Automatic HTML to XML Conversion

Shijun Li[1], Mengchi Liu[2], Tok Wang Ling[3], and Zhiyong Peng[4]

[1] School of Computer, Wuhan University, Wuhan, China 430072
shjli@public.wh.hb.cn
[2] School of Computer Science, Carleton University,
1125 Colonel By Drive, Ottawa, ON, Canada K1S 5B6
mengchi@scs.carleton.ca
[3] School of Computer, National University of Singapore,
Lower Kent Ridge Road, Singapore 119260
lingtw@comp.nus.edu.sg
[4] State Key Lab of Software Engineering, Wuhan University,
Wuhan, China 430072
peng@whu.edu.cn

Abstract. We present a new approach to automatically convert HTML documents into XML documents. It first captures the inter-blocks nested structure, then the intra-blocks nested structure, which consists of blocks including headings, lists, paragraphs and tables in HTML documents, by exploiting both formatting information and structural information implied by HTML tags.

1 Introduction

As most web documents are in HTML, automatic understanding of HTML documents has many potential applications. How to capture the nested structure in HTML documents and convert it into XML is a significant challenge.

The main part of an HTML document is the *body* element, which consist of a sequence of *blocks* that may be *heading*, *paragraph* (*p* element), *list*, and *table*. A sequence of *blocks* partitioned according to HTML grammar may not be in the same semantic hierarchy. Thus, the first important step is to capture the inter-block nested structure in an HTML document. As a *heading* element, briefly describes the topic and nested structure, we can capture the inter-block nested structure of an HTML document according to different levels of *heading* in an HTML document. However, there are some HTML documents, in which the authors use formatting information rather than *heading* to show the nested structure. So we must exploit the formatting information that includes *bold*, *strong* tags and the biggest font etc. to capture the inter-block nested structures.

After capturing the inter-block nested structure, the next step is to capture the intra-block nested structure that includes *paragraph*, *list* and *table*. We exploit both structural information that HTML tags imply, such as a *list* consists of *items*, and formatting information to capture the intra-block nested structure.

HTML tables may have nested headings. To capture their nested structure, we need normalize HTML tables by inserting redundant cells into them. Based

Q. Li, G. Wang, and L. Feng (Eds.): WAIM 2004, LNCS 3129, pp. 714–719, 2004.

on the normalized table, we can map the attribute-value pairs to XML documents. To convert HTML tables without marked headings via *th* element, the key is recognizing their headings. We introduce the notion of *eigenvalue* in the formatting information and a heuristic rule to recognize the headings of HTML tables. Without losing generality, we only discuss the well-formed HTML document. Since for those HTML documents that are not well-formed, we can use a free utility HTML TIDY by W3C to convert them into well-formed HTML ones.

This paper is structured as follows. Section 2 introduces the approach that captures inter-block nested structure in HTML documents. Section 3 introduces the approach that captures intra-block nested structure. Section 4 discusses related work. Finally we conclude in Section 5.

2 Capturing Inter-block Nested Structure

2.1 Exploiting Structural Information

We use the content of the title as the root element, and convert the content of the *meta* elements in the *head* element as the attributes of the root element. The following rule shows how to convert *head* element.

Rule 1 Let D be an HTML document as follows:
$D =$<html><head><title>T</title> <meta name="n_1" content="s_1"...
name="n_k" content="s_k"> </head> <body>B</body> </html>.
Then $\psi(D) =< T\ n_1 =$ "s_1"$\ldots n_k =$ "s_k" $> \psi(B) < /T >$, where ψ is the converting function which converts an HTML document into XML one.

Most HTML documents use *heading* blocks to express its nested structure. We can use it to capture the inter-block nested structure as follows.

Rule 2 Let D be an HTML section: $D = S_0$<hn>T_1</hn>$S_1 \ldots$<hn>T_m </hn $> S_m$, where hn is the most important heading in D, and each of $S_0, \ldots,$ S_m is a sequence of HTML *blocks*. Then $\psi(D) = \psi(S_0) \oplus < T_1^{'} > \psi(S_1) < /T_1^{'} > \oplus \ldots \oplus < T_m^{'} > \psi(S_m) < /T_m^{'} >$, where $T_i^{'} = T_i$ if T_i is a character string, and $< T_i^{'} >=< N_i \ \ url =$ "U" $>$ if $T_i =$<a $url =$ "U"> N_i , $i = 1, 2, ...m$.

