In Proceedings of 19th International Conference on Conceptual Modeling (ER 2000), Salt Lake City, USA, October 9-12, 2000. LNCS. Springer-Verlag.

A Conceptual Model for the Web

Mengchi Liu[1] and Tok Wang Ling[2]

[1] Department of Computer Science, University of Regina
Regina, Saskatchewan, Canada S4S 0A2
mliu@cs.uregina.ca

[2] School of Computing, National University of Singapore
Lower Kent Ridge Road, Singapore 119260
lingtw@comp.nus.edu.sg

Abstract. Most documents available over the web conform to the HTML specification. Such documents are hierarchically structured in nature. The existing graph-based or tree-based data models for the web only provide a very low level representation of such hierarchical structure. In this paper, we introduce a conceptual model for the web that is able to represent the complex hierarchical structure within the web documents at a high level that is close to human conceptualization/visualization of the documents. We also describe how to convert HTML documents based on this conceptual model. Using the conceptual model and conversion method, we can capture the essence (i.e., semistructure) of HTML documents in a natural and simple way.

1 Introduction

Most documents available over the web conform to the HTML specification. They are intended to be human readable through a browser and thus are constructed following some common conventions and often exhibit some hierarchical structure. How to represent such documents at a conceptual level is an important issue.

In the past few years, a number of data models have been developed in the database community to retrieve data from the web, such as UnQL [2], OEM [12], Strudel [4], etc. For a survey, see [6]. These proposals mainly use relational, graph-based or tree-based data models to represent the web data. They focus on inter-document structures, with little attention to intra-document structures and thus can only represent the web data at a very low level. For example, none of the existing data models and languages can represent Michael Ley's DBLP bibliography web document at `http://www.informatik.uni-trier.de/~ley/db` in a natural and simple way.

Because of the difficulties with HTML, XML [1] has been proposed to uniformly represent data over the Internet. It provides natural support for describing the hierarchical structure in web documents and allows specific markup to be created for specific data. It may become as important and as widely used as HTML in the future.

The World Wide Web Consortium (W3C) has recently recommended the Document Object Model (DOM) [14] as an application programming interface for HTML and XML documents. DOM defines the logical structure of documents and the way a document is accessed and manipulated. It represents documents as a hierarchy of various node objects. With DOM, programmers can access, change, delete, add or build HTML or XML documents, and navigate their structure. Nevertheless, DOM is a very low level data model for HTML/XML documents as it is intended for programmers to write programs to access and manipulate HTML and XML documents, rather than for the user to find the information within the documents.

W3C has also recommended XPath [3] as a language for operating on the abstract, logical structure of an XML document, rather than its surface syntax. It models an XML document as a tree of various nodes at a level higher than DOM, such as element nodes, attribute nodes, text nodes, etc.

Before XML becomes widely used as HTML, a powerful data model to describe the abstract, logical structure of HTML documents and a flexible language to query large HTML documents is still in high demand.

In this paper, we propose a conceptual model for HTML that is able to represent any complex hierarchical structure within the web documents at a high level that is close to human conceptualization/visualization of the documents. We also describe how to convert HTML documents based on this conceptual model.

The rest of the paper is organized as follows. Section 2 introduces our conceptual model for the web. Section 3 presents a set of rules to automatically structure HTML documents. Section 4 shows how to use the rules to convert HTML documents. Section 5 summarizes and points out further research issues.

2 Conceptual Model for the Web

In this section, we introduce our conceptual model for the web. We note that this conceptual model is very simple with only a few constructs but powerful enough to represent any HTML documents. First, we assume the existence of two kinds of symbols: a set $\mathcal{U}$ of urls, and a set $\mathcal{S}$ of strings. Note that $\mathcal{U}$ is a subset of $\mathcal{S}$.

Each web document in HTML is considered structured in our conceptual model. They have a title and a body. Our purpose is to extract conceptual structures within web documents. To this end, we ignore the features that are used to enhance the visual aspects of the web documents, such as fonts, colors, style, etc.

The notion of *objects* in our conceptual model is defined recursively as follows.

