

A Heuristic Approach for Converting HTML Documents to XML Documents

Seung-Jin Lim and Yiu-Kai Ng

Computer Science Department, Brigham Young University
Provo, Utah 84602, U.S.A.
{ng,sjlim}@cs.byu.edu

Abstract. XML is rapidly emerging, and yet there still exist numerous HTML documents on the Web. In this paper, we present a heuristic approach for converting HTML documents to XML documents. During the conversion process, we eliminate all the HTML elements in an HTML document from the resulting XML document since these elements are designed for the display of data exclusively, but retain the character data of each element along with the implicit hierarchy among the data. The proposed conversion approach extracts the data hierarchy of HTML documents as closely as possible with no human intervention. The approach can be adopted to construct the data hierarchy of an HTML document and to collect data in HTML documents into an XML repository.

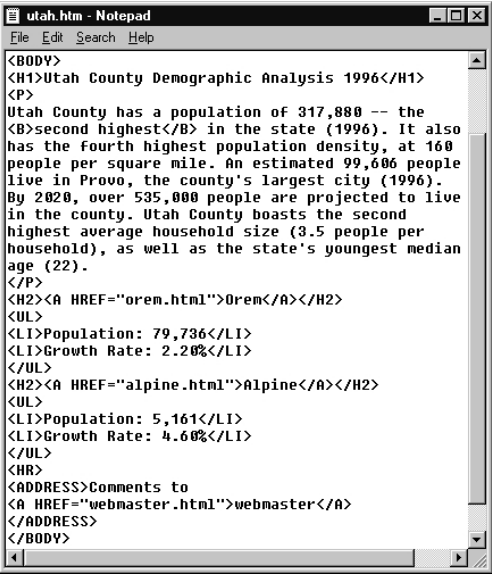
1 Introduction

Since Extensible Markup Language (XML) was introduced in 1996 and became the official W3C Recommendation in 1997, it has been emerging as a de facto standard in electronic data exchange quickly. At present, there are still a large number of new and existing HTML documents, either dynamically created via CGI-like technologies or statically created, on the Web. It is worth to migrate existing HTML documents to XML documents to avoid inefficiency and additional complexity for managing documents in two different formats. Migration of dynamic HTML documents to XML can be done easily by modifying the respective CGI-like modules. It is, however, more time consuming to migrate static HTML documents to XML unless conversion tools are provided.

In this paper, we present a heuristic approach, called *Html2Xml*, to convert HTML documents to XML documents. We consider the following two issues:

- First, in the resultant XML document, it is desirable to just retain data and exclude HTML tags that are contained in the source HTML document since the primary role of HTML tags is to merely define the display, i.e., the “look-and-feel,” of data.
- Second, the explicit or implicit data hierarchy of the source HTML document should be determined and preserved in the resultant XML document.

Consider the body block of the HTML document, *Utah County Demographic Analysis 1996* (UCDA), as shown in Figure 1. According to the container-content constraint specified in HTML grammar, Figure 2(a) shows a typical hierarchy of the body block of UCDA. In Figure 2(a), the hierarchy of the data, such as the



```
<BODY>
<H1>Utah County Demographic Analysis 1996</H1>
<P>
Utah County has a population of 317,880 -- the
<B>second highest</B> in the state (1996). It also
has the fourth highest population density, at 160
people per square mile. An estimated 99,606 people
live in Provo, the county's largest city (1996).
By 2020, over 535,000 people are projected to live
in the county. Utah County boasts the second
highest average household size (3.5 people per
household), as well as the state's youngest median
age (22).
</P>
<H2><A HREF="ore.html">Orem</A></H2>
<UL>
<LI>Population: 79,736</LI>
<LI>Growth Rate: 2.20%</LI>
</UL>
<H2><A HREF="alpine.html">Alpine</A></H2>
<UL>
<LI>Population: 5,161</LI>
<LI>Growth Rate: 4.60%</LI>
</UL>
<HR>
<ADDRESS>Comments to
<A HREF="webmaster.html">webmaster</A>
</ADDRESS>
</BODY>
```

Fig. 1. Body block of utah.htm

population and growth rate of a city in Utah County, is interleaved with HTML tags such as Bs, ULs, and LIs. As a result, the data content of the first paragraph (P) yields three leaf nodes in the hierarchy since it is divided into three parts by B. Because of the separation, the association between a city name and its population and growth rate is not clearly shown in the figure. We believe the hierarchy in Figure 2(b) depicts the relationship among the data “better” than that of Figure 2(a) since in Figure 2(b) population 79,736 is tied with its city “Orem” and population 5,161 is tied with its city “Alpine” according to the container-content relationship.

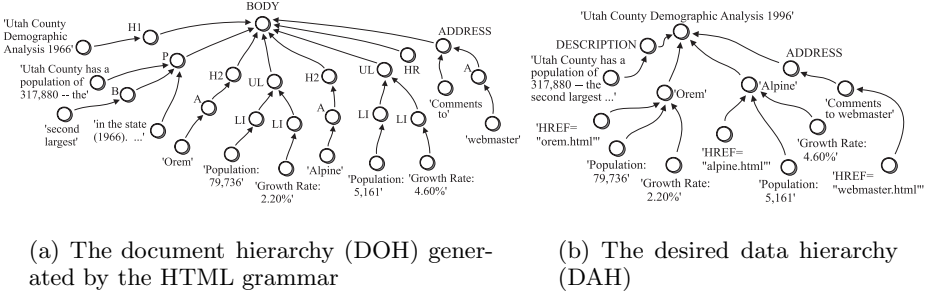


Fig. 2. Two different hierarchies of the HTML document in Figure 1

Our Html2Xml approach is based on the notion of *data hierarchy* (DAH). The DAH of an HTML document H is a tree representation of the data in H , with the exclusion of most of the HTML elements in H . Also, data in a DAH are identified by their hierarchical relationships with other data without using HTML elements. In this regard, the notion of DAH distinguishes itself from other approaches [1,2,6,7] where manipulation of HTML elements is indispensable in the process of locating a particular data item. Furthermore, our Html2Xml approach does not require any knowledge of the structure of a source HTML document beforehand, and we generate the XML representation of all the data, rather than a selected portion, in an HTML document with no human intervention, which is required in [10].

