A DTD-Syntax-Tree Based XML file Modularization Browsing Technique

Zhu Zhengyu[1], Changzhi Li, Yuan Gao

Computer College of Chongqing University, Chongqing (400044), China

ABSTRACT

First, by using the current mature HTML information retrieval techniques, an XML information retrieval system framework will be given in this paper. Then, a DTD-tree based XML file modularization browsing technique will be introduced to browse the retrieval result (a list of XML URLs). Compared with the current XML retrieval systems[2,6-9], our new system has the following advantages: 1) It can retrieve XML information among all the XML files on Internet. 2) It can let the user browse XML files more easily and faster by modularizing these files. 3) It can effectively reduce the load on data transmission. So it is applicable for cases of the users of mobile devices or users of dial-in terminal connections.

Keywords

DTD syntax tree, XML, Basic Content Module, Information Retrieval System.

1. INTRODUCTION

XML [1] is a simplified subset of the Standard Generalized Markup Language (SGML), a text-based data format for structured documents. XML is a tag language that looks similar in style to HTML [3]. However, it supports a much richer set of features such as user-definable tags. XML is now a recommendation of W3C.

XML is perceived as the next wave of the Internet technology with the potential of replacing HTML for several reasons. With the increase in popularity in the use of XML pages, it is becoming an important thing that how XML information can be retrieved effectively from XML files.

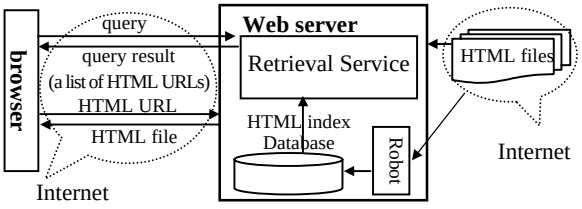
The current information retrieval systems on XML files (XML-IRS) can be divided into two types. One [7,8] is based on RDBMS. XML data are transformed and saved in the RDB. The user queries XML information by SQL. The other [2,6,9] is based on some special XML query language. It works directly on XML files. The user queries XML information by a query language something like SQL. There exist some disadvantages for these systems. The RDBMS-based systems can only retrieve information on those XML files managed by the RDBMS. The query language-based systems require users must specify definitely the XML file's URL (XML URL, for shortening), know its internal structure, and have some knowledge about the language.

First, In this paper we will discuss how to use the current HTML information retrieval techniques to build a basic XML-IRS, which can retrieve XML information among all the XML files on Internet. Then, we will introduce a DTD-tree based XML modularization browsing technique to let users browse the retrieval result more easily and faster.

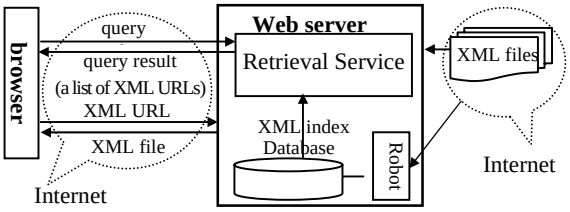
2. A basic XML-IRS framework

Generally, a traditional information retrieval system on HTML files (HTML-IRS) has the framework as is shown in Figuer1 and its work process is as follows. 1) With some strategy, the Robot searches and gathers the index information about each HTML page on Internet and builds the HTML index database. 2) The user enters a query condition at the browser. 3) The Web server searches data in the HTML index database and finds out the query result (a list of URLs of all the HTML pages) satisfying the given query condition on Internet. 4) The query result will be transmitted to the browser and shown to the user. 5) The user will read and click each URL in the query result one by one. 6) When a URL is clicked, the actual HTML page will be returned to the browser.

Both XML files and HTML files are plain text files and both are made up of tags and contents. Any XML file can be directly read and understood by the user. An XML file can be easily wrapped as a new HTML file which enables the user to read the original XML file in the browser. So by using the technique on the HTML IRS, we can easily build a basic XML IRS framework as is shown in Figure2. Its work process is very similar to that of the HTML IRS. To browse the query result (a list of XML URLs) one by one, the user can directly read each XML file in the browser, or get the needed XML information from each XML file by some tool such as XQuery[2] or XQL[9] system.



[Figure1] The traditional HTML IRS framework



[Figure2] The basic XML IRS framework

3. The DTD-based XML modularization technique

Commonly, an XML-IRS ought to complete two main tasks. The first task is how to find out all the XML URLs (the query result) among all the XML files on Internet for a given query condition. The second task is how to fast browse and finally retrieve out the needed XML information from the query result.