(1) A string $s \in \mathcal{S}$ is a *lexical* object.
(2) Let o be an object and $u \in \mathcal{U}$. Then $o\langle u\rangle$ is a *linking* object, and o is called the *label* and u is called the *anchor* of the linking object.
(3) Let a, o be objects. Then $a \Rightarrow o$ is an *attributed* object, and a is called the *attribute* and o is called the *value* of the attributed object.

(4) If $o_1, ..., o_n$ are distinct objects with $n > 1$, then $o_1|...|o_n$ is a *or-value* object.
(5) If $o_1, ..., o_n$ are objects with $n > 1$, then $\{o_1, ..., o_n\}$ is a *list* object.

In an object-relational database, we can have homogeneous tuples and sets. Tuple elements can be accessed using attribute names while set elements are directly accessed. In a web document, we may have attributed objects together with other objects. Thus, it is sometimes impossible to distinguish tuples from sets. In addition, web documents conceptually supports lists instead of sets as duplication is allowed and the order of elements exists. Thus, we use list objects for tuples, sets and lists in our conceptual model. The attribute names can be used to access components of list objects that matches them.

An or-value object is used to indicate that the intended value is one of them. It was first used in [10] and then extended in [9] to record conflicting information when manipulating semistructured objects.

The following are examples of objects:

Lexical objects:	*Computer Science, Database Systems*
Linking objects:	$Faculty\langle faculty.html\rangle$, $Jim\ Carter\langle /faculty/carter/\rangle$
Attributed objects:	$Title \Rightarrow CS\ Department$, $Location \Rightarrow Europe$
Or-value objects:	$Female|Male$, $15|20|23$
List object:	$\{Title \Rightarrow CS\ Dept,\ Research\langle research.html\rangle\}$

In our conceptual model, a web document is modeled as a list object. As we access a web document using its URL, we treat the URL together with its web document as a web object. We formalize it as follows.

Let u be a URL and t a list object. Then $u : t$ is a web object.

The following is an example of web object:

csdept.html : {
  Title ⇒ *CSDept*,
  People⟨*people.html*⟩⇒{
    Faculty⟨*faculty.html*⟩,
    Staff⟨*staff.html*⟩,
    Students⟨*students.html* ⟩},
  Programs ⇒{
    Ph.D Program⟨*phd.html*⟩,
    M.Sc Program⟨*msc.html*⟩,
    B.Sc Program⟨*bsc.html*⟩},
  Research⟨*research.html*⟩
}

Note that a linking object is different from a web object even though it is somewhat similar to a web object. A web object corresponds to a web document while a linking object is simply part of a web object.

Example 1. Part of Michael Ley's latest DBLP bibliography server web document at `http://www.informatik.uni-trier.de/~ley/db` can be represented in our conceptual model with simplified URLs such as a_1, b_1 etc. to fit in the paper as follows:

