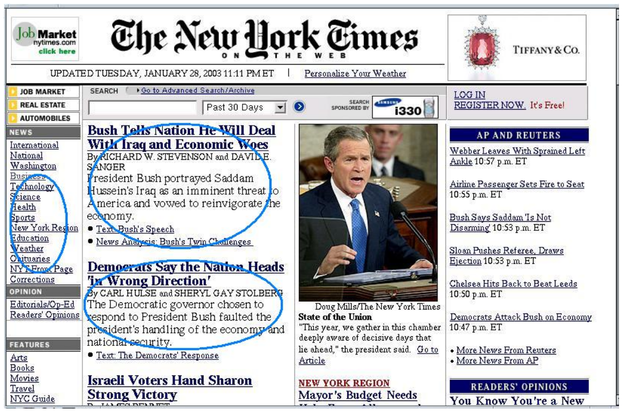
On the Power of Semantic Partitioning of Web Documents

Guizhen Yang   Saikat Mukherjee
Wenfang Tan   I.V. Ramakrishnan
Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794-4400
{guizyang, saikat, wtan, ram}@cs.sunysb.edu

Hasan Davulcu
Department of Computer Science
and Engineering
Arizona State University
Tempe, AZ 85287-5406
hdavulcu@asu.edu

1 Introduction

A growing number of Web sites are maintained by content management software and thus a large number of Web pages are machine-generated via templates. Normally in such Web pages there is *implicitly* a fixed "schema" and what changes is the content. Informally a schema for a Web page represents concepts and relationships among them in a hierarchical fashion. For example, Figure 1 is a screen shot of the New York Times front page (see http://www.nytimes.com). Observe that this page includes: (i) a taxonomy of items such as "NEWS" (consisting of hyperlinks labeled with "International", "National", ...), "OPINION" (consisting of hyperlinks "Editorial/Op-Ed", ...), etc.; (ii) several headlines of news articles where each article begins with a hyperlink labeled with the news headline (*e.g.*, "Bush tells Nation ...") followed by the author of the article (*e.g.*, "By Richard W. Stevenson ..."), followed by a time-stamp and a text summary of the article (*e.g.*, "President Bush portrayed ..."). The schema for this fragment of the New York Times front page therefore includes the taxonomy (which does not change) and the template for the news article. We should point out that the schema will also include several additional elements pertaining to other content appearing in the page.


Job Market nytimes.com click here

The New York Times ON THE WEB

UPDATED TUESDAY, JANUARY 28, 2003 11:11 PM ET | Personalize Your Weather

TIFFANY & Co.

JOB MARKET
REAL ESTATE
AUTOMOBILES

SEARCH ▸ Go to Advanced Search/Archive
Past 30 Days
SEARCH SPONSORED BY SAMSUNG i330

LOG IN
REGISTER NOW. It's Free!

NEWS
International
National
Washington
Business
Technology
Science
Health
Sports
New York Region
Education
Weather
Obituaries
NYT Front Page
Corrections

OPINION
Editorials/Op-Ed
Readers' Opinions

FEATURES
Arts
Books
Movies
Travel
NYC Guide

Bush Tells Nation He Will Deal With Iraq and Economic Woes
By RICHARD W. STEVENSON and DAVID E. SANGER
President Bush portrayed Saddam Hussein's Iraq as an imminent threat to America and vowed to reinvigorate the economy.
- Text: Bush's Speech
- News Analysis: Bush's Twin Challenges

Democrats Say the Nation Heads 'in Wrong Direction'
By CARL HULSE and SHERYL GAY STOLBERG
The Democratic governor chosen to respond to President Bush faulted the president's handling of the economy and national security.
- Text: The Democrats' Response

Israeli Voters Hand Sharon Strong Victory

Doug Mills/The New York Times
State of the Union
"This year, we gather in this chamber deeply aware of decisive days that lie ahead," the president said. Go to Article

NEW YORK REGION
Mayor's Budget Needs

AP AND REUTERS
Webber Leaves With Sprained Left Ankle 10:57 p.m. ET
Airline Passenger Sets Fire to Seat 10:55 p.m. ET
Bush Says Saddam 'Is Not Disarming' 10:53 p.m. ET
Sloan Pushes Referee, Draws Ejection 10:53 p.m. ET
Chelsea Hits Back to Beat Leeds 10:50 p.m. ET
Democrats Attack Bush on Economy 10:47 p.m. ET
• More News From Reuters
• More News From AP

READERS' OPINIONS
You Know You're a New

Figure 1: New York Times Front Page

The important question then is: *Can the implicit schema in template-driven HTML documents be made explicit?* We formulate the problem of schema discovery from HTML documents as one of discovering semantic structures in Web documents and partitioning them accordingly. Our objective is to take a HTML document generated by a template and automatically partition it into semantically meaningful clusters via structural and semantic analysis. Each partition will consist of items related to a semantic concept. For example, Figure 2 is such a tree corresponding to the New York Times front page in Figure 1. Observe in this figure that the headline news items are all grouped under the "Headline News" category.

