

Wrapping Web Data into XML

Wei Han, David Buttler, Calton Pu

Georgia Institute of Technology
College of Computing
Atlanta, Georgia 30332-0280
USA

{weihan, buttler, calton }@cc.gatech.edu

Abstract

The vast majority of online information is part of the World Wide Web. In order to use this information for more than human browsing, web pages in HTML must be converted into a format meaningful to software programs. Wrappers have been a useful technique to convert HTML documents into semantically meaningful XML files. However, developing wrappers is slow and labor-intensive. Further, frequent changes on the HTML documents typically require frequent changes in the wrappers. This paper describes XWRAP Elite, a tool to automatically generate robust wrappers. XWRAP breaks down the conversion process into three steps. First, discover where the data is located in an HTML page and separating the data into individual objects. Second, decompose objects into data elements. Third, mark objects and elements in an output format. XWRAP Elite automates the first two steps and minimizes human involvement in marking output data. Our experience shows that XWRAP is able to create useful wrapper software for a wide variety of real world HTML documents.

1 Introduction

XML is becoming a commonly accepted data exchange standard on the Internet, allowing distributed applications to be integrated into sophisticated services. There has been a lot of work on creating a new breed of distributed web services, such as supply-chain management and E-commerce. These new services are typically composed from component software programs or so-called agents. However, much of the valuable information on the Web is still, and will be in the immediate future, in "human-oriented" HTML documents.

A popular approach to capture HTML information in a format meaningful to software programs is to write wrappers to encapsulate the access to sources and produce a more structured data, such as XML, to enable a number of applications such as electronic commerce [3, 5, 6, 2]. (This is sometimes called *screen scraping*.) However, developing and maintaining wrappers by hand turned out to be slow, labor-intensive and error-prone. Semi-automatic wrapper generation systems improve the wrapper developing process, but are still not scalable enough to catch up with the explosion of Web pages, sites, and applications because of the inevitable cost of human involvement.

In this paper, we propose a systematic approach to build an automated system for wrapper construction for Web information sources. Called **XWRAP Elite**, it builds on our previous work in [4]. The goal of XWRAP Elite is to provide a tool for the easy transformation of human-orientated HTML into machine-readable, semantically meaningful XML. Implemented by wrappers, this transformation enables new XML-based applications, such as sophisticated query services, mediator-based information systems, and agent-based systems. A main challenge is to generate the wrappers automatically, minimizing human involvement and associated costs. Our main contribution in this paper consists of a set of algorithms and heuristics that balance the requirement of semantic input with the need to automate the information extraction and wrapper generation process.

2 System Architecture

Figure 1 shows the overall architecture of XWRAP Elite system. XWRAP Elite takes a sample docu-

of locating the sub-tree containing the relevant data objects (typically the meaningful text).

In XWRAP Elite, three individual sub-tree discovery methods have been implemented, as well as a method to combine them. The three methods are Largest Tag Count, Highest Fanout, and Largest Size Increase.

- The Largest Tag Count method considers the number of tags contained in each sub-tree. A large number of tags in a sub-tree typically indicates that a particular region of text has sophisticated structure and thus potentially richer in content than a sub-tree with a small number of tags.
- The Highest Fanout method compares sub-trees based on the number of immediate children each has. The larger the fanout, the more likely the sub-tree is the immediate parent of all of the data objects. This works well for tables, for example.
- The Largest Size Increase method compares the increase of the visible content between subtrees. Subtrees with a larger increase are more likely to contain usable content. The justification is similar to the Largest Tag Count, but in the measurement of text size (large content) instead of number of text elements (structural sophistication).

After the object-rich subtree extraction process, the problem of extracting the object separator tag in a web page is reduced to the problem of finding the right object separator tag in the chosen subtree. This problem is divided into two parts. First we decide which tags in the chosen subtree should be considered as candidate object separator tags. Since we are primarily interested in the content, we consider only the child nodes in the chosen subtree as candidate tags. Second, we identify the best object separator tag in the set of candidate tags that separate the objects most effectively. In the current version of XWRAP Elite, we use the Omini System [1], which supports five separator tag identification heuristics. Each of the five heuristics independently ranks the candidate tags; after the individual heuristics have completed their evaluation, the results are combined to improve the accuracy.

4 Element Extraction

After individual objects have been extracted from a page, the next step, called Element Extraction, identifies the elements inside of the objects. A data object

typically consists of several elements that are separated by a group of element separators. An element separator could be either an HTML tag, such as `<td>` in an HTML table, or a plain-text delimiter, such as the slash sign in "Cay Horstmann, Gary Cornell / Paperback / Prentice Hall /".

We have found that data objects from the same content region in a Web page are often homogeneous, meaning that they share the same structure and group of element separators. This allows us to decompose these data objects into similar elements once we locate the common group of element separators. So in our approach, we look for similar patterns in tag and text separation and then compile a group of separators that are suitable for most of the objects.