http://www.informatik.uni-trier.de/$\sim$*ley/db* : {
Title $\Rightarrow$*DBLP Bibliography*,
Body $\Rightarrow${
Search $\Rightarrow${*Author*$\langle a_1 \rangle$, *Title*$\langle a_2 \rangle$, *Advanced*$\langle a_3 \rangle$}, *Home Page Search*$\langle a_4 \rangle$},
Bibliographies $\Rightarrow$ {
Conferences$\langle b_1 \rangle$$\Rightarrow${*SIGMOD*$\langle b_{11} \rangle$, *VLDB*$\langle b_{12} \rangle$, *PODS*$\langle b_{13} \rangle$, *EDBT*$\langle b_{14} \rangle$},
Journals$\langle b_2 \rangle$ $\Rightarrow${*CACM*$\langle b_{21} \rangle$, *TODS*$\langle b_{22} \rangle$, *TOIS*$\langle b_{23} \rangle$, *TOPLAS*$\langle b_{24} \rangle$},
Series$\langle b_3 \rangle$ $\Rightarrow${*LNCS/LNAI*$\langle b_{41} \rangle$, *DISDBIS*$\langle b_{42} \rangle$},
Books $\Rightarrow${*Collections*$\langle b_{51} \rangle$, *DB Textbooks*$\langle b_{52} \rangle$},
By Subjects$\langle b_4 \rangle$ $\Rightarrow${*Database Systems*$\langle b_{61} \rangle$, *Logic Prog*$\langle b_{62} \rangle$, *IR*$\langle b_{63} \rangle$}},
Full Text $\Rightarrow$*ACM SIGMOD Antholog*$\langle c_1 \rangle$,
Reviews $\Rightarrow$*ACM SIGMOD Digital Review*$\langle c_2 \rangle$,
Links $\Rightarrow${
Research Groups $\Rightarrow${*Database Systems*$\langle d1 \rangle$, *Logic Programming*$\langle d2 \rangle$},
Computer Science Organization$\langle e_1 \rangle$ $\Rightarrow${
ACM$\langle e_{11} \rangle$ (*DL*$\langle e_{12} \rangle$, *SIGMOD*$\langle e_{13} \rangle$, *SIGIR* $\langle e_{14} \rangle$),
IEEE Comptuer Society$\langle e_{15} \rangle$$\Rightarrow$*DL*$\langle e_{16} \rangle$}
Related Services $\langle f_1 \rangle$ $\Rightarrow${
CoRR$\langle f_{11} \rangle$, *ResearchIndex*$\langle f_{12} \rangle$, *NZ-DL*$\langle f_{13} \rangle$,
CS BibTex$\langle f_{14} \rangle$, *HBP*$\langle f_{15} \rangle$, *Virtual Library* $\langle f_{16} \rangle$}}
}

As the above examples show, our conceptual model provides an intuitive representation of the complex hierarchical structure within a web document at a high level that is close to human conceptualization/visualization of the documents. None of the existing data models can do so in such a simple way.

3 Converting HTML Documents

Web documents are intended to be human readable though a browser. They are constructed following some common conventions and often exhibit some hierarchical structure. By examining a large number of web documents, we have discovered a set of general rules that can be used to convert web documents into web objects in our conceptual model. We now introduce our conversion method. In particular, we focus on HTML 4.01 [13], the latest version of HTML.

Web documents are not parsed by browsers and may have syntactic errors. To make our presentation simple, we assume that web documents are syntactically correct. Our conversion method is mainly based on the contents shown by a web browser and therefore we ignore the features that are used to enhance the visual aspects of the web documents, such as fonts, colors, style, etc. We assume that there is an HTML interpreter that can be used to parse the HTML files, remove optional tags and irrelevant components, and automatically identify the corresponding components for our definition below in the web document based on various tags, such as <p>, <h*n*>,
, etc.

A web document in HTML starts with <html> tag and ends with </html> tag. It normally consists of two parts: *head* and *body*. The head mainly specifies the *title* of the document while the body specifies the *contents* of the web document. Web documents can be classified into two kinds based on the kind of the contents: *Frame-based web documents* whose body starts with <frameset> and ends with </frameset>, and *Regular web documents* whose body starts with <body> and ends with </body>.

The purpose of frame-based web documents is to allow the visitor to see more than one page at a time, without completely cluttering up their screen. Frame-based web documents have a number of frames in their body and each frame contains its own web document. A frame-based web document has the following form, where F_i is a frame or a nested frameset:

<html><head><title>T</title></head>
<frameset ...> $F_1, ..., F_n$ </frameset>
</html>

Let u be the URL of a web document of the above form, C a converter that converts components of a web document, and $\uplus$ list concatenation operator. Then we can convert a frame-based web document into a web object as follows:

$$u : \{Title \Rightarrow T, Frameset \Rightarrow C(F_1) \uplus ... \uplus C(F_n)\}$$

For each frame $F =$ <frame name = "N" src = "U"> where N is the name and U is the url of the frame, we convert it into $C(F) = \{N\langle U\rangle\}$. For a frameset $FS =$ <frameset ...>$F_1, ..., F_n$</frameset>, we convert it recursively into $C(FS) = C(F_1) \uplus ... \uplus C(F_n)$.

In other words, we convert a possibly nested frameset into a list object and each frame in a frameset into a linking object in the list object.