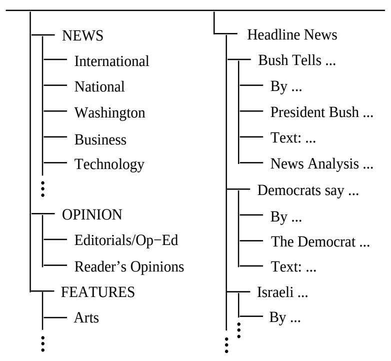


Figure 2: Screen Shot of the Semantic Partition Tree for New York Times Front Page

Semantic partitioning has several important and powerful implications in practice. First, it eases the task of formulating queries to retrieve data from Web documents. In the New York Times example, one can pose a query to retrieve all the links under the "NEWS" item in the taxonomy. Knowledge of the schema made explicit via semantic partitioning is the key to transforming legacy HTML documents into more semantics-oriented document formats such as XML [XML, 2003] and DAML [DAML, 2000]. Yet another application is audio-browsable Web content. By putting a dialog interface to the content of a Web page which is reorganized based on the knowledge of its schema, a user can easily browse its content using audio. More generally a Web site itself can be navigated using voice commands. Audio browsable Web content can significantly expand the reach of the Web to visually challenged individuals. Finally semantic partitioning can enable the creation of self-repairing wrappers, the technology that

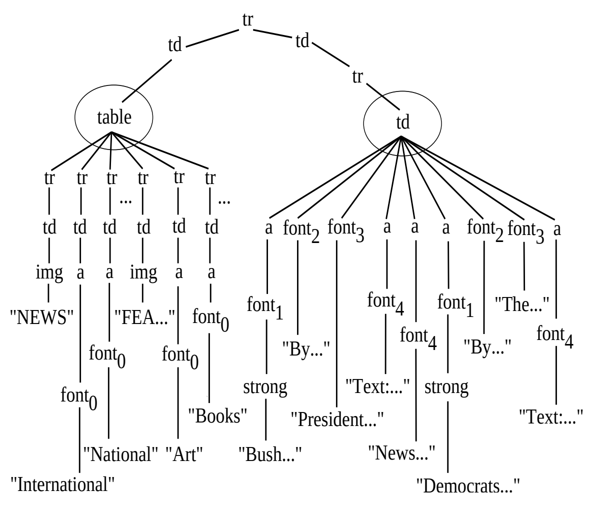


Figure 3: DOM Tree Fragment of New York Times Front Page

provides a database-like interface to Web documents. The rest of this paper describes our approach to semantic partitioning and its applications.

2 Semantic Partitioning

Herein we describe the ideas underlying our approach to semantic partitioning that is carried out through a combination of structural and semantic analysis.

2.1 Structural Analysis

Structural analysis is based on the observation that in well-organized HTML documents semantically related items, as discerned in their rendered views, exhibit spatial locality. For example, observe that in Figure 1 all the taxonomic items such as “NEWS”, “OPINION”, “FEATURES”, etc., and the corresponding hyperlinks under them are all spatially clustered together. The same observation holds for all the headline news items, their associated authors, and the corresponding text summary. This organization is particularly clear in Web sites (especially those owned by portals, product vendors, service providers, etc.) that are maintained using content management software which automatically generates Web pages from templates.

In most Web documents spatial locality can be identified by looking for similarities in the path structures of the corresponding DOM trees. For example, the root-to-leaf path strings of all the links under the “NEWS” item in the taxonomy (“International”, “National”, etc.) in Figure 1, which consist of tag names and their associated attributes (see Figure 3), are all identical: $tr \cdot td \cdot table \cdot tr \cdot td \cdot a \cdot font_0$ (here $font$ tags with different subscripts denote $font$ tags with different attribute values such as $size$, $color$, etc.). The content categories in the taxonomy such as “NEWS”, “OPINION”, and “FEATURES” also have similar path structures.

