Wrapping 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 information that is available online, and coming online in this near future is only available in HTML. In order to use this information for more than human browsing, it must be converted into a machine-readable format. Wrappers have been the key tool to make the conversion from HTML into semantically meaningful and well-structured XML data. However, developing wrappers is slow and tedious work with typically brittle results. This paper describes XWRAP Elite, a tool to automatically generate robust wrappers, which breaks down the conversion process into three procedures: discovering where the data is located in an HTML page and separating the data into individual objects; decomposing objects into data elements; marking objects and elements in an output format. XWRAP Elite automates the first two procedures and requires minimal human involvement in marking output data. In addition, there is a code generation component to package all of the pieces into a stand-alone wrapper.

1 Introduction

There has been a lot of work on creating a new breed of distributed web services, such as supply-chain management and E-commerce. XML is becoming a universal data exchange standard on the Web, offering an easy method to integrate distributed applications to provide these sophisticated services. These new services are also a revolution from existing information sources in that they are accessible via software programs or agents. However, where does XML data come from? Most valuable information on the Web is still, and will be in the immediate future, in "human-oriented" HTML.

A popular approach to solve the problem is to write wrappers to encapsulate the access to sources and produce a more structured data, such as XML, to enable other applications [3, 5, 6, 2]. However, developing and maintaining wrappers by hand turned out to be 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**, which is an improvement of our previous work in [4]. The goal of our work is to provide a methodology for the easy transformation of human-orientated HTML into machine-readable, semantically meaningful XML. This transformation will enable numerous XML-based applications, such as sophisticated query services, mediator-based information systems, and agent-based systems. A main challenge is to generate the transformation automatically, requiring as little human involvement as possible. Our main contribution here is to provide 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 document as input, and generates a wrapper to convert the HTML into meaningful XML data through four

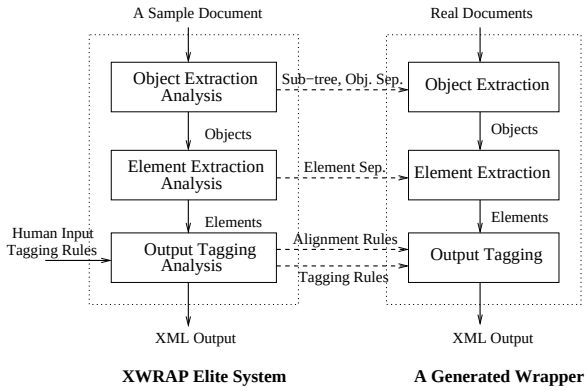

Figure 1: XWRAP Elite System Architecture

steps. As a side effect, tagged XML data for the sample document is produced to help validate the process.

In the first step, XWRAP Elite parses the sample document into an HTML tree, locates the sub-tree that contains the data objects and discovers rules to split the sub-tree into individual objects. It then generates an *Object Extraction* component based on the sub-tree and the object separator rules. XWRAP Elite also extracts objects from the sample document and passes them into the next step.

Next, XWRAP Elite studies the extracted objects to obtain a group of element separators and then decomposes the objects into elements. Element separators include both HTML tags and plain-text strings. The group of element separators determines an *Element Extraction* component.

Third, XWRAP Elite analyzes the elements from the remaining objects to learn element patterns based on regular expressions and element orders, and generate alignment rules to group similar elements into the same location across objects. A wrapper developer inputs tagging rules by assigning an element name to each group. XWRAP Elite generates an *Element Tagging* component according to the alignment rules and the tagging rules.

Finally, XWRAP Elite packages a wrapper by integrating the Object Extraction, Element Extraction and Output Tagging component. The wrapper can convert an HTML page similar with the sample document into XML.

The rest of the paper is organized as follows. Section 3 describes the automated object extraction procedure. Section 4 describes how elements are extracted for individual objects. Section 5 describes how elements are semantically tagged. Section 6 concludes the paper.

3 Object Mining and Extraction

There are three steps to extracting data objects from an HTML document: document preparation, primary content location, and object separation. Once these steps are completed, the data objects are still strings of text containing HTML tags. They are not, as of yet, semantically meaningful outside of human interpretation. The semantic information is added in the following steps.

The first step prepares the HTML for object extraction. It takes a raw HTML page and performs the following two tasks:

First, the page is cleaned using a syntactic normalization algorithm, which transforms the given page into a well-formed document which follows rules similar to well-formed XML documents [8].

Second, the document is converted into a tag tree representation based on the nested structure of start and end tags. There are many standard tools that can convert plain HTML text into a well-formed document, such as Tidy [7] from the W3C. Once the document is well-formed, it is trivial to create a tree structure from it. Figure 2 shows two data objects from a Barnsandnobles' Web page, and Figure 3 displays the tree structure of the first object.