Regular web documents have a number of sections in their body with the following form, where $S_1, ..., S_n$ are sections:

<html><head><title>T</title></head>
<body> S_1 ... S_n </body>
</html>

Let u be the URL of a web document of the above form and C a converter that converts components of a web document. Then we can convert a regular web document into a web object as follows:

$$\begin{aligned} u\text{: } \{&Title \Rightarrow T, \\ &Body \Rightarrow \{C(S_1), ..., C(S_n)\}\} \end{aligned}$$

3.1 Converting Sections

In HTML documents, there can be two kinds of sections: with a heading and without a heading. If it has a heading, then we can convert it into an attributed objects with the heading as its attribute and the rest as its value. Formally, Let $S =$ <hn>H</hn>T be a section with a heading H and contents T. Then

$$C(S) = C(H) \Rightarrow C(T)$$

If a section has no heading $S = T$, then $C(S) = C(T)$.

Example 2. Consider the HTML document with URL *http://www.cs.uregina.ca* as follows:

```
<html>
<head><title>Computer Science Department</title></head>
<body>
<h2>History</h2>
The department was founded in 1970.
<h2>Programs</h2>
The department offers B.Sc. M.Sc. and Ph.D degrees
<h2>Facilities</h2>
The department has up to date equipment and software
</body>
</html>
```

This HTML document has 3 sections with headings. We can convert it into a web object as follows:

http://www.cs.uregina.ca: {
  Title ⇒ *Computer Science Department,*
  Body ⇒{
    History ⇒ *The department was founded in 1970,*
    Programs ⇒ *The department offers B.Sc. M.Sc. and Ph.D degrees,*
    Facilities ⇒ *The department has up to date equipment and software*}
  }

Within a section, there may be a sequence of paragraphs, lists, tables, etc. In what follows, we discuss how to translate each of them.

3.2 Converting Paragraphs

A section in HTML may have a sequence of paragraphs. Let $S = P_1, ..., P_n$ be a section where $P_1, ..., P_n$ are paragraphs. Then we convert it as follows:

$$\begin{array}{ll} C(S) = \{C(P_1), ..., C(P_n)\} & \text{when } n > 1 \\ C(S) = C(P_1) & \text{when } n = 1 \end{array}$$

A paragraph in a section starts with a tag `<p>`. Some paragraphs may have a emphasized beginning that is usually bold, italic, etc. or is followed by a colon ':'. Let $P =$ <t>B</t> $: R$ or $P =$ <t>$B :$ </t>R be such a paragraph, where t is either `b`, `i`, `em`, or `strong`. Then we convert it as follows:

$$C(P) = C(B) \Rightarrow C(R)$$

If a paragraph has no emphasized beginning $P = R$, then $C(P) = C(R)$.

Let R be the part of a paragraph other than the emphasized heading. If R has logical parts $R_1, ..., R_n$ with $n \geq 1$, then

$$\begin{array}{ll} C(R) = \{C(R_1), ..., C(R_n)\} & \text{if } n > 1 \\ C(R) = C(R_1) & \text{if } n = 1 \end{array}$$

Each logical part is converted as a paragraph recursively.

Example 3. The following is a section of CIA World Factbook page about Canada at `http://www.odci.gov/cia/publications/factbook/`. It consists of a sequence of paragraphs.

```
<p>
<b>Location:</b> Northern North America
<p>
<b>Geographic coordinates:</b> 60 00 N, 95 00 W
<p>
<b>Map references:</b> North America
<p>
<b>Area:</b>
<br><i>total:</i> 9,976,140 sq km
<br><i>land:</i> 9,220,970 sq km
```

We can convert it into the following list object:

{*Location* ⇒*Northern North America,*
Geographic coordinates ⇒*60 00 N, 95 00 W*
Map references ⇒*North America,*
Area ⇒{*total* ⇒*9,976,140 sq km, land* ⇒*9,220,970 sq km*}}

3.3 Converting Multimedia Features and Hypertexts

In HTML documents, we can include multimedia features such as images, applets, video clips, sound clips, etc. In our conceptual model, we convert them into attributed objects using keywords *Image, Applet, Video, Sound*, etc. as their attributes.

