

# Web Structure Analysis for Information Mining

Lakshmi Vijjappu<sup>1</sup>, Ah-Hwee Tan<sup>2</sup>, and Chew-Lim Tan<sup>1</sup>

<sup>1</sup> School of Computing, National University of Singapore, 10 Kent Ridge Crescent,  
Singapore 119260

vijjappu,tancl@comp.nus.edu.sg

<sup>2</sup> Kent Ridge Digital Labs, 21 Heng Mui Keng Terrace, Singapore 119613, ahhwee@krdl.org.sg

## Abstract

*Our approach to extracting information from the web analyzes the structural content of web pages through exploiting the latent information given by HTML tags. For each specific extraction task, an object model is created consisting of the salient fields to be extracted and the corresponding extraction rules based on a library of HTML parsing functions. We derive extraction rules for both single-slot and multiple-slot extraction tasks which we illustrate through two sample domains.*

## 1. Introduction

Information extraction (IE) as defined by Message Understanding Conferences (MUC) refers to the task of, given a document, automatically finding the essential details in the text. For example, given a web page on a seminar announcement, an IE system would extract salient information such as topic, speaker, date, venue, and abstract. A key element of IE systems is the set of text extraction rules that identify the relevant information to be extracted.

IE systems have been built with different levels of success on several kinds of text domains. CRYSTAL [1] was a learning-based IE system that took parsed annotated sentences and found patterns for extraction in novel sentences. Webfoot [2] was an attempt at general IE that processed fragments by looking at Hyperlink Markup Language (HTML) tags. SRV [3] was another learning architecture for IE. It took a user-defined feature set together with a set of hand tagged training documents and learned rules for extraction. Craven *et al.* [4] reported that greater accuracy could be achieved by representing each web page as a node in graph and each hyperlink an edge. Cardie [5] provided a list of learning-based IE problems, including the difficulty of obtaining enough training data and the lack of corpora annotated with the appropriate semantic and domain-specific supervisory information. The generation of

training examples is complicated because the IE process often requires the output of earlier levels of analysis such as tagging and partial parsing. DiPasquo [6] argued that there was inherent information in the layout of each page. Hannes and Tom [7] describe WebL, and then show its usage by implementing a meta-search engine that combines search results from the AltaVista and HotBot public-search services. WebL is a high level, object-oriented scripting language that incorporates two novel features: service combinators and a markup algebra. The markup algebra extracts structured and unstructured values from pages for computation, and is based on algebraic operations on sets of markup elements. Un Yong [8] describes a system called DiscoTEX that combines IE and KDD methods to perform a text-mining task, discovering prediction rules from natural-language corpora. Hence by parsing the HTML formatting, one can improve upon traditional text processing.

Recently, the use of wrappers for IE from the web has been popular. A typical wrapper application extracts the data from Web pages that are generated, based on predefined HTML templates. The systems generate delimiter-based rules that use linguistic constraints. Wien [9] uses only delimiters that immediately precede and follow the actual data. SoftMealy [10] is a wrapper induction algorithm that generates extraction rules expressed as finite-state transducers. World Wide Web Wrapper Factory [11] does extraction by using an HTML parser to construct a parse tree following a Document Object Model. In general, HTML tags can help in many tasks involving natural language processing on the web. In this paper, we consider the more specific problem of exploiting HTML tags for IE from the Web. The motivation of the work stems from a need for a simpler tool to facilitate the writing of extraction rules for our own applications.

We adopt an object model approach to extracting information from HTML pages. An object model, consisting of the salient fields of the web pages and their extraction rules based on an HTML parsing library, represents a projection of a user's interests

and requirement on a group of web pages. We derive object models for both single-slot extraction, wherein the extracted fields are independent; and multiple-slot extraction, wherein the extracted field are related. We present object models and experimental results for two sample domains, namely news extraction model based on single-slot extraction rules and link extraction based on multiple-slot extraction rules.

## 2. Object Model and Extraction Rules

As depicted in figure 1, the proposed system comprises the following main components.

- HTML Parsing Library:** A library of HTML parsing functions serves as the basic building block of writing object models. Functions are provided to manipulate attributes of HTML tags, including text, tables, and links. A variety of text extraction tasks exist ranging from extracting the contact email address, the list of products to complex sequential pattern from the page. A text extraction task could thus consist of identifying certain text fragments based on single criteria or multiple criteria or continuously check for patterns. Hence based on the nature of extraction, IE systems are characterized as single-slot or multi-slot. Single-slot locates and extracts isolated texts from the text; multi-slot locates a sequential pattern and recursively searches for similar patterns in the web page. Hence parsing

functions have been drafted to accommodate single-slot as well as multi-slot/pattern functions. A subset of the single-slot functions is listed in Table 1. Multi-slot functions would be prefixed by *pat\_*. A simple user-interface for writing extraction rules has been developed.