2.2 Exploiting Formatting Information

For the HTML documents in which the authors use formatting information rather than *heading* to show the nested structure, the key is to find out all the text strings that have the most important formatting information, whose function is just as the most important *heading*. Then the conversion is just the same as the conversion of the *heading* blocks. Rule 3 shows this process. The converting functions in Rule 2 and Rule 3 are recursive, we can recursively apply them to $S_0, \ldots, S_m$ till the whole nested structure in the HTML document is captured.

Rule 3 Let D be an HTML section and $T_1, \ldots, T_m$ the character strings that have the biggest font, or in bold tags, or in strong tags. Let $D = S_0\ T_1\ S_1\ \ldots\ T_m\ S_m$, where $m \geq 1$, and each of $S_0, \ldots, S_m$ is a sequence of HTML *blocks*. Then $\psi(D) = \psi(S_0) \oplus < T_1^{'} > \psi(S_1) < /T_1^{'} > \oplus \ldots \oplus < T_m^{'} > \psi(S_m) < /T_m^{'} >$, where the meaning of $T_i^{'}$, $i = 1, 2, ...m$, is the same as the one stated in Rule 2.

3 Capturing Intra-block Nested Structure

3.1 Converting Text-Level Elements and Paragraph Elements

After capturing the inter-block nested structure, the next step is capturing the intra-block nested Structure. If a character string does not contain formatting information stated in Rule 5, then we call it a simple character string. We convert character strings and *a* (anchor) elements as follows.

Rule 4 Let $D = S$. If S is a simple character sting, then $\psi(D) = S$. If S is an *a* element: $S =$<a $url =$ "U"> N <a>, then if it is not the content of a heading block (i.e. h1 to h6 element) which we address in Rule 2, we convert it as an XML empty element: $\psi(D) =$<N $url =$ "U"/>.

HTML documents usually contain nested structure via formatting information including bold, italic, or colon ':' etc. We can capture the nested structure by these emphasized formatting information as follows.

Rule 5 Let $D =< t > T_1 < t > S_1 \ldots < t > T_n < t > S_n$, or $D = T_1 : S_1 \ldots\ T_n : S_n$, where t is either `b`, `i`, `em`, or `strong`. Then $\psi(D) =< T_1^{'} > \psi(S_1) < /T_1^{'} > \ldots < T_n^{'} > \psi(S_n) < /T_n^{'} >$, where the meaning of $T_i^{'}$, $i = 1, 2, ...m$, is the same as the one stated in Rule 2.

If the content of a p element is a simple character string, we introduce an XML element *describe*. Otherwise we just convert its content.

Rule 6 Let $D =< p > S < /p >$ *or* $D =< p > S$ (as $< /p >$ can be omitted in HTML), where S is a text string. If S is a simple character string, then $\psi(D) =< describe > s < /describe >$. Otherwise, $\psi(D) = \psi(S)$

3.2 Converting Lists

HTML supports unordered lists *ul*, ordered lists *ol*, and definition lists *dl*. Both unordered and ordered lists consist of *li* element, which are their items. If a *li* element is a simple character string, we introduce an XML element named *item* to mark it. Definition lists consist of two parts: a *term* given by the *dt* element, and a *description* given by the *dd* element. We summarize the process for unordered lists and ordered lists as Rule 7 and for definition lists as Rule 8.

Rule 7 Let $L =< ul > L_1 \ldots L_n < /ul >$, or $L =< ol > L_1 \ldots L_n < /ol >$, where L_i is a *li* element and $i = 1, \ldots, n$. Then $\psi(L) = \psi(L_1) \oplus \ldots \oplus \psi(L_n)$. If the content of a *li* element is a simple character string S, then $\psi(L_i) =< item > S < /item >$.

Rule 8 Let $L =< dl > R_1 \ldots R_m < dl >$, where $R_i =< dt > T_i < /dt >< dd > D_{i1} < /dd > \ldots < dd > D_{in_i} < /dd >$, and $i = 1, 2, \ldots, m$; $n_i \geq 0$. Then $\psi(L) = \psi(R_1) \oplus \ldots \oplus \psi(R_m)$. If T_i is a simple character string, then $\psi(R_i) =< T_i > \psi(D_{i1}) \oplus \ldots \oplus \psi(D_{in_i}) < /T_i >$. Otherwise $\psi(R_i) = \psi(T_i) \oplus \psi(D_{i1}) \oplus \ldots \oplus \psi(D_{in_i})$.

