001

[63] Zizi, M., Lafon, M. Hypermedia exploration with interactive dynamic maps. *International Journal on Human Computer Interaction Studies.* 1995

Appendix A

Example Task Specification

This example shows an XML specification of two information extraction tasks described in the section 7.1.

Experiment 1 - Personal Information

Variant A

```
<?xml version="1.0" encoding="iso-8859-2"?>
<task name="staff">
    <!-- Input field spec and path specification -->
    <input>
        <spec case="lower" name="namelabel">name</spec>
        <spec case="sensitive"
              name="nameval">^[A-Z][A-Za-z,\.\ ]+$</spec>

        <spec case="lower" name="deplabel">department</spec>
        <spec case="lower" name="dptlabel">dept</spec>
        <spec case="lower" name="depval">^[a-z0-9\ $]+</spec>

        <spec case="lower" name="maillabel">e-?mail</spec>
        <spec case="lower"
              name="mailval">^[a-z0-9_\.]+@[a-z0-9_\.]+</spec>
    </input>

    <!-- Inspecion model (field hierarchy) -->
    <model name="person">
        <data spec="nameval" name="name">
            <label spec="deplabel|dptlabel">
                <data spec="depval" name="department"/>
            </label>
            <label spec="maillabel">
                <data spec="mailval" name="email"/>
```

```
            </label>
        </data>
    </model>
</task>
```

Variant B

```
<?xml version="1.0" encoding="iso-8859-2"?>
<task name="staff">
    <!-- Input field spec and path specification -->
    <input>
        <spec case="lower" name="namelabel">name</spec>
        <spec case="sensitive"
              name="nameval">^[A-Z][A-Za-z,\.\ ]+$</spec>

        <spec case="lower" name="deplabel">department</spec>
        <spec case="lower" name="dptlabel">dept</spec>
        <spec case="lower" name="depval">^[a-z0-9\ $]+</spec>

        <spec case="lower" name="maillabel">e-?mail</spec>
        <spec case="lower"
              name="mailval">^[a-z0-9_\.]+@[a-z0-9_\.]+</spec>
    </input>

    <!-- Inspecion model (field hierarchy) -->
    <model name="person">
            <label spec="namelabel">
                <data spec="nameval" name="name"/>
            </label>
            <label spec="maillabel">
                <data spec="mailval" name="email"/>
            </label>
            <label spec="deplabel|dptlabel">
                <data spec="depval" name="department"/>
            </label>
    </model>
</task>
```

Experiment 2 - Stock Quotes

```
<?xml version="1.0" encoding="iso-8859-2"?>
<task name="quotes">
    <!-- Input field spec and path specification -->
    <input>
        <spec case="lower" name="float">[0-9\.]+</spec>
```

```
        <spec case="lower" name="int">[1-9][0-9,]+</spec>
        <spec case="lower" name="na">N/A</spec>

        <spec case="lower" name="lastlabel">^last</spec>
        <spec case="lower" name="chglabel">change</spec>
        <spec case="lower" name="openlabel">^open</spec>
        <spec case="lower" name="bidlabel">bid</spec>
        <spec case="lower" name="vollabel">vol</spec>
    </input>

    <!-- Inspecion model (field hierarchy) -->
    <model name="quote">
        <label spec="lastlabel">
            <data spec="float" name="last"/>
        </label>
        <label spec="chglabel">
            <data spec="float" name="change"/>
        </label>
        <label spec="openlabel">
            <data spec="float" name="open"/>
        </label>
        <label spec="bidlabel">
            <data spec="na" name="bid"/>
        </label>
        <label spec="vollabel">
            <data spec="int" name="volume"/>
        </label>
    </model>

</task>
```

Appendix B

Document Type Definitions

In following sections we give the formal Document Type Definitions (DTD) for all the XML document formats used in our experimental information extraction system.

B.1 Task Specification

Following DTD defines the XML document format of the extraction task specification. An example of such specification is given in Appendix A. It allows to specify the formats of all fields by regular expressions in the `<input>` section and to define the template tree in the `<model>` section. This tree consists of the `<label>` and `<data>` nodes. Both these tags specify an expected format of a logical structure tree node by using one or more format specification from the `<input>` section. The difference between label and data is that the label is just matched to a structure tree node whereas the data forms part of the extracted output data.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- The root element -->
<!ELEMENT task (input, model)>
<!ATTLIST task
          name CDATA #REQUIRED>

<!-- Field format specifications -->
<!ELEMENT input (spec*)>

<!-- Specification of a named field format. The #PCDATA contains
    a regular expression. The comparison may be either case
    sensitive or lowercase. -->
<!ELEMENT spec (#PCDATA)>
<!ATTLIST spec
          name CDATA #REQUIRED
          case (lower|sensitive) "lower">
```

```
<!-- Template tree specification -->
<!-- A hierarchy of labels (not included in the output) and data
    (included in the output). The spec attribute must correspond
    to a name of any <spec> element above, logical disjunction
    can be denoted by "name1|name2". -->
<!ELEMENT model (label*, data*)>
<!ATTLIST model
          name CDATA #REQUIRED>

<!ELEMENT label (label*, data*)>
<!ATTLIST label
          name CDATA ""
          spec CDATA "">

<!ELEMENT data (label*, data*)>
<!ATTLIST label
          name CDATA #REQUIRED
          spec CDATA "">

<!-- End of the DTD -->
```

B.2 Logical Document Representation

This DTD defines the XML document format for representing the set of URIs that form a logical document. The <logical_document> element represents the whole logical document with the URI of the main page, the contained <document> elements contain the URIs of the physical documents an example of the document is shown in section 6.2 on page 58.

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT logical_document (document+)>
<!ATTLIST logical_document
          url  CDATA  #REQUIRED>

<!ELEMENT document EMPTY>
<!ATTLIST document
          url  CDATA  #REQUIRED>

<!-- End of the DTD -->
```

B.3 Logical Structure Representation

This is a Document Type Definition for the XML representation of the resulting logical document structure model that is the product of the logical structure analysis. The format of this file is described in detail in section 6.2.2 on page 58.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- The root element, structure of a logical document -->
<!ELEMENT document (text*)>
<!ATTLIST document
          title CDATA ""
          url   CDATA "">

<!-- A node of the logical structure tree -->
<!ELEMENT text (text*)>
<!ATTLIST text
          content CDATA ""
          visual  CDATA #REQUIRED
          expr    CDATA #REQUIRED
          weight  CDATA #REQUIRED>

<!-- End of the DTD -->
```