If related items in a Web page exhibit spatial locality, we can find structurally similar Web page elements by looking for patterns in the occurrences of path strings of the leaf nodes in a DOM (sub)tree. For instance, let us denote the path string $tr \cdot td \cdot table \cdot tr \cdot td \cdot img$ from Figure 3 using T_1, and $tr \cdot td \cdot table \cdot tr \cdot td \cdot a \cdot font_0$ using T_2. The subtree rooted at $table$ (shown circled) in Figure 3 has the following string: $T_1T_2T_2 \ldots T_1T_2T_2 \ldots$. The problem of spatial locality discovery can then be reduced to the problem of *sequential pattern* analysis. For instance, $T_1T_2^*$ (here $*$ denotes Kleene closure) is a sequential pattern that can be observed from the string $T_1T_2T_2 \ldots T_1T_2T_2 \ldots$, in which T_1 represents a taxonomic item such as “NEWS”, “OPINION”, etc. and T_2^* represents the collection of hyperlinks, such as “International”, “National”, etc. Similarly, we can obtain the string, $T_3T_4T_5T_6T_6T_3T_4T_5T_6$, and the sequential pattern, $T_3T_4T_5T_6^*$, from the subtree rooted at td (shown circled) in Figure 3. (Here T_3, T_4, T_5, T_6 denote the the path strings containing $font_1, font_2, font_3, font_4$, respectively.)

Sequential pattern discovery can be performed recursively bottom-up starting from the leaves of the DOM tree of a Web page. Near-leaf patterns correspond to small partitions. Small partitions can be aggregated into bigger partitions for Web pages with rich content such as Yahoo. The algorithmic details of structural analysis is provided below.

2.2 Algorithms

To transform the DOM tree of a HTML document into a tree-like semantic structure, we simply invoke the top-level algorithm $PartitionTree$ on the root of the given DOM tree. This algorithm first traverses the DOM tree top-down and then restructures it bottom-up.

```
Algorithm PartitionTree($n$)
input
    $n$ : a node in a DOM tree
begin
1.  if $n$ is a leaf node then
2.      $n.type$ = the sequence of HTML tags from the root to $n$
3.  else if $n$ has only one child node $c$ then
4.      PartitionTree($c$)
5.      Replace $n$ with $c$ and remove $n$ from the DOM tree.
6.  else
7.      for each child node $x$ of $n$ do PartitionTree($x$) endfor
8.      FindPartition($n$)
9.  endif
end
```

In our data structure, each node of the tree has an additional attribute, $type$, which stores the type assigned to this node. This attribute basically encodes the summary of structural recurrence discovered for the subtree rooted at this node. We will use the notation $n.type$ to represent the $type$ attribute of a node n.

In Line 2 of the algorithm $PartitionTree$, all the leaf nodes are typed. Internal nodes with only one child are handled in Lines 4–5. In such a case, the type of this only child node is computed and then simply propagated up the tree. However, for an internal node with multiple children, we first invoke $PartitionTree$ on all of its children to collect their type information (Line 6). Then the algorithm $FindPartition$ is invoked upon this node to perform a pattern discovery on its children nodes (Line 7).

```
**Algorithm** FindPartition($n$)
**input**
    $n$ : an internal node in a DOM tree
**begin**
1.  $S$ = the sequence of all the child nodes of $n$
2.  **for** each node $c$ in $S$ **do**
3.      **if** $c.flatten$ = $true$ **then**
4.          Replace $c$ with the sequence of all the child nodes of $c$.
5.      **endif**
6.  **endfor**
7.  $\tau = \varepsilon$
8.  **do**
9.      Collapse adjacent nodes in $S$ which share the same type.
10.     $\alpha$ = MaximalRepeatingSubstring(TypeStr($S$))
11.     **if** $\alpha \neq \varepsilon$ **then** $\tau = \alpha$ **endif**
12.     **if** $|\alpha| > 1$ **then**
13.         **for** each substring $\rho$ in $S$ such that TypeStr($\rho$) = $\alpha$ **do**
14.             Replace $\rho$ with NewNode($\rho$,$seq(\alpha)$).
15.         **endfor**
16.     **endif**
17. **while** $|\alpha| > 1$
18. **if** $\tau = \varepsilon$ **then**
19.     $n.flatten = true$
20. **else**
21.     Partition $S$ into $\beta_0\gamma\beta_1 \ldots \gamma\beta_m$, where TypeStr($\gamma$) = $\tau$.
22.     **for** each $\gamma\beta_i$ **do**
23.         Replace $\gamma\beta_i$ with NewNode($\gamma\beta_i$, NewType($\tau$)).
24.     **endfor**
25.     $n.type$ = NewType($\tau$)
26. **endif**
27. Make the nodes in $S$ the new children of $n$.
**end**
```