For an image link $I =$ `<img src = "`U`" alt="`T`">`, where U is a URL and T is a string, we convert it as follows:

$$C(I) = Image \Rightarrow T\langle U \rangle.$$

The cases for other multimedia features are handled in the similar way. Due to space limitation, we omit them here.

For a hypertext link $H =$ `<a href = "U">`T`</a>`, where U is a URL and T is a string, we convert it into a linking object:

$$C(H) = C(T)\langle U \rangle.$$

For each character string S, $C(S) = S$.

Consider the following paragraph without emphasized heading:

```
<a href="dick.html">Dick</a> likes <a href="jane.html">Jane</a>
```

It has three logical units and thus can be converted into the following object:

{*Dick*⟨*dick.html*⟩*, likes, Jane*⟨*jane.html*⟩}

3.4 Converting Lists

HTML supports three kinds of lists: ordered lists, unordered lists and definition lists. One can also nest one kind of list inside another.

For an unordered list $L =$ `<ul><li>`L_1`...<li>`L_n`</ul>` or an ordered list $L =$ `<ol><li>`L_1`...<li>`L_n`</ol>`, we convert it into a list object as follows:

$$
\begin{array}{ll}
C(L) = \{C(L_1), ..., C(L_n)\} & \text{when } n > 1 \\
C(L) = C(L_1) & \text{when } n = 1
\end{array}
$$

For a definition list $L =$ <dl><dt>N_1<dd>L_1...<dt>N_m<dd>L_m</dl>, we convert it into a list object as follows:

$$
\begin{array}{ll}
C(L) = \{C(N_1) \Rightarrow C(L_1), ..., C(N_m) \Rightarrow C(L_m)\} & \text{when } n > 1 \\
C(L) = C(N_1) \Rightarrow C(L_1) & \text{when } n = 1
\end{array}
$$

For a nested ordered or unordered list item $I = T$<l>
T_1 ...<li>T_m</l>, where l is either ol or ul, we convert it into a nested list objects;

$$C(I) = C(T) \Rightarrow \{C(T_1), ..., C(T_m)\}$$

Example 4. Consider the following HTML document containing nested lists with URL research.html:

```
<html>
<head><title>CS Department Research</title></head>
<body>
<h2>Research Areas</a></h2>
<ol>
<li>Artificial Intelligence
<ul>
<li>Cognitive Science <li>Linguistics <li>Reasoning
</ul>
<li>Database Systems
<ul>
<li>Query Processing  <li>Data Models <li>Active Databases
</ul></ol>
<h2>Research Groups and Labs</a></h2>
<ol>
<li>Programming Languages
<li>Intelligent Systems
<li>Natural Language Lab
</ol>
</body>
</html>
```

We can convert it into a web object in our conceptual model as follows:

research.html : {
  Title $\Rightarrow$ *CS Department Research,*
  Body $\Rightarrow${
    Research Areas $\Rightarrow${
      Artificial Intelligence $\Rightarrow${*Cognitive Science, Linguistics, Reasoning*},
      Database Systems $\Rightarrow${*Query Processing, Data Models, Active Databases*}},
    Research Groups and Labs $\Rightarrow${
      Programming Languages,
      Intelligent Systems,
      Natural Language Lab}}
  }

Example 5. Consider the following HTML document containing definition lists with URL `index.html`.

```
<html>
<head><title>Graduate Studies in CS</title></head>
<body>
<h2></h2>
<dl>
<dt>General Information
<dd>The department was established in 1970
<dt>Programs of Study
<dd>It offers M.Sc, and Ph.D in Computer Science
<dt>Financial Support
<dd>A variety of scholarships are available
<dt>Facilities
<dd>The research labs have all kinds of state-of-the-art equipment
</dl>
</body>
</html>
```

We convert it as follows:

index.html : {
  Title ⇒ *Graduate Studies in CS,*
  Body ⇒{
    General Information ⇒ *The department was established in 1970*,
    Programs of Study ⇒ *It offers M.Sc, and Ph.D in Computer Science*,
    Financial Support ⇒ *A variety of scholarships are available*,
    Facilities ⇒ *The research labs have all kinds of state-of-the-art equipment*}
  }

3.5 Converting Tables

Tables in HTML are used to arrange data into rows and columns of cells. Tables may have caption and column/row headings. Tables can be used in two different ways: to present a list with a better layout; to present real tabular information. We convert tables as follows.