4.1 Element Separation

The methodology used in Section 3 to discover an object separator cannot be directly applied to searching for the group of element separators for two reasons. First, objects are similar to each other while elements in an object can vary widely in format. Second, the assumption that there is always a single HTML tag that marks the boundary between objects is not valid when applied to elements. Text delimiters often serve as element separators.

We chose two different approaches to discover tag-separators and text-separators. First, we build HTML tag-trees for objects and apply an initial heuristic on the trees to obtain a basic group of tag-separators. We then apply three complementary heuristics to refine the group of tag-separators. For text-separators, we extract text strings from objects and then analyze them with a string-based heuristic to get text separators.

4.1.1 Tag Separators

Given a data object that corresponds to an HTML tag tree, tag-separators decompose the tree into subtrees that contain full elements of interest, i.e., tag-separators don't break any element into pieces. For example, Figure 3 shows that `` is a tag-separator while `` is not since node `b` only contains a part of the book title, "Core Java 2, Volume 1: Fundamentals".

We search the appropriate combination incrementally. We first look for all the commonly-used tag-separators that are always separators in the object, such as `
`

and <a>. Then we apply the following complementary heuristics to discover additional tag-separators.

The first is the Highest Count Tag Heuristic. The higher the count of occurrences a tag has, the more likely it is a tag-separator. This heuristic treats the highest-count tag as a tag-separator.

The second heuristic is the Repeating Pattern Heuristic. Some tags often appear in adjacent pairs, either as a parent-child pair or as a sibling pair, multiple times. If a tag is frequently in a repeating pattern when it appears, it is very possibly a tag-separator. The frequency of a tag being in a repeating pattern is implied by the difference between the tag occurrences and the repeating pattern occurrences. The smaller the difference is, the more frequent it is, and the more likely the tag is a tag-separator.

The third heuristic is the Standard Deviation Heuristic. The standard deviation of the interval between tag occurrences indicates a tag's regularity. The smaller a tag interval's standard deviation is, the more regularly the tag occurs, and the more likely the tag is tag-separator.

4.1.2 Text Separators

Elements in plain text strings are usually separated by some symbols, such as " " and "/". We built a list of commonly used text-separators based on the Web documents we studied. The current list includes the following symbols: slash ("/"), colon (":"), space (" "), semicolon (";"). Perhaps surprisingly to Unix programmers, we do not include line separators, such as "\n", in our commonly used text-separator list. HTML treats the line separators as a simple space, so the line separator is often used even in the middle of an element.

However, we have observed that these commonly used text-separators sometimes also represent math signs or time formatting marks. For example, a slash can be used for the division operation (i.e. $15/3 = 5$), and a colon can separate time units (i.e. "15:30 p.m.").

Our text-separator discovery heuristic operates as follows. We scan an object; if any symbol on the list appears in the object without digits occurring on both sides, we put the symbol into a group of candidate text-separators. The pruned group contains all the text-separators for the object.

4.2 Leveraging Separators

After we obtain tag-separators and text-separators for each object, we leverage them to build a representative separator group, which contain only separators that appear in most groups. The current criteria include separators appearing in at least two groups and more than five percent of the total groups. Such a leverage avoids a symbol being a text-separator when it is an integral part of a text element, such as the colon in Figure 2.

5 Output Tagging

After Element Extraction, we obtain data objects in elements. However, these data objects are still strings of text containing HTML tags. They are semantically meaningful only because of human interpretation. The machine-readable semantic information is added by marking objects and elements with XML tags associated with their semantic meaning. This process is called **Output Tagging**.

A common approach to identify data elements is using string regular expression matching. However, accurate regular expressions are always very difficult to obtain automatically, especially when elements are similar. An inaccurate regular expression will misidentify data elements. Another approach to identify elements is based on the order of appearance of elements in an object. It works well for certain application domains, such as laboratory-experiment logs, but it cannot be applied to other areas. Furthermore, when an element is missing, it causes the misidentification of the rest of the elements in an object.

Our approach is a hybrid that uses both regular expressions and the element appearance order. We initially assign an index number to each element according to the order in which they appear. Then we automatically generate some regular-expression patterns to help us align the index numbers in case an element is missing. Elements with the same number will be tagged in the same name.

Figure 4 shows elements separated from the two objects in Figure 2. Object1 has 14 elements, while two elements are missing in Object2. The 10th element in Object2 doesn't match the 10th element in Object1. However, the 10th element in Object2 has the same type (i.e. Hypertext link) as the 12th element in Object1, and the 11th element (i.e. "more") in Object2 has the same value as the 13th element in Object2. So we can align the 10th, 11th, 12th elements in Object2

Order	Object1	Object2
1		
2	Core Java2, Volume 1: Fundamentals	Java How To Program
3	In-Stock: Ships with 24 hours	In-Stock: Ships with 24 hours
4	Cay Horstmann, Gary Cornell	Harvey M. Deitel, Paul J. Deitel
5	Paperback	Paperback
6	Prentice Hall	Prentice Hall
7	November 1998	August 1999
8	Our Price:	Our Price:
9	\$34.40	\$64.75
10	You Save	
11	20%	more
12		
13	more	
14		

Figure 4: Elements From Two Objects Before Alignment

to the 12th, 13th, 14th elements in Object1, which is shown in Figure 5.