We proceed to present our results as follows. In Section 2, we include preliminary definitions that are used for further discussions in this paper. In Section 3, we introduce our approach for converting HTML documents to XML documents. In Section 4, we give the concluding remarks.

2 HTML/XML Documents

There are two major constructs in XML as well as in HTML documents: elements and character data (data in short). An *element* is delimited by its *start-tag* and *end-tag*, unless it is an empty element, and text that do not start with ‘<’ are *character data*. Moreover, an element is identified by its *name* and accompanied with an *attribute list* of zero or more attributes, each of which is of the form *attribute name = value*. (For a data item d , we consider d itself as the name of d .) An element contains *content* that appears between its start- and end-tags. For some non-empty HTML elements, the end-tag is implicit, whereas a non-empty XML element is always ended with an end-tag. The content of an HTML/XML element is either another element (i.e., elements can be nested) or character data. We call the former *element content* and the latter *data content*. Also, given an element e and its immediate content c , we say that e *immediately contains* c , denoted by $e \leftarrow c$, and e is called the *immediate container* of c or c is the *immediate content* of e . For instance, the assertion $\text{BODY} \leftarrow \text{UL} \leftarrow \text{LI} \leftarrow \text{‘Population: 5,161’}$ holds for the hierarchy in Figure 2(a) according to the HTML grammar. Empty elements are non-container elements.

Recall that a content is either an element or data, whereas a container is always an element. When an HTML document H is converted to an XML document X , data that encompass other data in H , such as **Orem** and **Alpine** in Figure 2(a), are converted to elements in X . Besides the container-content constraint, we use *child element* to denote an immediate element content and *parent element* to denote an immediate container in an XML document. Also, throughout this paper whenever we say “an element e is removed,” we mean that the start- and end-tags of e are removed but the data content of e is retained.

An HTML/XML document can be perceived as a tree, called *document hierarchy* (DOH), where a node denotes an element or data component, and edges are determined by the container-content relationships among nodes. If each node in a DOH D exclusively represents an HTML/XML data or an XML element, we call D a *data hierarchy* (DAH), denoted DAH_D .

Definition 1. Given an HTML/XML document D , the *document hierarchy* (DOH) of D is a tree, denoted $\text{DOH}_D = (V, E, g)$, where (i) a node in V denotes an element e or a character data d in D and is labeled by the name of e or d , respectively¹. For any node n and its child nodes n_1, n_2, \dots, n_m , n_j appears before n_k in the left-to-right, top-to-bottom manner in D if $1 \leq j < k \leq m$. The root node V_R in V is the **BODY** element if D is an HTML document, or the first element appeared after the prolog block if D is an XML document; (ii) E is a finite set of directed edges; and (iii) $g : E \rightarrow V \times V$ is the *edge definition function* such that $g(e) = (v_1, v_2)$ if $v_2 \leftarrow v_1$. \square

According to Definition 1, the hierarchy shown in Figure 2(a) is the DOH of **utah.htm**, where the name of each leaf node is surrounded by ‘ ’. To reference an element or data v in a hierarchy where v belongs, we define the *context* of v .

¹ Since we guarantee one-to-one relationships between D and DOH_D , we interchangeably use the nodes in DOH_D and the elements/data in D , as well as D and DOH_D .

Definition 2. Given a set S of elements or data that are hierarchically structured, the *context* of an element or data v in S is defined as $X_v = N_v$, if v is the root node of the hierarchy; otherwise, $X_v = X_{v'}.N_v$, where N_v is the *name* of v and $X_{v'}$ is the *context* of the predecessor of v in the hierarchy. (The dot ‘.’ in $X_{v'}.N_v$ is the *dot* operator which is the separator of $X_{v'}$ and N_v .) \square

The context of a node v in a DOH is the concatenation of the names of v and its ancestors along the path from the root node, and each distinct name is delimited by ‘.’ from its adjacent names. Furthermore, using an ordered list of context of the leaf nodes in a DOH, we can represent the DOH as a string. We adopt the convention $A.(B, C)$ for $(A.B, A.C)$, where $(A.B, A.C)$ is an ordered list of context. In addition, if sibling nodes share the same name, we append an index number to their names, such that $node[1]$ denotes the first node with the name $node$, according to their order of appearance in the source document.

Consider the hierarchy in Figure 2(b). Using the context of each leaf node, the data in the subtree rooted at **Alpine** can be expressed as follows:

(‘Utah County Demographic Analysis 1996’.‘Alpine’.‘HREF=“alpine.html”’,
‘Utah County Demographic Analysis 1996’.‘Alpine’.‘Population: 5,161’,
‘Utah County Demographic Analysis 1996’.‘Alpine’.‘Growth Rate: 4.60%’) =
‘Utah County Demographic Analysis 1996’.‘Alpine’.(
‘HREF=“alpine.html”’, ‘Population: 5,161’, ‘Growth Rate: 4.60%’)

3 Construction of the Data Hierarchy of an HTML Document

We now present our approach for constructing the data hierarchy (DAH), which captures the hierarchical relationships among the data contents, of a given HTML document D . The construction process of DAH_D includes (i) the exclusion of HTML elements from D and (ii) the refinement of the container-content relationships among the data in D .

3.1 Exclusion of HTML Elements

Among the defined HTML elements, the role of some elements is to define the block structure of data (called *type 1*), whereas others are to define the cosmetic style of the rendered data in a Web browser exclusively (called *type 2*).