For a table with a caption $T =$ `<table><caption>`H`</caption>`TC`</table>` where H is the caption and TC is the table contents

$$C(T) = C(H) \Rightarrow C(TC)$$

For a table without a caption $T =$ `<table>`TC`</table>`, $C(T) = C(TC)$

Let $R_1, ..., R_n$ be rows other than the row for column headings in the table contents TC. Then

$$C(TC) = \{C(R_1), ..., C(R_n)\}$$

Each row R_i is converted as follows:

1. If the table has column headings $H_1, ..., H_n$ and each row R has a row heading H, $R = \texttt{<tr><th>}H\texttt{<td>}C_1\texttt{...<td>}C_n$, then
   $C(R) = C(H) \Rightarrow \{C(H_1) \Rightarrow C(C_1), ..., C(H_n) \Rightarrow C(C_n)\}$
2. If the table has column headings $H_1, ..., H_n$, but each row R has no row heading $R = \texttt{<tr><td>}C_1\texttt{...<td>}C_n$, then
   $C(R) = \{C(H_1) \Rightarrow C(C_1), ..., C(H_n) \Rightarrow C(C_n)\}$
3. If the table does not have column headings but each row R has a row heading $R = \texttt{<tr><th>}H\texttt{<td>}C_1\texttt{...<td>}C_n$,
   $C(R) = C(H) \Rightarrow \{C(C_1), ..., C(C_n)\}$
4. If the table has no column and row headings, then for each row $R = \texttt{<tr>}C_1\texttt{...<td>}C_n$,
   $C(R) = \{C(C_1), ..., C(C_n)\}$

Example 6. Consider the following table in an HTML document:

```
<table>
<caption align = top>Bear Sightings</caption>
<tr>
<td><br><th>Babies<th>Adults<th>Total
<tr>
    <th>Northampton<td>2<td>4<td>6
<tr>
    <th>Becket<td>5<td>22<td>27
<tr>
    <th>Worthington<td>7<td>5<td>12
</table>
```

We can convert it into the following object based on Rule (13) Case 1:

$$
\begin{array}{l}
\textit{Bear Sightings} \Rightarrow \{ \\
\quad \textit{Northampton} \Rightarrow \{\textit{Babies} \Rightarrow \textit{2},\ \textit{Adults} \Rightarrow \textit{4},\ \ \textit{Total} \Rightarrow \textit{6}\}, \\
\quad \textit{Becket} \Rightarrow \qquad\ \{\textit{Babies} \Rightarrow \textit{5},\ \textit{Adults} \Rightarrow \textit{22},\ \textit{Total} \Rightarrow \textit{27}\}, \\
\quad \textit{Worthington} \Rightarrow \{\textit{Babies} \Rightarrow \textit{7},\ \textit{Adults} \Rightarrow \textit{5},\ \ \textit{Total} \Rightarrow \textit{12}\}\}
\end{array}
$$

In HTML documents, tables without column headings and row headings are often used to arrange items for visual effects. Consider the following portion of an HTML document:

```
<h2 align="center"> Faculty Profiles</h2>
<table border="0" cellpadding=3 cellspacing=3 align=center>
<tr>
<td colspan=50% align=left>
<a href="/faculty/bunt/">Rick Bunt</A>
<td colspan=50% align=left>
<a href="/faculty/carter/">Jim Carter</A>
<tr>
<td colspan=50% align=left>
<a href="/faculty/cheston/">Grant Cheston</A>
...
</table>
```

We can convert it into the following object:

Faculty Profiles $\Rightarrow${
Rick Bunt$\langle$*/faculty/bunt/*$\rangle$,
Jim Carter$\langle$*/faculty/carter/*$\rangle$,
Grant Cheston$\langle$*/faculty/cheston/*$\rangle$,
... }

Using our conversion method, we can convert part of Michael Ley's latest DBLP web document at `http://www.informatik.uni-trier.de/~ley/db` into our conceptual model as shown in Example 1.