The algorithm $FindPartition$ takes an internal node, n, as input. Its main function is to discover structurally similar items among all the children of n and restructure the subtree rooted at n accordingly. Because our algorithm climbs up a DOM tree from leaf nodes to the root, structural similarity may not be observed until it reaches a node high enough. Therefore, we associate a boolean attribute, $flatten$, with each node to signal whether a structural similarity pattern has been discovered at this node. The value of this attribute is initialized to $false$ for each node. However, if a pattern (or type) is not found at a node, then its $flatten$ attribute is set to $true$ (Line 19).

In Lines 1–6, all the child nodes of n are collected into a sequence, which will be partitioned into semantically related items later if they share structural similarity. But if we encounter a node, c, whose $flatten$ attribute has the value $true$ (which means a pattern is not found at this node), then we move all the child nodes of c into this sequence for further processing.

Note that when the algorithm $FindPartition$ is invoked on a node, all of its descendant nodes are already typed. Intuitively, since the type of a node summarizes the structure of the subtree rooted at that node, analysis of the sequence of sibling types is essential for structural similarity pattern discovery, which is done in two stages by our algorithm.

In the first stage, consecutive nodes having equivalent types are collapsed into a single node (Line 9). The intuition behind this is that they all relate to the same item. Next, in Line 10, an attempt is made to find a maximal repeating substring of the string corresponding to the type sequence of S (returned by $TypeStr(S)$).

If such a substring does not exist (hence no structural similarity), then the loop in Lines 8–17 is exited and the $flatten$ attribute of the current node is set to $true$ (Line 19). However, if a maximal repeating substring, α, is found and α contains at least two elements ($|\alpha| > 1$), then the sequence of consecutive nodes whose type sequence matches α is merged into a new node created by the procedure $NewNode$ (Lines 12–16). The first argument of $NewNode$ contains the sequence of nodes to be merged while the second argument indicates the type of this new node. The above collapsing-pattern-discovering-merging process is repeated until it cannot be performed any more.

In the main part of the second stage (Lines 21–25), the last pattern discovered during the first stage is used to partition the remaining sequence of nodes further. This is a simple heuristic that we apply to handle variations in document structures (*e.g.*, missing data items). Note that if τ contains only one type, then $NewType(\tau)$ returns τ directly; otherwise, it returns the compound type $seq(\tau)$.

Now we illustrate the working steps of the algorithm $FindPartition$ using an example. For simplicity, we will just show how it manipulates a sequence of types and omit other details. Suppose the type sequence of S is $T_1T_2T_3T_2T_3T_4T_1T_2T_3T_5$ immediately before the algorithm executes the loop starting at Line 8. T_2T_3 is a maximal repeating substring. Let us use a new type T_6 to denote $seq(T_2T_3)$. Then after the first iteration of the loop, the type sequence becomes $T_1T_6T_6T_4T_1T_6T_5$. The first two occurrences of T_6 can be collapsed into one, resulting in $T_1T_6T_4T_1T_6T_5$, in which T_1T_6 is a maximal repeating substring. Again, we use a new type T_7 to represent $seq(T_1T6)$. So after the second iteration the type sequence becomes $T_7T_4T_7T_5$ and the loop terminates. It is not hard to see that the first T_7 and the following T_4 will be put into one partition and the rest into another partition. T_7 is the type assigned to the current node.

The algorithms $PartitionTree$ and $FindPartition$ are illustrated using the DOM tree fragment shown in Figure 3. Let us consider the subtree rooted at the node td (shown circled) spanning the leaf nodes from "Bush..." to "Text...". The type of the "Bush..." leaf node, denoted by T_1, is $tr \cdot td \cdot tr \cdot td \cdot a \cdot font_1 \cdot strong$. Observe that the leaf node "Democrats..." has the same type T_1. So we can assign the types $T_1, T_2, T_3, T_4, T_4, T_1, T_2, T_3, T_4$ to the leaf nodes from "Bush..." to "Text...", respectively. Observe that all these leaf nodes are the only child of their parent node. As a result, their ancestor nodes are deleted (Lines 4–5 of $PartitionTree$) until they are propagated up the subtree and become siblings under the nearest td node.