We group HTML elements into type 1 and type 2 based on whether they form blocks of data such that one block is distinguishable from another hierarchically. Table 1 shows the grouping which follows the widely-supported HTML 3.2 [8]. Note that the primary role of text-level elements is to define the style of data, and they do not contribute to the formation of a hierarchical block with the exception of **A**. (**A** is considered as type 1 since any data d appeared in a linked document from another HTML document D are subordinate to D and **A** is a lead to d .) Also, we include into type 2 head-related elements, as well as all the non-container elements among the rest of the HTML elements. *Head-related* elements are type 2 elements since no data is presented in the head block, and all *empty* elements are excluded from type 1 as well since they contain no data. *Form-related* and *Java applet-related* elements are also excluded from type 1 since they

Table 1. Two groups of HTML elements, type 1 and type 2

HTML elements		type 1	type 2
head content			TITLE, META, ISINDEX, BASE, LINK, SCRIPT, STYLE, META
body content	headings	H1, H2, H3, H4, H5, H6	
	block-level	UL, OL, DIR, MENU, LI, DL, DT, DD, P, DIV, TR, TABLE, TH, TD, CAPTION, CENTER, BLOCKQUOTE	ISINDEX, HR, XMP, LISTING, PLAINTEXT
	font		TT, I, B, U, STRIKE, BIG, SMALL, SUB, SUP
	text-level phrase		EM, STRONG, DFN, CODE, SAMP, KBD, VAR, CITE
	special	A	IMG, APPLET, FONT, BR, BASEFONT, SCRIPT, MAP
	form		FORM, INPUT, SELECT, OPTION, TEXTAREA
	client-side image maps		MAP, AREA
	Java applet		APPLET, PARAM
address		ADDRESS	

† META with *name* attribute=“keywords” and *< BODY >* are considered as type 1, while other METAs, *< HTML >*, *< HEAD >*, and *< ! >* are treated as type 2 elements.

are used to query data rather than to present data. Among the elements that do not appear in Table 1, we exclude HTML and HEAD from type 1 but retain BODY as type 1 to ensure that the resulting DAH of an HTML document is a tree.

Note that table-related elements were not included in the earlier version of HTML specification and were added later to present data in a tabular form. They are the most versatile among the HTML elements in organizing data hierarchically and are treated as type 1. We emphasize HTML tables since they are very capable in presenting data hierarchically, and most of the data generated dynamically through CGI-like programs are presented in HTML tables.

Given an HTML document *D*, we first remove the HTML elements of type 2 from *D* and the resulting document is called an *approximated* document. For example, after B is removed from *utah.htm*, the content of P is restored as a paragraph in the approximated *utah.htm*, and HR is also removed hereafter.

3.2 Edge Definition Function of DAH

During the process of defining edges, we apply four *hierarchy resolution rules* to *D* which provide the mechanism for refining the hierarchy of *D*. The first rule describes how to handle an anchor A and its data in our Html2Xml approach.

Rule 1. Given an ordered list of container-content relationships ($p \leftarrow C_1, p \leftarrow A \leftarrow C_2, p \leftarrow C_3$), where *p* is the parent of *C*₁, A, and *C*₃, A is an anchor element, and *C*_{*i*} ($1 \leq i \leq 3$) is a data item, the *edge definition function g* of DAH yields $p \leftarrow C \leftarrow HREF$, where *HREF* is the *name* and *value* pair of the HREF attribute in A, and $C = concat(concat(C_1, C_2), C_3)$ ². Hereafter, A is removed. □

² $Concat(s_1, s_2) = trim(s_1 + s + s_2)$, where *s*₁ and *s*₂ are strings, *s* is a space, + is a string concatenation operator, and *trim* removes any leading and trailing spaces.

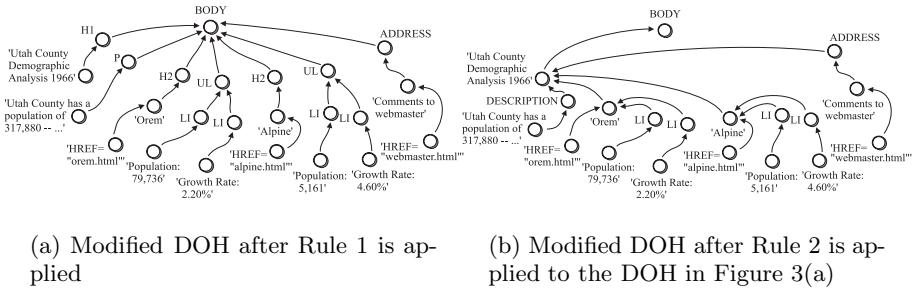


Fig. 3. Evolving DOHs

By Rule 1, we restore the data that might have been fragmented by **A**, and retain the HREF attribute of **A** as a placeholder for the data in the linked document.

Consider the ADDRESS element in `utah.htm` as shown in Figure 2(a) whose content is separated into three parts. By Rule 1, we obtain two edges such that ADDRESS \leftarrow 'Comments to webmaster' \leftarrow 'HREF="webmaster.html"'. Also, H2[1] \leftarrow A[1] \leftarrow 'Orem' yields H2[1] \leftarrow 'Orem' \leftarrow 'HREF="orem.html"', and H2[2] \leftarrow A[2] \leftarrow 'Alpine' yields H2[2] \leftarrow 'Alpine' \leftarrow 'HREF="alpine.html"'. Figure 3(a) shows the modified DOH after Rule 1 has been applied to the approximated `utah.htm`.

Next, we refine the hierarchy of the nodes at the same level in the current DOH. In this process, we adopt the precedence spectrum as shown in Figure 4, where items on the left of ‘ \gg ’ have higher precedence than that on the right.

We place headings higher than other body content elements since they are headings of the text blocks that appear next to the headings, and H1 has higher precedence than H6. Since the roles of OL, DL, DIR, and MENU are similar to that of UL, they have the same precedence. (In fact, DIR, MENU, and UL are treated the same in many Web browsers.) The roles of DIV, CENTER, and BLOCKQUOTE are similar, and hence we place them at the same precedence level. (In fact, CENTER is identical to DIV whose ALIGN attribute value is CENTER.) All these elements, along with ADDRESS, have the same precedence as P and TABLE since none of these elements is subordinate to one another. Since LI is used as a content of UL, OL, DIR and MENU, DT is used as a content of DL, and CAPTION is a content of TABLE according to the HTML specification, we place these elements at the next level in the precedence spectrum. DD is placed at the next level for a similar reason. TR, on the other hand, should have the same precedence as CAPTION according to the HTML specification; however, we place it at a level lower than CAPTION since we consider the content of CAPTION as the title of the entire table. The respective precedence of TR, TH and TD are obvious according to the container-content specification of these elements.