3.6 Converting Forms

HTML forms enable visitors to communicate with the web server. There are two basic parts of a form: the structure that consists of fields, labels and buttons that the visitor sees and fills out on a page, and the processing part that process the information the visitor fills out using CGI script typically written in Perl or some other programming language.

Conceptually, we are interested in the structure part; that is, what kind of information is presented and can be communicated with the web server. Each element on a form has a name and a value/type associated with it. The name identifies the data that is being sent, the value is the data that is built-in the HTML document, the type specifies that value that comes from the visitor.

HTML supports two ways to send information to the web server: *GET* and *POST*. The GET method appends the name-value pairs to the end of the URL and is for small amount of data. The POST method sends a data file with the name-value pairs to the server's standard input without size limit.

Using either GET or POST is mainly a physical level concern and does not have much conceptual value.

A form normally has two special buttons: *submit* and *reset*. The *submit* button is used to send the information and the *reset* button is used to reset the form.

Let F be a form as follows:

```
F = <form ...>
    S
    <input type = "submit" ...>
    <input type = "reset"...>
    </form>
```

We convert it as follows:

$$C(F) = FORM \Rightarrow C(S)$$

Note that we don't keep information about the *submit* and *reset* buttons in the result as we make *FORM* a reserved attribute in our conceptual model to indicate that its value are used for communication with the web server and those two buttons are implied.

For a text field of the form $TF = L$:`<input type = "`T`" name = "`N`">`, we convert it into

$$C(TF) = L \Rightarrow T$$

Note that T is a build-in type and is used to indicate that the form will communicate value of type T to the web server for the label L. Also note that the name N does not occur in our conceptual model as it is the internal representation of L and the visitor cannot see it. In practice. N and L are the same quite often.

Text areas in HTML allows the user to enter more text than can fit in the a text field. Let $TA =$ `<textarea name = "`N`" ... >T</textarea>` be a text area, we convert it into

$$C(TA) = N \Rightarrow T$$

Radio buttons on forms are used to allow the user to make only one choice. In our conceptual model, we use an or-value to indicate the choices. For a group of radio buttons of the forms:

$R =$ `<input type = "radio" name = "`N_1`" value = "`V_1`">`L_1
...
`<input type = "radio" name = "`N_m`" value = "`V_m`">`L_m

we convert it into the following or-value object:

$$C(R) = L_1 \mid ... \mid L_m$$

While radio buttons can accept only one answer per set, Check boxes allow the visitor to check as many check boxes as they like. For a group of check boxes of the form:

$B =$ `<input type = "checkbox" name = "`N_1` value = "`V_1`">`L_1
...
`<input type = "checkbox" name = "`N_m` value = "`V_m`">`L_m

we convert it into the following list of or-value object to indicate that the value can be a list:

$$C(R) = \{L_1 \mid ... \mid L_m\}$$

Menus allow visitors of HTML document to enter information easily. For a menu of the form:

$N =$ `<select name = "`N`" ...>`
`<option value = "`V_1`">`L_1
...
`<option value = "`V_n`">`L_n
`</select>`

We convert it into the following attributed or-value object:

$$C(N) = N \Rightarrow L_1 \mid ... \mid L_n$$

Example 7. Consider the following portion of an HTML document:

```
<form method=post action="http://site.com/cgi-bin/get_menu"><br>
<p>Name:<input type="text" name="Name">
<p><strong>Age</string><br>
<select name = "Age" size ="5">
<option value = "18-65">18-65
<option value = "66-10o">66-100
</select>
<p>Gender:
<input type = "radio" name = "Gender" value = "female">Female
<input type = "radio" name = "Gender" value = "male">Male <br>
<input type = "submit" value = "Send info">
<input type = "reset" value = "Start over">
</form>
```

We convert it into the following object:

Form $\Rightarrow$\{
Name $\Rightarrow$*text*,
Age $\Rightarrow$18-65|66-100,
Gender $\Rightarrow Female|Male\}$

4 Conclusion

The main contribution of the paper is the following. First, we have proposed a conceptual model for capturing, in a natural and simple way, the internal structure of HTML documents. Our conceptual model is close to human conceptualization/visualization of the documents. Unlike the data models that deal with the hierarchical structure of web documents at a very low level using trees and graphs such as WebOQL, OEM, UnQL, Strudel, our conceptual model only have a few simple but powerful high level constructs that can be used to best describe the contents in the web documents. Also, using trees and graphs cannot support ordering which is essential for various lists in HTML. Our conceptual model uses list objects which naturally correspond to various lists in HTML.