Now the algorithm $FindPartition$ is invoked on the sequence of types $T_1T_2T_3T_4T_4T_1T_2T_3T_4$. First, the two consecutive occurrences of T_4 are collapsed together (Line 9 of $FindPartition$). The resulting type sequence is $T_1T_2T_3T_4T_1T_2T_3T_4$, in which $T_1T_2T_3T_4$ is a maximal repeating substring. So the original sequence of nodes is partitioned into two parts, each corresponding to the pattern $T_1T_2T_3T_4$. The type assigned to the td node (nearest to the "Bush..." leaf node) is $seq(T_1T_2T_3T_4)$.

2.3 Semantic Analysis

Partitioning. The content and structure of template-generated Web pages will still vary due to, *e.g.*, updates to the backend databases that are used to populate the templates and slight variations in presentation styles. Therefore, a purely structural analysis will not always generate “correct” partitions. However, structural analysis can be combined with *semantic analysis* to produce high quality partitions. For example, to determine whether two segments of text are related, we use WordNet[1] to identify semantically-related nouns they may share. The notion of semantic relatedness is derived from the different types of relationships found in WordNet, like synonyms, hypernyms, etc. These sorts of simple heuristics can work well on news and consumer product Web pages. The output of the semantic analysis module is provided to the structural analysis module to provide additional constraints during the partition process.

Labeling. The $PartitionTree$ algorithm transforms a HTML document into a tree of partitions. However, in order to derive a schema from the partition tree it is necessary to summarize the content of the partitions. A succinct summary of a partition is known as the *label* of the partition. We have used a combination of heuristics based on structural analysis and domain knowledge to label partitions.

Very often it is the case that the labels of leaf partitions (in the partition tree) are usually provided by Web site designers in the page itself. In such circumstances, the node containing the label, having type T_1, is followed by multiple sibling nodes each having the same type T_2 which is different from T_1. This is illustrated in the circled text in the left-hand side menu in Figure 1, where the node “NEWS”, with type T_1, is followed by the sibling nodes “International”, “National”, “Washington”, etc. each having the type T_2. In such a case, the content of the first node is made the label of the entire partition. This results in the labeled partition “NEWS” as shown in Figure 2.

Even though the above heuristic performs well in practice, it is not a general technique to label arbitrary leaf partitions. Moreover, the heuristic cannot be applied to label internal partitions in the partition tree. In general, such labels will be very hard to obtain without leveraging some domain knowledge, commonly referred to as *ontologies*. Informally an ontology describes concepts, along with their features or attributes, in a domain of interest. Typically, the ontology captures a hierarchical parent-child relationship between the concepts. Given such a representation of a domain knowledge, the labeling problem is essentially reduced to classifying a partition to an appropriate concept in the ontology. An internal partition is classified to the least common ancestor concept of the concepts to which its children partitions have been classified. Thus, labeling is a 3-step process whereby: (i) the domain ontology has to be engineered; (ii) a classifier for every concept has to be generated; and (iii) the “best” concept for every partition is discovered using the classifiers and the taxonomy of concepts.

Recently there has been a lot of work on engineering domain ontologies [DAML, 2000]. In principle, our labeling technique can be used with any domain ontology which has been enriched with concept classifiers. For our work, we have used the human edited taxonomy of the Open Directory Project[2] as our reference ontology. The use of the Open Directory Project ontology facilitates automatic generation of concept classifiers, as described below.

[1]http://www.cogsci.princeton.edu/˜wn/

[2]http://www.dmoz.org

We use a combination of rules and statistical analysis as the concept classifier. The rules represent structural features of the particular concept. For example, in a News ontology the concept “Headline News” can be structurally characterized by a set of items where each item is characterized by a hyperlink, keywords for recognizing news sources (such as AP, Reuters, By, From, etc.), patterns for recognizing date and time when the news item was filed, and features associated with news summaries such as constraints on the text length. Using this rule, we were able to label the partitions corresponding to the circled texts in the central column of Figure 1 as “Headline News”. The labeled partition “Headline News” is shown in Figure 2.

While rules are expressive, statistical features are easier to generate for a concept. As such, we train a statistical classifier, in particular a Naive Bayes classifier [Mitchell, 1997], for every concept in the ontology. Associated with every concept in the Open Directory Project ontology is a set of Web pages which have been manually classified as pertaining to that concept. The bag of words contained in these pages can be used to train a Naive Bayes classifier for that concept. Note that training such a concept classifier is completely automatic due to the existing corpus of associated pages for that concept. Moreover, the high precision of human editing in the Open Directory Project results in fairly accurate concept classifiers.