Figure 2: Two Data Objects on Barnsandnoble

As has been noted earlier, web pages are designed for human browsing. In addition to the primary content region, many web pages often contain other information such as advertisements, navigation links, and so on. Therefore, given a tree structure for a web document, the task of identifying which part of the document is the primary content region is reduced to the

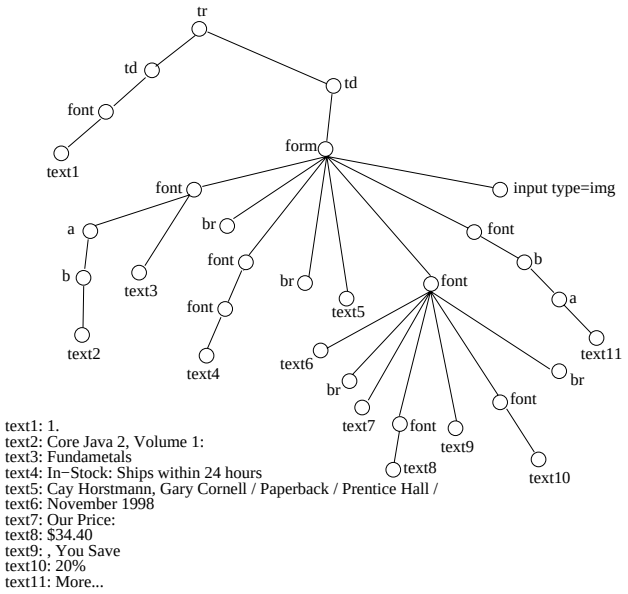

Figure 3: The Tree Structure for The First Object in Figure 2

problem of locating the smallest sub-tree of which contains all the relevant data objects.

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 larger number of tags indicate that a particular sub-tree is richer in content.
- 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.
- 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.

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 minimal subtree. The problem can be addressed in two steps. First we need to decide which tags in the chosen minimal subtree should be considered as candidate object separator tags. Second, we need a method to identify the right object separator tag from the set of candidate tags, which will effectively separate all the objects.

There are several ways of choosing the object separator tags. One may consider every node in the chosen subtree as a candidate tag or just the child nodes of the chosen subtree as the candidate tags. Based on the semantics of the minimal object-rich subtree, it is sufficient to consider only the child nodes in the chosen subtree as the candidate separator tags.

In the current version of XWRAP Elite, five separator tag identification heuristics are supported, covering a wide range of possible mechanisms for discovering object separators. 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.

See [1] for further detail.

4 Element Extraction

After individual objects have been extracted from a page, the next step is to identify the elements inside of the objects. This process is termed Element Extraction. 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 all of the data objects into elements once we locate the common group of element separators. So in our approach, we look for tag-separators and text-separators for each data object and then compile a group of separators that are suitable for all the objects.

4.1 Element Separation

The methodology used in Section 3 to discover an object separator cannot be applied to searching for the group of element separators. First, objects are similar to each other while elements in an object can vary widely in format. Also, the assumption that there is

always a single HTML tag that marks the boundary between objects is not valid when applied to elements. As we pointed out earlier, an element separator can also be a text delimiter.

We choose 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 propose three complimentary heuristics to revise 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 discompose 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 <font> is a tag-separator while <b> 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:

"/ ", ": ", " ", "; "

Surprisingly, 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. In order to output them in XML, we must mark objects and elements indicating their semantic meaning. This process is called **Output Tagging**. We assume data objects are homogenous, so we can name all the objects in one name. However, it is more difficult to mark elements because an object has multiple elements and some elements could be missing in the object, so we have to identify elements before we name them.

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.

Order	Object1	Object2
1	<a href="...">	<a href="...">
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	<a href="morelink">
11	20%	more
12	<a href="morelink">	<img src="...">
13	more	
14	<img src="...">	

Figure 4: Elements From Two Objects Before Alignment

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

Order	Object1	Object2
1	<a href="...">	<a href="...">
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	<a href="morelink">	<a href="morelink">
13	more	more
14	<img src="...">	<img src="...">

Figure 5: Elements From Two Objects After Alignment

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. 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 Ob-

ject1.

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 presented our approach for automatically generating wrappers for Web information sources. We automated the process to extract data at a fine-grained element level, while most of the 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. It does not necessarily require a set of sample documents, which allows other applications to integrate a data extraction component at run time. XWRAP Elite is a scalable solution in a sense that creating a new wrapper costs much less time. The user only needs to input object and element names. It facilitates ease of wrapper maintenance. Incorporating other technology, such as micro-feedback, it is possible to automatically revise wrappers when the data source has some changes.

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. Brennig, H. Garcia-Molina, S. Nesterov, V. Vassalos, and R. Yerneni. Template-based wrappers in the tsimmis 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.