- Object Model:** The library of basic HTML parsing functions can be used to specify what's interesting or target object of the user and extract the portions he/she is interested in. Some general functions could be devised which can be used across web pages in the same category. For each task, a set of functions can be formulated and this encapsulation of items of interests and extraction algorithm forms an object model. For e.g., an object model to extract a stock quote from a web page could be written and this object model could then be applied to any web site offering stock quotes.
- Extraction Engine:** An extraction engine, central to this architecture, is basically a compiler to the HTML parsing functions. A user specifies the URL of the web page and the object model, i.e., the fields of interests and a set of rules written using the HTML parsing functions to extract them. The extraction engine parses the extraction rules and produces the results in XML format.

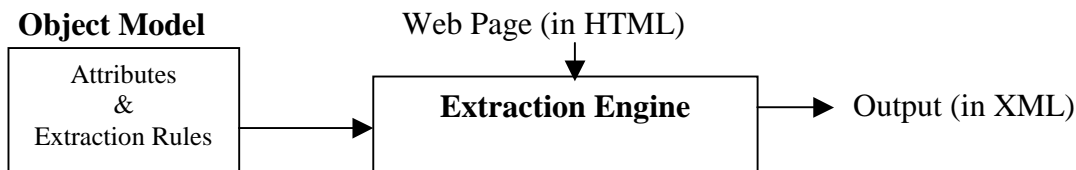


Figure1: The extraction engine produces the key information in the form of XML from HTML web pages according to the extraction rules of an object model.

Table 1: An illustrative list of HTML parsing functions.

HTML tag	Function	Comments
Tag Contents	<i>getTagContent (tag, n)</i>	Extracts the $n^{\text{th}}$ occurrence of the content enclosed within the start and end <i>tag</i> . E.g., <i>getTagContent (&lt;B&gt;, 2)</i> .
Tag Attribute	<i>getAttributeValue(tag, att, n)</i>	Retrieves the attribute <i>att</i> of the $n^{\text{th}}$ occurrence of the tag <i>tag</i> . E.g., <i>getTagAttribute (&lt;a&gt;, href, n)</i> .
Tag Position	<i>getTagPostion (tag, n)</i>	Returns the tag position or the line number of the $n^{\text{th}}$ occurrence of the tag from the start of the page. E.g., <i>getTagPosition (&lt;a&gt;, 3)</i> .
TagText	<i>getText (term, n)</i>	Locates the $n^{\text{th}}$ occurrence of term <i>term</i> in the page. E.g., <i>getText ("College", 1)</i> .
Table	<i>getTableData (col, m, term, n)</i>	Returns "m" elements from the column <i>term</i> from the $n^{\text{th}}$ table E.g., <i>getTableData (row, 1, "exchange")</i> .

### 3. News Article Extraction

An object model was manually derived by referring to news articles obtained from ZDnet and CNet. The news extraction model (Figure 2) is described by a set of 4 variables using the library of HTML parsing functions. The title of the article is normally found in the <meta> tag of the web page and its location indicates the start of the article. The next few lines would contain the author's name and the date of the article. The author's name is generally prefixed by the keyword "by". The dateline is in any of the date formats such as *mm-dd-yyyy* or *month-dd-year*. The content of the article, which follows, is normally enclosed in the html <p> or <span> tag. This object model is then used to see how it may extract the needed information from about 100 news articles from ZDnet and CNet. The results are tabulated in terms of *precision* and *recall* in Table 2.

**Table 2: News extraction performance for ZDNet and CNet.**

Section	Recall	Precision
Title	98%	100%
Author	90%	100%
Dateline	93%	100%
Text	100%	90%

The program had 100% precision in finding the title, dateline and author. When extracting the author, the recall dropped due to the fact that some articles did not have the keyword "By". Initially the recall for the dateline was 80%. The drop in the dateline was due to the fact that the author and the date were found in the same line and the program would extract the whole line as the author's name. So in cases where it did find an author and not a date, we searched for date patterns in the author which significantly improved the recall to 93%. All the contents of the article were retrieved but in most of the articles, some extra lines towards the end like "More news articles", "Did you miss the news for a day?" "etc. was also

```

$TITLE = getTagContent (<meta>,"name=title","content",0) || getTagText (<title>,$tag_pos)
$attribute_line_no = getTagTextPosition (<span>,"class=smhead",0)
$AUTHOR = searchKeyWord ($attribute_line_no,"by",-1)
$DATE = getformat (DATE,"content start",$attribute_line_no,10)
$TEXT = getParaText () || getTagText (<span>,"class=body",0)

```

**Figure 2: An object model for news extraction.**

retrieved. This object model when applied to articles from Straits Times Interactive (STI) produced a similar level of performance as shown in Table 3.