3 Applications of Semantic Partitioning

Several important applications are enabled by semantic partitioning. Herein we briefly sketch a few of them.

3.1 Semantic Annotations of Web Documents

The objective of the Semantic Web [The Semantic Web, 2003] is to define and share machine processable data which will enable a variety of automated tasks ranging from information search to data integration to Web services. The techniques for semantic partitioning proposed in this paper can serve as a useful technology for transforming unstructured HTML documents into structured data that is amenable to machine processing such as querying and reasoning. Specifically the transformed document will be annotated with semantic information derived by the semantic partitioning algorithm. Figure 1 is an example of such a transformed document. These annotations can be further aligned with the standard vocabulary of a specific ontology domain and presented as RDF documents [RDF, 2003] using machine learning techniques [Doan *et al.*, 2003].

3.2 Self-Repairing Wrappers

Wrappers are programs that provide database-like interfaces to Web sources [Adelberg, 1998; Ashish and Knoblock, 1997; Hammer *et al.*, 1997; Perkowitz *et al.*, 1997; Atzeni and Mecca, 1997]. Techniques for programmatic, semi- and fully- automated wrapper construction has been extensively researched and wrapper-based tools have been developed [Crescenzi *et al.*, 2001; Sahuguet and Azavant, 1999;

Baumgartner *et al.*, 2001; Liu *et al.*, 2000; Kushmerick *et al.*, 1997; Chidlovskii, 2001; Muslea *et al.*, 1999; Ashish and Knoblock, 1997; Cohen *et al.*, 2002; Hsu and Dung, 1998]. Fully and semi-automated approaches for constructing wrappers are typically based on the idea of *learning from labeled examples*. To build a wrapper examples of data of interest are labeled. From these examples, the system learns extraction expressions (such as regular expressions) using syntactic cues, such as HTML tags, and keyword strings in the pages. Extracting data of interest from Web sites amounts to applying these expressions at the appropriate locations in Web pages.

A critical problem confronting wrapper technology is changes in Web pages. Frequently, Web sites change to accommodate new presentation formats, services, and content offerings. Such changes can cause "brittleness" in data extractors thereby causing wrappers to fail. For example, if the headline news in New York Times's front page becomes embedded within a list structure instead of a table as shown in Figure 3 then the extraction expression learned with the table structure will fail to locate the news items. This raises the important question of whether it is possible to build *resilient* wrappers that can *automatically* adapt to structural changes in Web pages.

In a previous work we had attacked this problem from a syntactic viewpoint [Davulcu *et al.*, 2000]. A truly robust solution, however, will require semantic knowledge of the content in a Web page. The semantic partitioning ideas described in this paper can form the foundation for such a solution. To understand the idea at a high level, let us examine how a self-repairing wrapper based on semantic partitioning, will handle the change in New York Times's front page that makes each news item occur as a list instead of a table element. The old extraction expression matching the circled td (in Figure 3) node will fail to correctly identify the subtree containing all the news items, so the system will re-partition the Web page. If it finds a partition in the re-partitioned page labeled "Headline News" then it will once again generate the extraction expressions appropriate for the elements in this partition and update the information that it maintains for extraction runs. Thus semantic partitioning can contribute to making wrappers self-repairing.

For this self-repair process to work the wrapper will have to include the labels of the partitions that contain the data items marked for extraction. Recall that our labeling procedure can be enriched by using an ontology. In such a case we will also record in the wrapper the node in the ontology to which the partition is mapped. Self-repairing wrappers is a topic that has received little attention in the literature. The approach based on semantic partitioning seems quite promising.

3.3 Creating Audio-Browsable Web Content

The primary mode of interaction with the Web is via browsers that are designed for visual modes of interaction. This denies access to an entire community of users who suffer from visual disabilities. Semantic partitioning can lead to the creation of a new generation of technologies that will empower visually impaired individuals to access and navigate Web sites using non-visual modalities, such as voice commands and audio output. A unique aspect of such technologies will be that it will enable audio-based *exploratory browsing* of Web content in a structured and efficient way, and more importantly the audio-browsable content will not depend on special content providers.

