7th International Conference on DEVELOPMENT AND APPLICATION SYSTEMS
Suceava, Romania, May 27 – 29, 2004

QUERYING WEB DOCUMENTS TO RETRIEVE DISTRIBUTED DATA

Diana CIUPEC[1], Ştefan-Ciprian TANASĂ[2]
"Al. I. Cuza" University of Iaşi
Faculty of Computer Science
Str. G-ral Berthelot Nr. 16, RO-700483 Iaşi
[1)] *dgorea@infoiasi.ro,* [2)] *stanasa@infoiasi.ro*

Abstract. *Managing and querying web documents has been one of the most studied problems in recent years. In our paper we present a method of querying web pages using a XML query language. It is known that although most web pages are published in a non-XML format, there are means to convert a HTML document in a XHTML one, which conforms to XML syntax. We provide a few case studies for three different types of web queries: structure, content and hybrid. We propose an extension of XQuery language in order to support patterns for URLs. This extension can be used to recursively traverse an entire Web site.*
Keywords: *XQuery, web pages, search engine, XHTML, XML, web programming.*

Introduction

Today, a huge quantity of information is available through web pages. Therefore, a matter of research interest is developing a database management system to suit the needs of the Internet. The relational model seemed to be ineffective when dealing with problems like querying web pages, for example. Thus, a technique to integrate data from heterogeneous sources, in particular web pages and another data models, is needed. The solution is to publish data on the web in a standard format that can be queried in a common way. On the other side, querying web data raises other problems because of the wide distribution of data, unstructuredness, redundancy. Since today web pages contain significant information, in this paper we focus on the problem of querying both the structure and the content of web pages.

Today the common approach in querying web pages is keyword-based search through one or more search engines. This kind of querying is considerably weaker than database-like querying. On the other side, it would be impossible to query the entire web, like in a DBMS, because of the lack of schema, the dynamism of the network and the combination of form and content.

While the Web is dominated by HTML-style web pages, nevertheless the future is in a richer mark-up, which offers separation of information (content) from the way information is published (presentation). The languages that offer this richer markup are XML ([1]) and XHTML (strict version) ([2]).

However this does not limit the generality, we still can query old HTML pages ([3]) even if their syntax is not XML-like. There are tools like *tidy* ([4]) which translate a HTML document in a XHTML one.

XQuery - a language for querying web pages

At W3C (World Wide Web Consortium [5]) request, a language designated to fulfill the XML querying needs is developing (XQuery [6]). It is now the official recommendation and the vendors have already developed XQuery implementations and support in their database management systems (such as Tamino Server 4.1. [7]).

IBM and Oracle is now developing a Java API which connects XQuery with Java language (*javax.xml.xquery*). Also, the open-source community has come with some XQuery processing engines (QEXO-GNU XQuery Implementation [8], Galax [9]).

XQuery is a very expressive functional language with a simple and familiar syntax, which offers the possibility of natively manipulate XML data structures. A query is an expression that has to be evaluated and its result can be mixed with other expressions. From this point of view XQuery is the most convenient way of integrating more XML interfaces.
The XQuery data model is defined in terms of a formal data model, not text, although XML files are text. Each input and output for a query is a tree of nodes, similar to DOM trees. The types of possible nodes are: element, attribute, text, namespace, processing instruction etc.
The XQuery language is suitable both for structure queries and for content queries. A XQuery query can iterate over the document tree and retrieve nodes with a specified property, perceive the hierarchy relationships between nodes, perform semijoin or combine data from multiple documents. Furthermore, it is possible that the result of a query to be input for another one, like in pipeline systems.
The expressions are evaluated depending on a context node or atomic value.
Like in a template processor, a query can contain both element constructions and XQuery expressions. The curly braces insert an XQuery expression in an element constructor.
The main capability of the XQuery language is given by FLWOR (*For-Let-Where-Order-Return*) expressions, which is similar to SQL *select-from-having-where*.

- *For* constructs a list of bindings of a variable with values.
- *Let* associates a list of values to a variable.
- *Where* sets a filter for the list specified in a *for* construction.
- *Order-by* specifies a sort criterion for the list established in a *for* construction.
- *Return* inserts for each list item an element constructor in the result.

Several use cases and requirements indicate the XQuery ability to query without schema knowledge, to operate on hierarchy, to transform input structures and create new structures and to preserve the initial order and hierarchy of nodes.