Though our basic XML-IRS has the important capability that can do the first Task well, it is difficult and time-consuming to complete the second task. If the user directly reads all the XML files of the URLs in the query result, it is time-consuming because he may have to deal with a great number of URLs or large files. If the user uses some tool such as XQuery or XQL system, he must command the knowledge of its query language and know the internal structures of XML files in the query result.

To do the second task better, we will introduce a DTD-based XML modularization technique into our system.

3.1 The concept about BCMs

Based on the technique of making the contents of an HTML file blocked, an HTML-IRS has been presented in our former paper [4] and It can let users browse a set of HTML file's URLs easily and fast. Below we will introduce this idea into our XML-IRS. Because XML files are different from HTML files, the technique in our XML-IRS will also be different from that in HTML-IRS. First, we introduce the concept about BCMs.

Though an XML file might contain a lot of contents, generally, a different user might concern about its different part. So a better XML-IRS ought to be the one that can retrieve out only the needed part from the XML file and transmit it to the browser.

In our opinion, each XML file contains many Basic Content Modules (**BCM**s). Each BCM is a part of contents of an XML file and might be the needed XML information of users. Furthermore, some other XML information that might be needed by users can also be compounded from BCMs. We call them the Compound BCMs (**CBCM**s). Here is an example to illustrate the concept.

[Example 1] For bib.xml in List1 with its DTD in List2, its three BCMs are Module1 "the title and the authors of the books", Module2 "the table of contents of the books" and Module3 "all the books whose title contains a given *string*". They are shown respectively in List3, List4 and List5 (when the string is "Web"). Here Module3 is a dynamic BCM. The *string* value can be given dynamically from the user. Otherwise, CBCM "the title, the authors and the table of contents of the books" can be composed of Module1 and 2.

```
<bib>
   <book year="2000">
     <title>Data on the Web</title>
     <author>Serge Abiteboul</author>
     <author>Peter Buneman</author>
     <section>
       <title>Introduction</title>
       <p>Text ... </p>
       <section>
         <title>Audience</title>
         <p>Text ... </p>
       </section>
       <section>
         <title>Web Data and the Two Cultures</title>
         <p>Text ... </p>
         <image source="csarch.gif"/>
         <p>Text ... </p>
       </section>
     </section>
  </book>
……
</bib>
```

[List1] bib.xml

```
<!ELEMENT bib (book*)>
<!ELEMENT book (title, author+, section+)>
<!ATTLIST book year CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT section (title, (p | image | section)* )>
<!ELEMENT p (#PCDATA)>
<!ELEMENT image EMPTY>
<!ATTLIST image
    source  CDATA   #REQUIRED >
```

[List2] The DTD file for bib.xml

```
<bib>
   <book>
     <title>Data on the Web</title>
     <author>Serge Abiteboul</author>
     <author>Peter Buneman</author>
   </book>
   ……
</bib>
```

[List3] The title and the authors of the books

```
<bib>
   <book>
     <title>Data on the Web</title>
     <section>
       <title>Introduction</title>
       <section>
         <title>Audience</title>
       </section>
       <section>
         <title>Web Data and the Two Cultures</title>
       </section>
     </section>
   </book>
  ……
</bib>
```

[List4] The table of contents of the books

```
<book year="2000">
 <title>Data on the Web</title>
 <author>Serge Abiteboul</author>
 <author>Peter Buneman</author>
 <section>
  <title>Introduction</title>
  <p>Text ... </p>
  <section>
    <title>Audience</title>
    <p>Text ... </p>
  </section>
  <section>
    <title>Web Data and the Two Cultures</title>
    <p>Text ... </p>
    <image source="csarch.gif"/>
    <p>Text ... </p>
  </section>
 </section>
</book>
```

[List5] All the books whose title contains a given *string* (when the string is "Web")

3.2 The DTD-based representation of BCMs

In this section, we will discuss how to describe BCMs. Different from the method in the HTML-IRS where the modules are described by adding module tags directly to HTML files [4], in our XML-IRS, we will describe BCMs with a DTD syntax tree.

First, from each DTD we can produce a DTD syntax tree as below. Each element in the DTD is a node in the tree. The order-relation and nesting-relation among elements in the DTD will correspond respectively to the brother-relation and father-children-relation in the tree. Then, for describing BCMs, a special Module Attribute (MA) will be used in the tree. Each MA-value denotes a unique BCM. To simplify the discussion, we only consider the elements in a DTD and so no attributes appear in the tree.