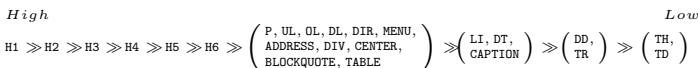


Fig. 4. Precedence among HTML elements

Rule 2. Given a set of sibling elements e_1, \dots, e_n which are located from left to right in an HTML document, for any two sibling elements e_i and e_j ($1 \leq i <$

$j \leq n$) such that $e_i \leftarrow c_1$ and $e_j \leftarrow c_2$, applying the *edge definition function* g to e_1, \dots, e_n yields (i) $c_1 \leftarrow e_j \leftarrow c_2$, if e_j is ADDRESS and $e_i \gg e_j$, (ii) $c_1 \leftarrow \text{DESCRIPTION} \leftarrow c_2$, if e_j is P and $e_i \gg e_j$, (iii) $e_i \leftarrow e_j \leftarrow c_2$, if e_i is CAPTION and e_j is TR, or (iv) $c_1 \leftarrow c_2$, if e_j is neither ADDRESS, P nor TR and $e_i \gg e_j$.

For an element e with more than one element on the left of e which has higher precedence than e , we retain only the edge between the element of the highest precedence and e if e is ADDRESS; otherwise, we retain the edge between e and the nearest element to e . Hereafter, any empty HTML elements are removed. \square

By Rule 2, the contents of a set of sibling elements are refined, if necessary. Note that we treat ADDRESS separately since it has a unique role, such as authorship and contact details for the source HTML document. We rename P elements as DESCRIPTIONs until the construction of an XML document X is completed since they play special roles during the construction of X . We will discuss CAPTION and TR in details in the next section.

Consider the DOH in Figure 3(a) and apply Rule 2 to the child elements of BODY. Since $H1 \gg P$, ‘Utah County Demographic Analysis 1966’ \leftarrow DESCRIPTION \leftarrow ‘Utah County has a population of ...’ by Rule 2(ii). Consider P and H2[1]. Since $H2 \gg P$ but P appears before H2[1], the contents of these elements are not rearranged. However, since $H1 \gg H2$, ‘Utah County Demographic Analysis 1966’ \leftarrow ‘Orem’ \leftarrow ‘HREF="orem.html"' by Rule 2(iv). Now, compare H2[1] and UL[1]. Since $H2 \gg UL$, (‘Orem’ \leftarrow LI[1], ‘Orem’ \leftarrow LI[2]) by Rule 2(iv), and eventually (‘Orem’ \leftarrow LI[1] \leftarrow ‘Population: 79,736’, ‘Orem’ \leftarrow LI[2] \leftarrow ‘Growth Rate: 2.20%’). H2[2] and UL[2] are processed similarly. Figure 3(b) shows the modified DOH after Rule 2 has been applied. Note that ADDRESS is retained and LI[1] (LI[3], respectively) cannot be resolved against LI[2] (LI[4], respectively) since LI[1] and LI[2] (LI[3] and LI[4], respectively) have the same precedence.

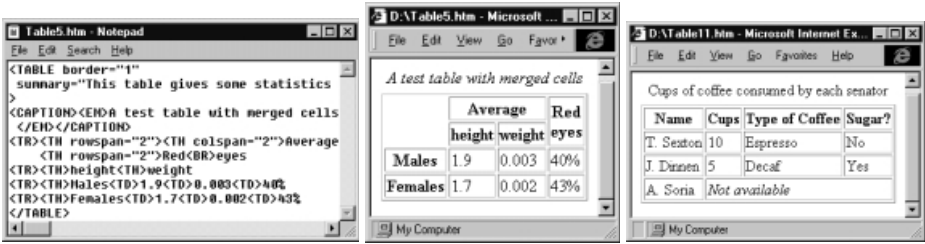
Rule 3. Given a container-content relationship $e_1 \leftarrow e_2 \leftarrow e_3$, where e_2 is neither ADDRESS, a table-specific element, nor DESCRIPTION attached by P, the *edge definition function* g of DAH produces $e_1 \leftarrow e_3$. \square

By now, the hierarchies of all the sibling elements in the DOH should have been refined. Any remaining HTML elements in the DOH, with the exception of ADDRESS and table-specific elements (to be discussed in Section 3.3), can be safely removed since they do not contribute to the determination of the data hierarchy of the given HTML document.

Consider the DOH in Figure 3(b). After applying Rule 3, LI[1], LI[2], LI[3], LI[4], and BODY are removed, and the resulting hierarchy is as shown in Figure 2(b) which is the resulting DAH for `utah.htm` since no HTML table is contained in `utah.htm`.

Rule 4. If a DOH which is modified by Rule 3 is a forest, then $t \leftarrow n_1, t \leftarrow n_2, \dots, t \leftarrow n_k$, where t is the content of the TITLE element of the source HTML document and n_1, \dots, n_k are the root nodes in DOH. \square

Rule 4 ensures that the resulting DAH is a tree. The content of the TITLE element is chosen to be the root in the DAH since it is the only required element in an HTML document H and provides an indication regarding what H is about.



(a) The source code of HTML Table 1 as shown in Figure 5(b) (b) HTML Table 1 with heading rows & columns (c) HTML Table 2 with a heading row

Fig. 5. Two typical types of HTML tables

3.3 Edge Definition Function for Table-Specific Elements

Among the table-specific elements, TR determines the number of rows, and TH and TD determine the number of columns in an HTML table (tables in short). THs are used for declaring table headings and TDs for asserting data of table cells. The data contents of TD elements are called *table data*. In contrast, the data contents of TH elements are column or row headings that are not considered as table data. However, data of either element is rendered in a Web browser.