3.3 Capturing the Attribute-Value Pairs of HTML Tables

HTML Table cells generally contain heading information via the *th* element and data via the *td* element, and may span multiple rows and columns. If a *th* or *td* element contains $colspan = n$, it means that the particular cell of the *th* or *td* is to be expanded to $n - 1$ more columns, starting from the current cell in the current row. If a *th* or *td* element contains $rowspan = n$, it means the particular cell of the *th* or *td* element is to be expanded to the next $n - 1$ rows in the current column. By inserting redundant cells according to the attributes colspan and rowspan, we can normalize an HTML table in the form of Table 1.

Table 1. The normalized HTML table

			$h_{1,q+1}$	$\ldots$	$h_{1,n}$
			$\vdots$		$\vdots$
			$h_{p,q+1}$	$\ldots$	$h_{p,n}$
$v_{p+1,1}$	$\ldots$	$v_{p+1,q}$	$d_{p+1,q+1}$	$\ldots$	$d_{p+1,n}$
$\vdots$		$\vdots$	$\vdots$		$\vdots$
$v_{m,1}$	$\ldots$	$v_{m,q}$	$d_{m,q+1}$	$\ldots$	$d_{m,n}$

Based on the normalized table Table 1, we introduce a mapping rule to map the attribute-value pairs of HTML tables to the corresponding XML documents, and a merging rule to merge the content of the same XML element as Rule 9, which covers not only two dimensional tables but also one dimensional tables.

Rule 9 Let the normalized table of an HTML table T be Table 1, the content of its *caption* be c, and the rows that contain data be $r_{p+1}, r_{p+2}, \ldots, r_m$, where $0 \leq p \leq m$. Then $\psi(T) =< c > \psi(r_{p+1}) \oplus \psi(r_{p+2}) \oplus \ldots \oplus \psi(r_m) < /c >$, where

$$
\begin{aligned}
\psi(r_i) =& < v_{i,1} > \quad \ldots \quad < v_{i,q} > \\
& < h_{1,q+1} > \ldots < h_{p,q+1} > \quad \psi(d_{i,q+1}) \quad < /h_{p,q+1} > \ldots < /h_{1,q+1} > \\
& \qquad\qquad\qquad\qquad \vdots \\
& < h_{1,n} > \ldots < h_{p,n} > \quad \psi(d_{i,n}) \quad < /h_{p,n} > \ldots < /h_{1,n} > \\
& < /v_{i,q} > \quad \ldots \quad < /v_{i,1} >, \\
& \textit{where } i = p+1, \ \ldots \ , m; \ 0 \leq p \leq m; \ 0 \leq q \leq n.
\end{aligned}
$$

Merging rule: $< t1 >< t2 > s2 < /t2 >< /t1 >$
$< t1 >< t3 > s3 < /t3 >< /t1 >$
$=< t1 >< t2 > s2 < /t2 >< t3 > s3 < /t3 >< /t1 >$

3.4 Recognizing Headings of HTML Tables

In HTML documents, there are many HTML tables without marked headings via *th* elements. To convert them by using Rule 9, we need to recognize their headings. Generally, authors use different formatting information, which are mainly font (size, bold), to mark headings for visual easy recognition. So we introduce a notion *eigenvalue* to measure the difference of formatting information.

Definition 1 In an HTML table, if the cell's font size is n, then its *eigenvalue* $\lambda = n$; moreover, if the cell has bold font, then $\lambda = \lambda + 1$. The *eigenvalue* of a row or a column is the *average eigenvalue* of all the cells in the row or column.

Based on the fact that the headings generally have bigger *eigenvalue* than the data part. We can recognize headings of HTML tables as follows.

Rule 10 Let an HTML table have m rows, and the eigenvalues of m rows be $\lambda_1, \ldots, \lambda_m$, respectively. Let the difference of two adjacent rows be $\lambda_{12}, \lambda_{23}, \ldots, \lambda_{(m-1)m}$, where $\lambda_{i(i+1)} = \lambda_i - \lambda_{i+1}, i = 1, \ldots, m-1$. Let k be the minimum that $\lambda_{k(k+1)} = \max(\lambda_{12}, \lambda_{23}, \ldots, \lambda_{(m-1)m})$ and $\lambda_{k+1} < \lambda_k$ hold. Then the first k rows are the column headings. The recognizing of row headings is just the same as of column headings.