5.1 Discovering Regular Expression Pattern

Regular expression patterns can be strong-matching and weak-matching. A strong-matching pattern normally implies a high possibility that two elements should be aligned if they both match the pattern, while a weak-matching pattern indicates that two elements should not be aligned if only one of them matches the pattern.

Constant-value elements, such as "Review" or "Our price", are often strong-matching patterns. Some elements contain constant strings with some variable numbers, for example, "ends in 5 hours 10 minutes." If we discard the variables, ("5" and "10" in this case,) the element has a constant-value-like pattern. Constant-value and constant-value-like elements appear in a substantial ratio of objects, so that we can easily discover them from statistics by counting their appearances.

We automatically recognize four types of weak-matching patterns, which corresponds to four different classes of elements: a hypertext link, an image, a dollar value (any element starting with "\$" and followed with numbers, such as "\$45.00"), and common strings (the rest of the elements).

These patterns help us avoid mismatching elements.

Order	Object1	Object2
1		
2	Core Java2, Volume 1: Fundamentals	Java How To Program
3	In-Stock: Ships with 24 hours	In-Stock: Ships with 24 hours
4	Cay Horstmann, Gary Cornell	Harvey M. Deitel, Paul J. Deitel
5	Paperback	Paperback
6	Prentice Hall	Prentice Hall
7	November 1998	August 1999
8	Our Price:	Our Price:
9	\$34.40	\$64.75
10	You Save	
11	20%	
12		
13	more	more
14		

Figure 5: Elements From Two Objects After Alignment

For example, in Figure 4, the 10th element in Object2, which is a hypertext link, should match the 12th element in Object1 instead of the 10th element in Object1.

5.2 Element Alignment

Element alignment matches elements to the elements in the largest-element-count object. The basic idea is to assign each element an index, which is the order of its matching element in the largest-element-count object. Elements with the same index share the same element name.

We search a matching element for an element E as follows.

- If E is constant-value-like and it has a strong-matching element in the largest-element-count object, assign the index of the matching element to E . If there are multiple strong-matching elements, choose the one with a closer index location. If there is no matching element or E is not constant-value-like, go to the next step.
- If an element in the largest-element-count object, whose index is larger than the index of the element that is immediately before E , shares a weak-matching pattern with E , we assume it is the matching element. If there are multiple matching elements, we choose the one with the smallest index. If there is no matching element, go to the next step.

- If an element in the largest-element-count object, whose index is smaller than or equal to the index of the element that is immediately before E , shares a weak-matching pattern with E , we assume it is the matching element. If there are multiple matching elements, we choose the one with the largest index. If there is no matching elements, E 's matching element is marked "unknown".

6 Conclusion

We have described XWRAP Elite, a tool for automatically generating wrappers for Web information sources. XWRAP Elite analyzes HTML documents a fine-grained object and element level to generate most of the wrapper code automatically. In contrast, many existing approaches require the wrapper developers to write information extraction rules by hand using a domain-specific language or to input their knowledge through an interactive GUI.

The automation of data extraction offers a number of advantages. XWRAP Elite is a scalable solution in the sense that creating a new wrapper costs much less time. The user only needs to input object and element names. It facilitates wrapper maintenance since web page changes are accommodated typically by running XWRAP Elite again. We are currently investigating techniques that support the automated detection and update of wrappers when target web pages change.

References

- [1] D. Buttler, L. Liu, and C. Pu. A fully automated object extraction system for the world wide web. *Proceedings of IEEE International Conference on Distributed Computing Systems*, April 2001.
- [2] J. Hammer, M. Brenning, H. Garcia-Molina, S. Nesterov, V. Vassalos, and R. Yerneni. Template-based wrappers in the tsimmi system. In *Proceedings of ACM SIGMOD Conference*, 1997.
- [3] C. A. Knoblock, S. Minton, J. L. Ambite, N. Ashish, P. J. Modi, I. Muslea, A. Philpot, and S. Tejada. Modeling web sources for information integration. In *Proceedings of AAAI Conference*, 1998.
- [4] L. Liu, C. Pu, and W. Han. XWrap: An XML-enabled Wrapper Construction System for Web Information Sources. *Proceedings of the International Conference on Data Engineering*, 2000.
- [5] L. Liu, C. Pu, and W. Tang. Continual queries for internet-scale event-driven information delivery. *IEEE Knowledge and Data Engineering*, 1999. Special Issue on Web Technology.
- [6] L. Liu, C. Pu, W. Tang, J. Biggs, D. Buttler, W. Han, P. Benninghoff, and Fenghua. CQ: A Personalized Update Monitoring Toolkit. In *Proceedings of ACM SIGMOD Conference*, 1998.
- [7] D. Raggett. Clean Up Your Web Pages with HTML TIDY. <http://www.w3.org/People/Raggett/tidy/>, 1999.
- [8] Extensible markup language (XML) 1.0. Technical report, W3C, 1998.