[Example 2] For the DTD in Example1, its DTD syntax tree with three BCMs is shown in Figure3. In a DTD syntax tree, we will use the following symbols to represent some special meanings:

( ) denotes MAs, all elements with MA=1 represent Module1, likewise MA=2 represents Module2 and MA=3 with a variable 'title%' represents Module3, a dynamic BCM;

⋯▸ denotes the nested element that is also its sub-element;

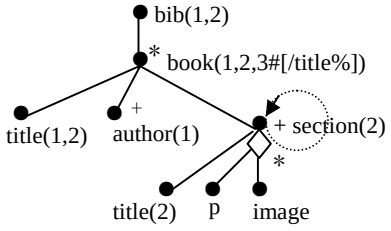
◇ denotes one of the elements;

\+ denotes one or more elements;

* denotes zero or more elements;

n# denotes that all the nodes of the sub-tree belong to Module n;

x% in [ ] denotes a variable (a path can be used before a variable)



[Figure3] A DTD syntax tree

Note: Although not every BCM of an XML file can always be described in the DTD syntax tree, what it can describe satisfies a lot of users' needs and in most cases that is enough.

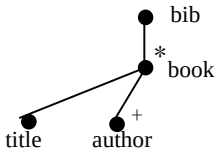
3.3 The automatic creation of BCMs

Now we will discuss how to create the actual BCM from its description in the DTD syntax tree. Based on the XQuery system, we have designed a service that can create automatically a query by the description of each BCM. This query is given in the form of XQuery language and its query result in XQuery is just the actual BCM. Its basic process of the algorithm is explained by an example.

[Example 3] For Mudule1 described in Figure3 (MA=1), the actual BCM can easily be produced by the following process. First, by the tree in Figure3, the sub-tree in Figure4(a) can be got by cutting off all the irrelative nodes (MA< >1). Second, from the sub-tree, the query in List6(a) can be easily created automatically. Third, the query will be executed in XQuery and its result is just Module1, namely List3.

```
<bib> {
  for $b in document("bib.xml")/bib/book
  return
     <book> {
         $b/title,
         for $a in $b/author
         return $a
     } </book>
} </bib>
```

[List6 (a)] The query corresponding to Module1

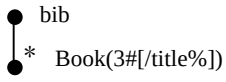


[Figure4 (a)] The sub tree for Module1

In the same way, for Module3, its sub-tree is Figure4(b) and the query is List6(b). This query includes a variable 'title%'. Just before it is executed in XQuery, a value given by the user will replace the variable. So the result of Module3 is dynamic XML information. Similarly, any CBCM can be produced in the same way as BCM.

```
for $b in document("bib.xml")/bib/book
where contains($b/title/text(), &title%)
return $b
```

[List6 (b)] The query corresponding to Module1



[Figure4 (b)] The sub-tree for Module3

3.4 The abstract DTD

To make our XML-IRS work faster, the abstract information about each XML file is necessary. So a new DTD called abstract DTD is given in List7 and can be used in XML files. It can be put in the first part of XML file or can be saved as an independent file.

```
<!ELEMENT abstracts (subject, abstract, keyword*, module*)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT abstract (#PCDATA)>
```

```
<!ELEMENT keyword (#PCDATA )>
<!ELEMENT module (ma, description, variable*)>
<!ELEMENT ma (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT variable (#PCDATA)>
```

[List7] The abstract DTD

[Example 4] For bib.xml in Example1, the abstract description in List8 might be included in it.

```
<abstracts>
   <subject> Data on the Web </subject>
   <abstract>
      This book discusses the various representations about the data on the
      Web such as Relational Databases, Object Databases, etc and the
      syntax for the data.
   </abstract>
   <keyword>Web</keyword>
   <keyword> structure </keyword>
   <module>
      <ma>1</ma> <description>the title and the authors of the books
  </description>
   </module>
   <module>
      <ma>2</ma> <description>the table of contents of the books </description>
   </module>
   <module>
      <ma>3</ma>
      <description>all the books whose title contains a given string </description>
      <variable>title%</variable>
   </module>
</abstracts>
```

[List 8] The abstract information of bib.xml

4. The change of the browser

To support the DTD-based XML modularization technique, the browser must be extended in two aspects.

(1) The Abstract Function

In our new browser, we add an abstract function in the sub-menu that appears when the user right-clicks the mouse on an XML URL. When the function is selected, the browser will get the abstract information of the XML file from the Web server in a very short time and show it in a sub-window as in Figure5. This function enables the user to avoid opening any irrelevant XML file.

Subject: Data on the Web
Abstract: This book discusses the various representations about the data on the Web such as Relational Databases, Object Databases, etc. and the syntax for the data.
Keyword: Web, structure
Module description:
☐ **1=** the title and the authors of the books
☐ **2=** the table of contents of the books
☐ **3=** all the books that its title contains a given string : [ ].