An envisioned audio-browser system based on semantic partitioning will work as follows. First the user provides the URL to the system, say http://www.nytimes.com, via audio or keyboard. After retrieving the front page of New York Times in Figure 1, the system performs structural and semantic analysis on this Web page to determine homogeneous, semantically related segments in it. The result is the semantic partition tree shown in Figure 2.

```
<form id="home">
 <field name="choice">
  <prompt>
   Alice, please choose one of these:
          Headline News.
          News.
          Opinion.
          Exit.
  </prompt>
  <grammar> Headline News|News|Opinion|..|Exit </grammar>
  <filled>
   <if cond="choice=='Headline News'">
    <goto next="#headline_news"/>
   <elseif cond="choice=='News'"/>
    <goto next="#news"/>
   <elseif cond="choice=='Opinion'"/>
    <goto next="#opinion"/>
   <elseif cond="choice=='Exit'"/><exit/>
   </if>
  </filled>
 </field>
</form>
```

Figure 4: Fragment of a VoiceXML Dialog for Browsing the Partition Tree in Figure 2

Next the audio-browser system automatically generates a speech dialog interface to the partitioned segments. Such a dialog can be created using the emerging standard of VoiceXML [VoiceXML, 2003]. Part of a sample VoiceXML dialog for the partition tree in Figure 2 is shown in Figure 4. The `<prompt>` tag indicates that the VoiceXML browser should play back all of the enclosed text in its scope as synthesized speech. A `<field>` tag is used to indicate an input field. It signals the VoiceXML browser to listen for user input and interpret it according to a grammar specified in the script.

Once this dialog interface is created, the user navigates the partitioned segments on demand. Following the above VoiceXML script, the system reads out the labels of the top-level partitions ('NEWS", "OPINION", ..., "Headline News", ..., etc.), pausing briefly after each item to let the user pick a partition by saying the label. If the user says "News", the system reads out the label and type of each item in the "NEWS" partition (in this case all the items are navigation links so type information may be summarized at the beginning). Once again, the user can pick any item by saying the label. If the user says "Business", the system follows this link to the business page, semantically partitions the resulting page, and sets it up for exploratory browsing with audio again. On the other hand the user may wish to listen to the headline news. In this case the user will ask the system to explore the headline news items one by one. At any point the user can also say any one of a set of browsing commands, such as “Back”, “Start over” (from the beginning of the segment), “Repeat” (last item) or “Stop”.

The above dialog lets the user explore the page depth-first. Alternatively, a dialog can be generated for breadth-first browsing. The user would tell the system which top-level items to keep (*e.g.*, Headline News and Opinion) and which to drop. The system would then let the user to explore each selected section separately.

4 Related Work

There is a large body of work on discovering schema information from either XML documents [Goldman and Widom, 1997; Nestorov *et al.*, 1998; Garofalakis *et al.*, 2000] or XML queries [Papakonstantinou and Velikhov, 1999; Papakonstantinou and Vianu, 2000]. However, the problem of recovering semantic structures from HTML documents has only been explored recently.

In [Yang and Zhang, 2001] Yang and Zhang propose to build semantic structures from HTML documents by detecting patterns and separation boundaries. They view a HTML document as a sequence of HTML tags and texts. Their pattern discovery technique relies on a hand-coded similarity function. Moreover, they do not consider the problem of labeling a partition.

The work of Chung et al. [Chung *et al.*, 2002] takes advantage of tree structures of HTML documents to transform them into XML counterparts. Their approach makes use of domain knowledge that is hand-coded into a concept classifier to identify elementary concepts and group them into bigger, structural concepts. However, their techniques do not fully explore layout regularity which is commonly observed in template-driven HTML documents.

Recently the proposals of [Crescenzi *et al.*, 2001; Arasu and Garcia-Molina, 2003] address the issue of schema discovery from a collection of Web pages. Our problem departs slightly from theirs because we are concerned with schema discovery from individual pages.

Finally, it is worth contrasting the problem of schema discovery for template-driven HTML documents to the important, well-studied problem of wrapper-based data extraction [Hammer *et al.*, 1997; Cohen *et al.*, 2002; Liu *et al.*, 2000]. We should point out that wrappers generate domain-specific queriable interface to HTML documents which is orthogonal to the schema discovery problem.

5 Conclusion

In this paper we proposed techniques based on structural and semantic analysis to partition Web documents into semantic structures. The idea of semantic partitioning has important implication to other data management problems. Currently we are exploring adapting our techniques to such problems as semantic annotation of Web documents, building self-repairing wrappers, and creating audio-browsable Web content.

References

[Adelberg, 1998] B. Adelberg. Nodose: A tool for semi-automatically extracting structured and semi-structured data from text documents. In *ACM SIGMOD*, 1998.