4 Related Work

To capture the nested structure in HTML documents, the manual approaches [3, 6] often involve the writing of complicated code. Lixto [10] manage to avoid the writing of code by providing a fully visual and interactive user interface. The induction systems [1, 2] need to be trained with some examples, under human supervision. In [5], an ontology-based approach are proposed, which also involves the writing of complicated code, such as RDF. [7] present a case-based, but it only cover a portion of HTML pages. In [2], a semi-automatic wrapper generation approach is presented that only exploits formatting information to hypothesize the underlying structure of a page, so that it cannot correctly convert complex *table* and *list*. In [4], a heuristic approach for automatic conversion is introduced. However, it only uses the structural information, neglecting the formatting information, so that it cannot convert the HTML documents that use formatting information to mark its nested structure. Since HTML tables are used frequently in HTML documents and are information rich, they gain a lot of attentions recently [8, 9]. In [8], an automatic approach that extracts attribute-value pairs is presented. But it gives no formal approach to capture the attribute-value pairs of HTML tables with nested headings, and cannot capture the nested structure in cells of HTML tables.

In this paper, we introduce a new approach to automatically convert HTML documents into XML documents. A system based on the approach presented in this paper has been implemented. We must stress the fact that some of the presented rules are based on heuristics and might fail. Compared with the other

related approaches, as our approach exploits both formatting and structural information implied by HTML tags to automatically capture the nested structure in HTML documents, and introduces a mapping rule to automatically map the attribute-value pairs in HTML tables to XML documents and a heuristic rule to recognize headings of HTML tables by formatting information, it improves the expressive capability of the existing approaches.

5 Conclusion

The main contribution of this paper is that we present a new approach that automatically captures the nested structure in HTML documents and converts it into XML ones. Although capturing the nested structure in HTML documents is difficult, as our approach exploits both the formatting information and the structural information implied by HTML tags, it improves the expressive capability of the existing approaches. It can be used as an effective auxiliary tool in recycling HTML documents as XML documents.

Acknowledgements

This research is supported by NSFC (National Natural Science Foundation of China) (60173045).

References

1. Nicholas Kushmerick, D. Weld, and R. Doorenbos. Wrapper Induction for Information Extraction. In IJCAI (1997), 729–737
2. Naveen Ashish and Craig A. Knoblock. Semi-Automatic Wrapper Generation for Internet Information Sources. In CoopIS (1997), 160-169
3. Joachim Hammer, Hector Garcia-Molina, Svetlozar Nestorov, Ramana Yerneni, Markus M. Breunig, and Vasilis Vassalos. Template-Based Wrappers in the TSIMMIS System. In SIGMOD Conference (1997), 532–535.
4. Seung Jin Lim and Yiu-Kai Ng. A Heuristic Approach for Coverting HTML Documents to XML Documents. In J.Loyd et al., editor, CL (2000), 1182–1196
5. Thomas E. Potok, Mark T. Elmore, Joel W. Reed, and Nagiza F. Samatova. An Ontology- Based HTML to XML Conversion Using Intelligent Agents. In J.Loyd et al., editor, HICSS (2002), 120–129.
6. Arnaud Sahuguet and Fabien Azavant. Building Intelligent Web Applications Using Lightweight Wrappers. Data and Knowledge Engineering (2001), 36(3): 283–316
7. Masayuki Umehara, Koji Iwanuma, and Hidetomo Nabeshima. A Case-Based Recognition of Semantic Structures in HTML Documents. In IDEAL (2002), 141–147
8. Yingchen Yang and Wo-Shun Luk. A Framework for Web Table Mining. In WIDM'02 (2002), 36–42
9. Shijun Li, Mengchi Liu, Guoren Wang, and Zhiyong Peng. Capturing Semantic hierarchies to Perform Meaningful Integration in HTML Tables. In APWEB (2004), 899–902
10. R. Baumgartner, S. Flesca and G. Gottlob. Visual Web Information Extraction with Lixto. In VLDB (2001), 119–128