There are a few popular types of HTML tables with respect to headings: (i) tables that have at least one heading row at the top and at least one heading column on the left, such as HTML Table 1 as shown in Figure 5(b) in [9]. We call this type of tables *column-row-wise*. (ii) Tables that contain at least one heading row at the top without heading columns, such as HTML Table 2 as shown in Figure 5(c) [9]. We call this type of tables *column-wise*. In a column-wise table, the heading rows yield the schema of the table. (iii) Other than these two types of tables, we notice that a large number of tables do not make use of table-related elements other than TABLE and TD. We, however, draw from our analysis a conclusion that the creator of this type of tables often implicitly designates the first row as a heading. Hence, we treat this type of tables as column-wise.

Two attributes of TH and TD, ROWSPAN and COLSPAN, play significant roles in determinating the data hierarchy of a table. Consider the source code of a table shown in Figure 5(a). It has four rows (i.e., four TRs) as rendered in Figure 5(b). (The source code shown in the figure is the original code which includes EM and BR before the approximation of the table is performed.) Note that the first TR contains three TH elements, implying that the row is a heading which includes three cells (i.e., three columns). The second row, on the other hand, contains two TH elements, implying that there are two heading cells, whereas each of the last two TRs contains one TH and three TDs, implying that there is one heading cell on the left followed by three data cells in each row. Apparently, the numbers of columns of the first two rows are not equal, nor with that of the last two rows. It is not clear which cell in one row belongs to the same column with a cell in another row until we take ROWSPANS and COLSPANS into consideration. When a TH or TD includes ROWSPAN="n" (COLSPAN="n", respectively), the associated cell is supposed to span n rows downward (n columns to the right, respectively).

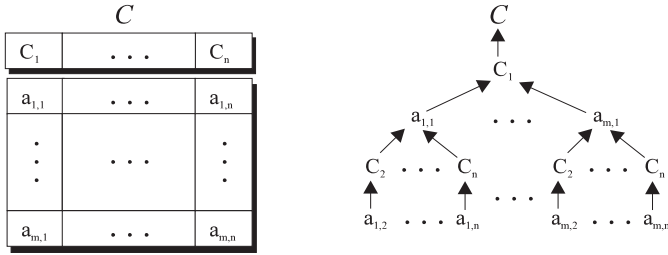


Fig. 6. A pseudo-table T and its corresponding DAH

We introduce the notion of pseudo-table since the properties of a pseudo-table are easy to understand and the mapping from a pseudo-table to a DAH is straightforward. A pseudo-table can be used to express either a column-row-wise table or a column-wise table with the table-specific elements mentioned above.

Definition 3. A *pseudo-table* $T = \{(a_{1,1}, \dots, a_{1,n}), (a_{2,1}, \dots, a_{2,n}), \dots, (a_{m,1}, \dots, a_{m,n})\}$ with column headings C_1, \dots, C_n and caption C , is a two-dimensional table, where each column heading C_i ($1 \leq i \leq n$) or table data $a_{i,j}$ ($1 \leq i \leq m$, $1 \leq j \leq n$) may be null. If a column heading or table data o is null, then the name of the object representing o is the empty string. Data values of rows i and j of the first column in T are different if $i \neq j$. \square

Recall that the hierarchy of HTML elements and data in a DOH is determined by their container-content relationships. The hierarchy of a pseudo-table, however, is defined over its caption, column headings and table data. Since column headings and table data may be null in a pseudo-table, we consider a special case: given the container-content relationships $o_1 \leftarrow o_2 \leftarrow o_3$, where o_i ($1 \leq i \leq 3$) is either a column heading or table data, $o_1 \leftarrow o_2 \leftarrow o_3$ is reduced to $o_1 \leftarrow o_3$ if the name of o_2 is the empty string.

Rule 5. Given an n -ary pseudo-table $T = \{(a_{1,1}, \dots, a_{1,n}), (a_{2,1}, \dots, a_{2,n}), \dots, (a_{m,1}, \dots, a_{m,n})\}$ with column headings C_1, \dots, C_n and caption C , the *edge definition function* g yields $V_R \leftarrow C_1 \leftarrow a_{i,1} \leftarrow C_j \leftarrow a_{i,j}$ ($1 \leq i \leq m$, $2 \leq j \leq n$), where V_R is labeled C if C is not empty, or is labeled “Table”, otherwise. \square

Since the caption of a pseudo-table T provides a short description on what the table is about, we choose the caption as the name for the root node of T (and hence the corresponding DAH). If CAPTION is missing, the root node of the corresponding pseudo-table is named “Table” simply to indicate the corresponding DAH is an HTML table. Furthermore, according to Rule 5, a DAH contains subtrees rooted at $a_{1,1}, \dots, a_{m,1}$ with the constraints $V_R \leftarrow C_1 \leftarrow a_{i,1}$ ($1 \leq i \leq m$). This is because each row in T can be uniquely identified from the other rows by the first column (i.e., $a_{i,1}$) in T (see Definition 3). Figure 6 shows the transformation between a pseudo-table and its corresponding DAH according to Rule 5. (Rule 5 indeed states how to construct the hierarchy of the components in a pseudo-table in order to obtain the corresponding DAH.) Figure 7 (Figure 8, respectively) shows how to map a column-wise HTML (column-row-wise, respectively) table to a pseudo-table.

Colspan and Rowspan. An HTML table may not have the same number of THs or TDs in each row, and COLSPANS and ROWSPANs play an important role in

$h_{1,1}$...	$h_{1,n}$		$C_1 =$...	$C_n =$
\vdots		\vdots		$h_{1,1} \dots h_{k,1}$...	$h_{1,n} \dots h_{k,n}$
$h_{k,1}$...	$h_{k,n}$				
$d_{k+1,1}$...	$d_{k+1,n}$	\longleftrightarrow	$a_{1,1} = d_{k+1,1}$...	$a_{1,n} = d_{k+1,n}$
\vdots		\vdots		\vdots		\vdots
$d_{k+m,1}$...	$d_{k+m,n}$		$a_{m,1} = d_{k+m,1}$...	$a_{m,n} = d_{k+m,n}$

A single dot (.) is the dot operator as presented in Definition 2.