Second, we have presented a set of generic rules to automatically convert HTML documents into this conceptual model. Discovering rules to structure the HTML text has already been addressed in the literature such as [5, 7, 8, 11]. However, the rules presented in this paper are more general and systematic. Because we use list objects to represent tuples, lists, tables, frames, forms, menus, etc. and ignore many features that are used to enhance the visual aspects of the web documents, it may be impossible to convert some objects in our data model back into HTML format. In other words, we lose some information during conversion. However, we can capture the essence of HTML documents in a natural and intuitive way.

A wrapper to convert HTML documents into our conceptual model based on the conversion method presented here has been implemented in Java and can be downloaded from the web page at `http://www.cs.regina.ca/~mliu/webmodel`.

We would like to extend our work by developing powerful query languages based on our conceptual model presented here to query the structure and contents of HTML documents. Also, we would like to extend the conceptual model to properly represent the features omitted here so that we can convert between our conceptual model and HTML as suggested by one of the referees.

References

1. T. Bray, J. Paoli, and C.M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. *W3C Recommendation.* See http://www.w3c.org/TR/1999/REC-xml-19980210, February 1998.
2. P. Buneman, S. Davidson, G. Hilebrand, and D. Suciu. A Query Language and Optimization Techniques for Unstructured Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 505–516, 1996.
3. J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. *W3C Recommendation.* See http://www.w3c.org/TR/1999/REC-xpath-19991116, November 1999.
4. M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A Query Language for a Web-Site Management System. *SIGMOD Record*, pages 4–11, 1997.
5. M. Fernandez, D. Florescu, A. Levy, and D. Suciu. Reasoning About Web-Site Structure. In *Proceedings of AAAI'98 Workshop on AI and Information Integration*, 1998.
6. D. Florescu, A. Levy, and A. Mendelzon. Database Techniques for the World-Wide Web: A Survey. *SIGMOD Record*, 27(3):59–74, 1998.
7. J. Hammer, H. Garcia-Molina, J. Cho, A. Crespo, and R. Aranha. Extracting Semistructured Information from the Web. In *Proceedings of the Workshop on Management of Semistructured Data*, 1997.
8. C. A. Knoblock, S. Minton, J. L. Ambite, N. Ashish, P. J. Modi, I. Muslea, A. G. Philpot, and S. Tejada. Modeling Web Sources for Information Integration. In *Proceedings of the 15th National Conference on AI*, 1998.
9. M. Liu and T. W. Ling. A Data Model for Semistructured Data with Partial and Inconsistent Information. In *Proceedings of the International Conference on Advances in Database Technology (EDBT 2000)*, pages 317–331, Konstanz, Germany, March 27-31 2000. Springer-Verlag LNCS 1777.
10. M. Liu, T. W. Ling, and T. Guan. Integration of Semistructured Data with Partial and Inconsistent Information. In *Proceedings of the International Database Engineering and Application Symposium (IDEAS '99)*, pages 44–52, Montreal, Canada, August 2-4 1999. IEEE-CS Press.
11. I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical Wrapper Induction for Semistructured Information Sources. *To appear in Journal of Autonomous Agents and Multi-Agent Systems.*
12. Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange across Heterogeneous Information. In *Proceedings of the International Conference on Data Engineering*, pages 251–260. IEEE Computer Society, 1995.
13. D. Raggett, A. L. Hors, and I. Jacobs. HTML 4.01 Specification. *W3C Recommendation.* See http://www.w3c.org/TR/html401, December 1999.
14. L. Wood, A. L. Hors, et al. Document Object Model (DOM) Level 2 Specification. *W3C Recommendation.* See http://www.w3c.org/TR/2000/CR-DOM-Level-2-20000307, March 2000.