**Table 3: News extraction performance for STI news articles.**

Section	Recall	Precision
Title	98%	100%
Author	90%	100%
Dateline	93%	100%
Text	98%	89%

### 4. Link Extraction

A typical web page returned by a search engine consists of hyperlinks to the retrieved query. Each link is characterized by an absolute URL, a caption (hyperlinked text), and a summary describing it. The HTML anchor tag <a href=""> is used to create hyperlinks. To extract the above pattern, an object model (Figure 3) is derived based on multi-slot HTML parsing functions. The initial position of a potential pattern would be any hyperlink text obtained through *pat\_getHyperLinkPosition*, which contains the query terms. The position is then passed to the other functions as an anchor for parsing. To extract the URL, the parsing function *getAttributeValue* is used. Summary is the text between two consecutive hyperlinks obtained by the function *pat\_getTagToTag*.

The search pages may also contain hyperlinks to the engine's categories, shopping links, and advertisements etc. These can be eliminated by the fact that they normally do not have a summary or a text describing them and also the Caption would not contain the query terms. The extraction rules also filter links to categories in the engine by identifying relative URL's. In addition, image hyperlinks are ignored.

```

<PATTERN>
  $pos = pat_getHyperLinkPosition ("Terms",$pos)
  $CAPTION = pat_getTagText (<A>,$pos)
  $URL = pat_getTagAttribute (<A>,"href",$pos)
  $SUMMARY = pat_getTagToTag (<A>,$pos)
</PATTERN>

```

Figure 3: An object model for link extraction. Terms are the keywords for filtering irrelevant links.

Extraction rules were manually derived by referring to search engines like Yahoo!, Google, AltaVista, and Lycos. When tested on Excite, Netscape, LookSmart, AOL, and GO/InfoSeek, a similar level of performance was obtained. Tables 4 and 5 show the system performance in terms of precision and recall for each individual search engine. GO/Infoseek had a poorer recall due to the fact that some captions were linked to images.

Table 4: Link extraction performance on the reference set.

	Search engine	Precision	Recall
1	Yahoo!	100%	85.33%
2	AltaVista	100%	100%
3	Google	100%	85.3%
4	Lycos	100%	95.33%

Table 5: Link extraction performance on other search engines.

	Search engine	Precision	Recall
1	Excite	100%	100%
2	Netscape	100%	100%
3	LookSmart	100%	90%
4	AOL	100%	100%
5	GO/Infoseek	100%	70%

## 5. Conclusion

This paper presents an object model approach to extracting information from HTML pages. Our object model may be considered a simplified version of wrapper. Its objective is to serve as an easy to use tool for the user to quickly write rules based on the structure of some sample web pages to extract the needed information. While the object model is also delimiter based, it is geared towards HTML tags and attributes. Furthermore, its looping construct through the use of multi-slot extraction allows recursive search of the relevant information.

Building on this architecture, object models for different types of web pages can be formulated, say for company pages, educational institution pages, online shopping pages, etc. Once object models spanning the various types of web pages are formulated, it becomes easy to build an automatic engine which, when given any web page, can extract

information and present it to the user in the format desired by the user. A personalized search engine can also be built by integrating all the object models.

## References

- [1] S. Soderland, D. Fisher, J. Aseltine and W. Lehnert. Crystal: Inducing a conceptual dictionary". *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pp1314-1319, 1995.
- [2] S. Soderland. Learning to extract text-based information from the World Wide Web. *Proceedings of Third International Conference on Knowledge Discovery and DataMining (KDD-97)*, pp251-254, 1997.
- [3] D. Freitag. Information extraction from HTML: Application of a general machine learning approach. *Proceedings of the 15th Conference on Artificial Intelligence (AAAI-98)*, pp517-523, 1998.
- [4] M. Craven, S. Slattery, and K. Nigam. First-Order Learning for Web Mining. *Proceedings, 10<sup>th</sup> European Conference on Machine Learning*, pp250-255, 1998.
- [5] C. Cardie. Empirical methods in information extraction. *AI magazine*, pp55-79, 1997.
- [6] D. DiPasquo. Using HTML Formatting to Aid in Natural Language Processing on the World Wide Web. *Senior Honors Thesis*, School of Computer Science, CMU, 1998.
- [7] Hannes Marais and Tom Rodeheffer. Automating the Web with WebL. In Dr. Dobb's Journal, January 1999.
- [8] Un Yong Nahm and J. Mooney. Using Information Extraction to Aid the Discovery of Prediction Rules from Text Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000) Workshop on Text Mining, pages 51 - 58, Boston, MA, August, 2000
- [9] Kushmerick., Weld, and Doorenbos, 1997. Wrapper induction for information extraction. *Proceedings of the 15th International Conference on Artificial Intelligence (IJCAI-97)* 729-735
- [10] C. Hsu and M. Dung, 1998. Generating finite-state trans-ducers for semi-structured data extraction from the web. *J.of Information Systems* 23(8):521-538
- [11] Fabien Azavant and Arnaud Sahuguet. The World Wide Web Wrapper Factory at <http://db.cis.upenn.edu/W4F/doc.html>