Querying Web documents

Querying Web pages is a very useful issue because a local search engine can use the information retrieved. Also, another software module can process the obtained data for different purposes such as statistical ones.
Let us consider a scenario in which researches of a university have published on Web their curriculum vitae (resume). A query might extract all titles of articles written in the last two years or the number of books for each researcher.
There are two kinds of basic queries on Web pages: structure and content queries. These two types can be mixed into a third type, which is hybrid query.
A simple structure query can be:

```
<keywords>
{
let $site := document(
  “http://www.infoiasi.ro/book.html”)
for $bold in distinct-values( $site//b)
  order-by $bold/text()
      return <kw>{$bold/text()}</kw>
}
</keywords>
```

The above query constructs a part of an XML document, which contains a list of strings appearing in bold format in the HTML source.
In this paper we choose the XML format for the results of queries because it is more general and can be used by other applications or published in HTML format via a XSL transformation.
A simple content query is:

```
<appearances>
{
let $site := document(
  “http://www.infoiasi.ro/book.html”)
for $v in $site/html/body//*
  where contains($v/text(),
    “Web programming”)
  return <ap>{$v/text()}</ap>
}
</appearances>
```

This query performs a keyword search in the *body* section of the specified HTML document. The context in which the keyword appears is returned.

Another example we give is a hybrid one that traverses the links in a document.

```
<images>
{
let $d := document(
  "http://www.uaic.ro/book17/toc.html")
for $a in $d//a
   where contains($a/text(),"chapter")
  let $i := document($a/@href)//img
  return
     <chapter imgs="{count($i)}">
        {$a/text()}
     </chapter>
}
</images>
```

This query iterates over the table of content of a book and for each chapter counts the number of images.
Here is another example of query, which recursively traverses documents in a Web site to search a specific string in first three paragraphs. In order to achieve that we define a recursive function that follows the links from the start page.

```
define function traverse(
  $e as xs:element) as xs:anyType
{
  let $l1 := for $p in $e//p[
    position() < 3 ]
  where fn:contains($p/text(),
    "Web programming")
  return $p

  let $l2 := for $a in $e//a
  return traverse(document($a/@href))

  return $l1, $l2
}

<results>
{
   traverse(
     "document(http://www.uaic.ro/)")
}
</results>
```

The function *traverse* iterates over the first three *p* elements and retains in a list the elements that contain the string *"Web programming"*. After that it extracts all links and for each of them makes a recursive call. The results of the recursive calls are then concatenated and returned.
We notice that, in order to retrieve elements with a certain property from an entire site, we have to define recursive functions which effectively implement this traverse. It would be easier to have an embedded mechanism for recursive processing.
We can easily observe that our traversal function doesn't check if there are loops or if it visits a document twice.

Proposal for XQuery extension

We propose a better way to refer documents from Internet using patterns for URLs. Also, we propose an extension from the node references to file systems. Some examples are bellow:

```
document(
"http://www.infoiasi.ro/book19/*.html")
//p[align="justify"]
```

Select all paragraphs that are align justify from the HTML documents located in *book19* directory.

```
document(
"http://www.uaic.ro/~stanasa//*.html")
/h1
```

Select all *heading 1* elements from HTML documents that are published by *stanasa* user.
If the *www.infoiasi.ro* is stored on the local machine, the list of HTML documents can be obtained from local file system. Otherwise we can invoke queries for search engines in order to get the initial list of HTML documents. The final list will be obtained by recursively processing the HTML documents from initial list.
Using the proposed patterns it is easier to extract data recursively. The XQuery code becomes shorter and we do not have to define recursive functions or implement algorithms for graph crossing. This can be done in XQuery implementation.
A more complex example can be:

```
document("http://*.info.uaic.ro")
//*.xml
```

This expression selects all XML documents from sites having the suffix *info.uaic.ro*, such as *vidar.info.uaic.ro*, *thor.info.uaic.ro*, *www.info.uaic.ro*.
A site can be indicated by the IP address. So, we can also use patterns for IP addresses:

```
document(
  "ftp://193.231.30.*/pub//stud?.xsl")
  //xsl:value-of
```

In this example we use another protocol than HTTP, and apply the "*?*" wildcard, which substitutes only one character.

Conclusions

XQuery provides support for advanced local and world wide search. The search can be done depending on the structure of the documents.
We can extract different information from a lot of locations from the entire Internet, using protocols such as HTTP and FTP. So, we can easily make reports based on distributed data from intranet/extranet.
The XML output facilitates creation of particular Web services. We can offer query results for Web or standalone applications using SOAP ([10]). A HTML document can be obtained via a XSL transformation.

Using patterns for URLs and the extension discussed above, the XQuery code becomes shorter and easier to write.

Future work

First we plan to implement the support for the extension of XQuery discussed in this paper.
Also, we plan to develop a Web application for data extraction from local file system and from an intranet. After that we can extend this application to the entire Internet.

References

[1] W3C, *XML Home Page*, http://www.w3.org/xml
[2] W3C, *XHTML Home Page*, http://www.w3.org/TR/xhtml1
[3] W3C, *HTML Home Page*, http://w3.org/MarkUp
[4] Tidy Home Page, http://tidy.sourceforge.net
[5] W3C, http://www.w3.org
[6] W3C, *XQuery-An XML Query Language*, http://www.w3.org/TR/XQuery
[7] *Tamino Server Home Page*, http://www.softwareag.com/tamino
[8] *QEXO Home Page*, http://www.qexo.org
[9] *Galax*, http://db.bell-labs.com/galax
[10]*SOAP*, http://w3.org/TR/SOAP