Fig. 7. Mapping from a column-wise HTML table to a pseudo-table

$hh_{1,1}$...	$hh_{1,n}$		$C_1 =$...	$C_n =$
\vdots		\vdots		$hh_{1,1} \dots$		$hh_{1,n} \dots$
$hh_{k,1}$...	$hh_{k,n}$		$hh_{k,1}$		$hh_{k,n}$
$hv_{k+1,1}$...	$hv_{k+1,l}$	$d_{k+1,l+1} \dots d_{k+1,l+n-1}$	\longleftrightarrow	$a_{1,1} = hv_{k+1,1} \dots$	$a_{1,n} =$
\vdots		\vdots	\vdots		$\dots hv_{k+1,l}$	$d_{k+1,l+n-1}$
$hv_{k+m,1}$...	$hv_{k+m,l}$	$d_{k+m,l+1} \dots d_{k+m,l+n-1}$		\vdots	\vdots
					$a_{m,1} = hv_{k+m,1} \dots$	$a_{m,n} =$
					$\dots hv_{k+m,l}$	$d_{k+m,l+n-1}$

A single dot (.) is the dot operator as presented in Definition 2.

Fig. 8. Mapping from a column-row-wise HTML table to a pseudo-table

mapping such an HTML table to a pseudo-table. If a TH or TD contains COLSPAN = “n”, 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. Hence, at the current row, we insert $n-1$ cells to the right of the current cell and replicate the data content of the current cell $n-1$ times to the new cells. If a TH element contains ROWSPAN = “n”, the particular cell of the TH element is to be expanded to the next $n-1$ rows. For that we insert $n-1$ cells beneath the current TH cell in the current column, and push the data content h of the current cell all the way down to the $(n-1)$ th new cell, rather than replicating h to the underneath rows $n-1$ times. As a result, h appears at the $(n-1)$ th new cell, and each of the cells above the $(n-1)$ th new cell in the same column is left as the empty string. This is necessary for retaining the correct association among the table data across all the rows in a column. On the other hand, if ROWSPAN is contained in a TD, we insert $n-1$ new cells underneath the current TD cell, and replicate the data content of the TD to the inserted cells since each table data in different rows of the same column is meant to represent a data entry with the same content. After COLSPANS and ROWSPANS are processed, the corresponding table conforms to the specification of a pseudo-table as stated in Rule 5.

Consider the table in Figure 5(c) and suppose the cell at the i th row and the j th column is denoted by $cell(i, j)$. Note that “Not available” appears across the last three column spaces of the forth row of the table due to COLSPAN=“3” in cell(4, 2) (the source code is not included in this paper due to page limit), and hence we replicate “Not available” of cell(4, 2) to the next two inserted cells cell(4, 3) and cell(4, 4), in its corresponding pseudo-table as shown on the left in Figure 9.

In the following example, we demonstrate the process of constructing the DAH (via the pseudo-table) of the HTML table shown in Figure 5(b).

Cups of coffee consumed by each senator				A test table with merged cells			
Name	Cups	Type of Coffee	Sugar?		Average. height	Average. weight	Red eyes
T. Sexton	10	Espresso	No				
J. Dinnen	5	Decaf	Yes	Males	1.9	0.003	40%
A. Soria	Not available	Not available	Not available	Females	1.7	0.002	43%

Fig. 9. The pseudo-tables of the HTML tables in Figures 5(c) and 5(b)

Consider the source code in Figure 5(a). The first and third TH both contain ROWSPAN=“2”. Thus, the null data of the first TH is pushed down to the next row in the corresponding pseudo-table T . At a glance, it may look like that this action has no effect to the table since the pushed-down value is a null data. Indeed, with this action, the pushed-down null data is inserted into cell(2,1) in T and subsequently the TH with *height* (*weight*, respectively) is moved to the new location which is cell(2,2) (cell(2,3), respectively) in T . This is desirable since the correct association among “height,” “weight,” and other table data are now in place in T . In addition, the second TH contains COLSPAN = “2” and subsequently its data content “Average” is replicated once to the right in the same row in T , and “Red eyes”, which is originally in the third TH, is moved to the forth column and then pushed onto the forth column of the next row in T since ROWSPAN = “2”.

Recall that EM and BR, as shown in the source code in Figure 5(a), should have already been removed after the approximation of the table by Rule 5. Also, there are two heading rows in T and each C_i ($1 \leq i \leq 4$) is determined by the concatenation of the data contents of the two rows in the i th column. As a result, C_1 is the empty string, $C_2 = \text{Average.height}$, $C_3 = \text{Average.weight}$, and $C_4 = \text{Red eyes}$. Furthermore, the caption of T is the caption of the HTML table in Figure 5(a). The resulting pseudo-table T is as shown on the right in Figure 9.

We map the resulting pseudo-table T to the DAH. Since T contains the caption C , ‘A test table with merged cells’, C forms the root node of the DAH according to Rule 5. Next, consider C_1 . Since C_1 is empty, we skip C_1 in the hierarchy $V_R \leftarrow C_1 \leftarrow a_{i,1} \leftarrow \dots$ and create two child nodes of C by using $a_{1,1}$ (Males) and $a_{2,1}$ (Females). The rest of the cells $a_{i,j}$ ($1 \leq i \leq 2$, $2 \leq j \leq 4$) in the last two rows of T yield nodes and edges in DAH_T as follows: $a_{1,1} \leftarrow C_2 \leftarrow a_{1,2}$; $a_{1,1} \leftarrow C_3 \leftarrow a_{1,3}$; $a_{1,1} \leftarrow C_4 \leftarrow a_{1,4}$; $a_{2,1} \leftarrow C_2 \leftarrow a_{2,2}$; $a_{2,1} \leftarrow C_3 \leftarrow a_{2,3}$; $a_{2,1} \leftarrow C_4 \leftarrow a_{2,4}$. We render the resulting DAH in WebView [4], as shown under “[ROOT=E:\Table5.xml]” in the left pane of Figure 10. Note that the association of a table data D with another can be conceived by examining the context of D . For instance, ‘A test table with merged cells’.Males.Average.height.‘1.9’ captures the data hierarchy of “1.9”.