[Arasu and Garcia-Molina, 2003] Arvind Arasu and Hector Garcia-Molina. Extracting structured data from web pages. In *ACM SIGMOD*, 2003.

[Ashish and Knoblock, 1997] N. Ashish and C. Knoblock. Wrapper generation for semi-structured internet sources. *ACM SIGMOD Record*, 26(4), 1997.

[Atzeni and Mecca, 1997] P. Atzeni and G. Mecca. Cut & paste. In *ACM PODS*, 1997.

[Baumgartner *et al.*, 2001] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual web information extraction with lixto. In *International Conference on Very Large Data Bases (VLDB)*, 2001.

[Chidlovskii, 2001] Boris Chidlovskii. Wrapping web information providers by transducer induction. In *European Conference on Machine Learning*, 2001.

[Chung *et al.*, 2002] Christina Yip Chung, Michael Gertz, and Neel Sundaresan. Reverse engineering for web data: From visual to semantic structures. In *ICDE*, 2002.

[Cohen *et al.*, 2002] William Cohen, Matthew Hurst, and Lee Jensen. A flexible learning system for wrapping tables and lists in html documents. In *International World Wide Web Conference*, 2002.

[Crescenzi *et al.*, 2001] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *International Conference on Very Large Data Bases (VLDB)*, 2001.

[DAML, 2000] DARPA agent markup language, http://www.daml/org. 2000.

[Davulcu *et al.*, 2000] H. Davulcu, G. Yang, M. Kifer, and I.V. Ramakrishnan. Computational aspects of resilient data extraction from semistructured sources. In *ACM PODS*, May 2000.

[Doan *et al.*, 2003] AnHai Doan, Pedro Domingos, and Alon Y. Halevy. Learning to match the schemas of data sources: A multistrategy approach. 50(3):279–301, 2003.

[Garofalakis *et al.*, 2000] Minos Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: A system for extracting document type descriptors from xml documents. In *ACM SIGMOD*, 2000.

[Goldman and Widom, 1997] Roy Goldman and Jennifer Widom. DataGuides: Enabling query formulation and optimization in semistructured databases. In *VLDB*, 1997.

[Hammer *et al.*, 1997] J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni, M. M. Breunig, and V. Vassalos. Template-based wrappers in the TSIMMIS system. In *ACM SIGMOD*, 1997.

[Hsu and Dung, 1998] Chun-Nan Hsu and Ming-Tzung Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems*, 23(8):521–538, 1998.

[Kushmerick *et al.*, 1997] N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In *Intl. Joint Conf. on Artificial Intelligence*, volume 1, 1997.

[Liu *et al.*, 2000] Ling Liu, Calton Pu, and Wei Han. XWRAP: An XML-enabled wrapper construction system for web information sources. In *ICDE*, 2000.

[Mitchell, 1997] Tom M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[Muslea *et al.*, 1999] Ion Muslea, Steve Minton, and Craig Knoblock. A hierarchical approach to wrapper induction. In *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 190–197, 1999.

[Nestorov *et al.*, 1998] Svetlozar Nestorov, Serge Abiteboul, and Rajeev Motwani. Extracting schema from semistructured data. In *ACM SIGMOD*, 1998.

[Papakonstantinou and Velikhov, 1999] Yannis Papakonstantinou and Pavel Velikhov. Enhancing semistructured data mediators with document type definitions. In *ICDE*, 1999.

[Papakonstantinou and Vianu, 2000] Yannis Papakonstantinou and Victor Vianu. DTD inference for views of xml data. In *ACM PODS*, 2000.

[Perkowitz *et al.*, 1997] M. Perkowitz, R. B. Doorenbos, O. Etzioni, and D. S. Weld. Learning to understand information on the internet: An example-based approach. *Journal of Intelligent Information Systems*, 8(2), 1997.

[RDF, 2003] Resource description framework, http://www.w3/org/RDF. 2003.

[Sahuguet and Azavant, 1999] A. Sahuguet and F. Azavant. Web Ecology: Recycling HTML pages as XML documents using W4F. In *ACM SIGMOD Workshop on the Web and Databases (WebDB)*, 1999.

[The Semantic Web, 2003] W3C semantic web, http://www.w3.org/2001/sw/. 2003.

[VoiceXML, 2003] Voicexml forum, http://www.voicexml.org/. 2003.

[XML, 2003] Extensible markup language, http://www.w3.org/XML/. 2003.

[Yang and Zhang, 2001] Yudong Yang and Hongjiang Zhang. HTML page analysis based on visual cues. In *ICDAR*, 2001.