Content Module

[Figuer5] The sub window for abstract function

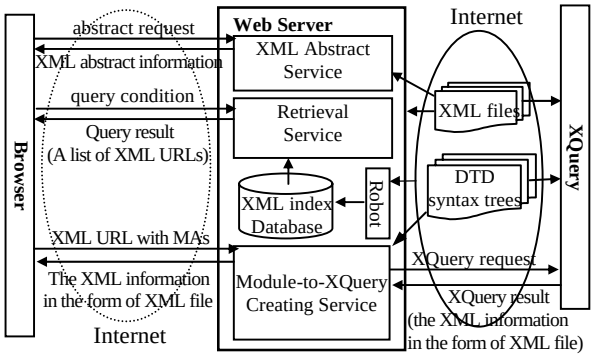
Note: Even if in the case of dial-in terminal connections, it takes about three or four seconds to transfer abstract information with size of 2k.

(2) The Modularization Function

In our new browser, the user can select BCMs by clicking the box option "☐" in Figure5. The browser will send the URL with the selected MAs to the Web server, and receive from the Web server only the XML information corresponding to these BCMs not the whole XML file. Especially, in fields "[ ]" values can be entered and they will be used to replace the variables in queries for dynamic BCMs.

5. The advanced XML IRS framework

With the DTD-based XML modularization technique, we finally build up an advanced XML IRS as in Figure6 and its work process is as below.



[Figure6] The advanced XML-IRS framework

1) The *Web Server* (*Robot)* gathers information on Internet and builds the *XML Index Database* over all XML files on Internet. The abstract information of XML files is also included in it.

2) When receiving a query condition from The *Browser*, the *Web Server (Retrieval Service)* will search information in the *XML Index database*, and return the query result (a list of XML URLs) to the *Browser*.

3) For each URL in the query result, the user uses Abstract Function to decide if he needs to open it. The *Web server (XML Abstract Service)* can get its abstract information from the *XML Index Database* and return it to the *Browser* in a very short time.

4) If not interested in the XML file, the user can go to the next URL. Otherwise, he can select BCMs (maybe he needs to enter a set of values), and then click the Content Module button in the sub-window as in Figure5. The Browser will transfer the URL with the MAs (and the values) to the *Web Server*.

5) The *Web Server (Creating Service)* will create the query by the DTD syntax tree (maybe the entered values will replace all variables in the query) and send it to *XQuery*.

6) *XQuery* gets the XML information satisfying the query from the XML file and returns it as an XML file to the *Web Server*. In turn, it will be wrapped in an HTML file, transferred to the *Browser* and finally shown to the user in the form of XML file.

6. Conclusion

Our new XML-IRS is given by combining the current mature HTML-IRS techniques with the XQuery system. It has the following advantages. 1) Unlike the RDBMS-based systems [7,8], it can retrieve XML information among all the XML files on Internet, getting all the XML URLs related to a given query condition. 2) Compared with the XML query language [2,6,9] based

systems, it is more user-friendly because it dose not require the user to command any XML query language and to know the internal structures of all the XML files. 3) It works effectively for two reasons: it avoids transmitting and browsing any useless XML file, and it transfers only the needed part in an XML file to the user. So it is more suitable to the users of mobile devices or dial-in terminal connections.

7. REFERENCES

[1] Extensible Markup Language (XML) 1.0(Second Edition). W3C recommendation, 6 Oct 2000. http://www.w3.org/TR/2000/REC-xml-20001006

[2] XQuery 1.0: An XML Query Language. W3C Working Draft, 20 Dec 2001.

http://www.w3.org/TR/2001/WD-xquery-20011220.

[3] HTML 4.01 Specification. W3C Recommendation, 24 December 1999.

http://www.w3.org/TR/1999/REC-html401-19991224.

[4] Zhu, Z. Y., et al. A Technique for Browsing HTML files Faster Based on its Block Content. COMPUTER ENGINEERING AND APPLICATIONS (Chinese Magazine), May 2002, 38(10): 192-194.

[5] IBM Alphawork. XML Parser. http://www.alphaworks.ibm.com.

[6] Böhm, K., On Extending the XML Engine with Query-Processing Capabilities. Proceedings of the IEEE Advances in Digital Libraries 2000 (ADL 2000).

[7] Cheng, J. and Xu, J., XML and DB2. Proceedings of the 16th International Conference on Data Engineering. 1998.

[8] Bertino, E. and Catania, B., Integrating XML and Databases. IEEE Internet Computing. Vol.5, No.4. July-August 2001: 84-88.

[9] Ishikawa H., et al. The Design of a Query Language for XML Data. Proceedings of the 10th International Workshop on Database & Expert Systems Applications. 1-3 Sept, 1999, Florence, Italy.