3.4 DAH to XML

According to the XML recommendation [3], an XML document consists of three consecutive blocks: (i) a prolog, (ii) a data block of one or more XML elements, and (iii) an optional miscellaneous block. The major components of a prolog include an XML declaration, followed by a document type declaration (DTD). A DTD provides the grammar of a class of XML documents, and the second block (i.e., *data block*), where the actual data are contained, must conform to the grammar. The optional third block contains miscellaneous data such as comments which is not the primary focus of this paper.

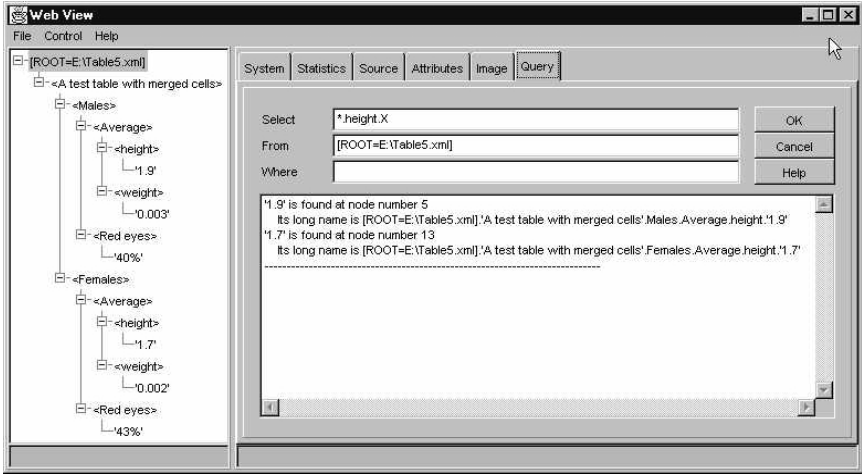


Fig. 10. DAH of the HTML table in Figure 5(a), rendered in WebView

An XML document X is *well-formed* if X contains an XML declaration and all the markups are properly nested with no overlap. Furthermore, a well-formed XML document X is *valid* if X includes the DTD with which X complies. Violations of these constraints are treated differently. If an XML document X violates the well-formedness constraint, *fatal* error is invoked and a normal process on X must terminate, whereas a violation of the validity constraint is only considered as an error, and the document can still be processed normally. Thus, our emphasis in generating XML documents is on well-formed XML documents.

The well-formedness property of the resulting XML document X converted from an HTML document is guaranteed by our Html2Xml approach using a stack machine Σ [4]. In our Html2Xml approach, the processing thread enters a new element whenever it processes a new node n in a DAH. For n , there are two cases to be considered: (i) n is a child node of the previously processed node n_0 , or (ii) n is a sibling node of n_0 . In case (i), the stack symbol of n is pushed on Σ , on top of the stack symbol of n_0 , and in case (ii) the stack symbol of n_0 is first popped and then the stack symbol of n is pushed onto Σ . Moreover, if n is a leaf node in the DAH, no stack operation is performed but the name of n is appended to X as the data content of the element whose stack symbol is still at the top of Σ . With that we guarantee that X is well-formed.

Document Type Declaration. DTD can be internal, external, or both. In our Html2Xml approach, we generate external DTDs. Using a stack machine Σ (as discussed earlier), an external DTD D for an XML document X is created while X is generated. At the beginning of the process, D is initialized by the text declaration `<?xml encoding="UTF-8"?>`, where "UTF-8" can be replaced by other encoding names such as "UTF-16" or "ISO-10646-UCS-4". As the nodes in a source DAH are processed and elements are identified for X , element declarations are appended to D . An element declaration is of the form `<!ELEMENT Name contentspec>`, where *Name* is the name of the element e

that is appended to X and *contentspec* is the placeholder for the content specification of e . In our Html2Xml approach, during the process of generating X , *contentspec* is replaced by (i) (EMPTY), if e is an empty element; (ii) (#PCDATA), if e contains a data content; or (iii) the list of the names of children of e , if e has child elements. We discuss further updating *contentspec* when e has child elements.

Given a node with name **node** in a DAH, there may exist more than one child node with the same name, such as **cnode**[1], **cnode**[2], ..., as discussed in Section 2. If so, **cnode** in *contentspec* of `<!ELEMENT node contentspec>` can be immediately followed by an occurrence symbol '?', '*' or '+', which denotes that the preceding content particle, i.e., **node**, of the respective symbol may occur once or more times (+), zero or more times (*), or at most once (?). Note that the implications of these three symbols are not distinct. For instance, if **cnode** occurs just once, either `<!ELEMENT node (cnode)?>`, `<!ELEMENT node (cnode)*>`, `<!ELEMENT node (cnode)+>`, or even `<!ELEMENT node (cnode)>` is a valid declaration. In our Html2Xml approach, (cnode) will be appended to the resulting XML document if **cnode** occurs just once in the source DAH, or (cnode)* will be appended if **cnode** occurs more than once.

Along with the update of *contentspec* in an element declaration, we need to consider attribute list declarations of an element e as well if e accompanies any attribute. When an element e is found to accompany attributes of the form $attr_i = attrVal_i$ ($i \geq 1$), this information is added after the element declaration of e in the DTD as `<!ATTLIST e attr_i AttType DefaultDecl>` for each $attr_i$, where *AttType* is CDATA which denotes that the corresponding attribute is of string type, and *DefaultDecl* is #IMPLIED which denotes that no default value is provided for the attribute. Note that $attr_i$ may appear more than once with different values $val_1, val_2, \dots, val_n$ ($1 \leq n$). In such a case, ($attrVal_i$) is first bound to (val_1), then to ($val_1 \mid val_2$) when val_2 is identified, and so forth while the DAH is processed. Eventually, ($attrVal_i$) is bound to ($val_1 \mid val_2 \mid \dots \mid val_n$).

Applying algorithm Dah2Xml³ to the DAH in Figure 2(b) yields the resulting XML document X as shown below.

```
<?xml version="1.0" ?>
<!DOCTYPE UtahCountyDemographicAnalysis1996 SYSTEM
  "UtahCountyDemographicAnalysis1996.dtd">
<UtahCountyDemographicAnalysis1996 AddressDescription="Comments to
  webmaster" AddressLink="webmaster.htm">
  <DESCRIPTION>Utah County has a population of 317,880--...</DESCRIPTION>
  <Orem Link="orem.htm"><Population:79,736/><GrowthRate:2.20%/></Orem>
  <Alpine Link="alpine.htm"><Population:5,161/><GrowthRate:4.60%/></Orem>
</UtahCountyDemographicAnalysis1996>
```

³ Algorithm Dah2Xml is not included in this paper but can be found in [5].

Also, the resulting DTD D of the DAH is shown below.

```
<?xml encoding="UTF-8"?>
<!ELEMENT UtahCountyDemographicAnalysis1996 (DESCRIPTION, Orem, Alpine)>
<!ATTLIST UtahCountyDemographicAnalysis1996 AddressDescription CDATA #IMPLIED>
<!ATTLIST UtahCountyDemographicAnalysis1996 AddressLink CDATA #IMPLIED>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ELEMENT Orem (Population:79,736, GrowthRate:2.20%)>
<!ATTLIST Orem Link CDATA #IMPLIED>
<!ELEMENT Population:79,736 (EMPTY)>
<!ELEMENT GrowthRate:2.20% (EMPTY)>
<!ELEMENT Alpine (Population:5,161, GrowthRate:4.60%)>
<!ATTLIST Alpine Link CDATA #IMPLIED>
<!ELEMENT Population:5,161 (EMPTY)>
<!ELEMENT GrowthRate:4.60% (EMPTY)>
```

Recall that ‘Utah County Demographic Analysis 1996’ is the root node of the DAH. It forms the root element of X and D is named after it by default. Since ADDRESS is treated as an attribute list of its parent element in X , the subtree rooted at ADDRESS is converted to the AddressDescription and AddressLink attributes of the root in X . In addition, DESCRIPTION, Orem, and Alpine are attached to X as the immediate contents of the root since they are child nodes of the root in the DAH. Note that the child node of DESCRIPTION in the DAH is a leaf node, and hence the *contentspec* of the DESCRIPTION element is declared as (#PCDATA) in D , and the *contentspec* of any leaf node is (EMPTY). Also, each HREF node is converted as an attribute to its parent element.

As another example, the XML document and its DTD of the DAH as shown in the left pane of Figure 10, where table-specific elements are involved, are

```
<?xml version="1.0" ?>
<!DOCTYPE ATestTableWithMergedCells SYSTEM "ATestTableWithMergedCells.dtd">
<ATestTableWithMergedCells>
  <Males>
    <Average><height>1.9</height><weight>0.003</weight></Average>
    <RedEyes>40%</RedEyes>
  </Males>
  <Females>
    <Average><height>1.7</height><weight>0.002</weight></Average>
    <RedEyes>43%</RedEyes>
  </Females>
</ATestTableWithMergedCells>
<?xml encoding="UTF-8"?>
<!ELEMENT ATestTableWithMergedCells (Males, Females)>
<!ELEMENT Males (Average, RedEyes)>
<!ELEMENT Average (height, weight)>
<!ELEMENT RedEyes (#PCDATA)>
<!ELEMENT height (#PCDATA)>
<!ELEMENT weight (#PCDATA)>
<!ELEMENT Females (Average, RedEyes)>
```

3.5 Implementation of the Html2Xml Approach

Portions of the proposed conversion approach, which includes the approximation (discussed in Section 3.1) and table-specific elements (discussed in Section 3.3),

have been implemented as a Java class using JDK 1.1.7. The Java class generates the data hierarchy of any given HTML table T in the lexical form whose representation is similar to the ordered list of the data context in T . For rendering and querying DOHs and DAHs graphically, WebView [4] can be used.

4 Concluding Remarks

We have presented a heuristic approach to convert HTML documents to XML documents. XML is rapidly emerging but there are still numerous existing HTML documents. It is desirable to convert these HTML documents to XML documents rather than to obsolete them or maintain documents of two different specifications. During the process of conversion, we eliminate all the HTML elements in a source HTML document from the resultant XML document. The approach for constructing the data hierarchy, as a part of our conversion approach, can be used for extracting the data hierarchy of an HTML document. Our approach can be used by existing wrappers and integrators (in data warehousing systems), which frequently access large number of HTML tables for extracting hierarchically structured data to automate the data acquisition process for XML repositories.

References

1. G. Arocena and A. Medelzon. WebOQL: Restructuring Documents, Databases and Webs. In *Proceedings of the 14th ICDE*, 1998.
2. P. Atzeni and G. Mecca. Cut and Paste. In *Proceedings of the 16th Intl. Symposium on Principles of Database Systems*, pages 144–153, 1997.
3. T. Bray, J. Paoli, and C. Sperberg-McQueen. Extensible Markup Language (XML) 1.0 W3C Recommendation 10-February-1998, February 1998.
4. S.-J. Lim and Y.-K. Ng. WebView: A Tool for Retrieving Internal Structures and Extracting Information from HTML Documents. In *Proceedings of the 6th International Conference on Database Systems for Advanced Applications (DASFAA'99)*, pages 71–80, April 1999.
5. S.-J. Lim and Y.-K. Ng. A Heuristic Approach for Converting HTML Documents to XML Documents. <http://lunar.cs.byu.edu/paper.html>, July 2000.
6. A. Mendelzon, G. Mihaila, and T. Milo. Querying the World Wide Web. In *Proceedings of the Conf. on Parallel & Distributed Info. Systems*, 1996.
7. A. Mendelzon and T. Milo. Formal Models of Web Queries. In *Proceedings of the 16th Intl. Symposium on PODS*, pages 134–143, 1997.
8. D. Raggett. HTML 3.2 Reference Specification. <http://www.w3.org/TR/REC-html32>, January 1997.
9. D. Raggett, A. Hors, and I. Jacobs. HTML 4.0 Specification – W3C Recommendation. <http://www.w3.org/TR/REC-html40>, April 1998.
10. A. Sahuguet and F. Azavant. Looking at the Web through XML-glasses. In *Proceedings of the 4th Intl. Conf. on Cooperative Info. Systems*, 1999.