8th International Conference on DEVELOPMENT AND APPLICATION SYSTEMS
S u c e a v a, R o m a n i a, M a y 25 – 27, 2 0 0 6

AGENT-BASED SEARCHING SEMI-STRUCTURED DATA ON THE WEB

Sabin-Corneliu BURAGA
"Al. I. Cuza" University of Iaşi, Romania – busaco@infoiasi.ro

Abstract. *The paper present a Web agent designed to search semi-structured documents. These documents consist in a set of HTML and XML mark-ups (tags). To properly store the found documents it can be used our defined XML-based language called WQFL (Web Query Formulating Language). Also, this language can be used to formulate structured queries for Web search activity. The Web agent is implemented in Java language and uses Jess – a script language and a development environment for expert systems.*
Keywords: *Web, search, semi-structured data, XML, Web agent.*

Introduction

The World-Wide Web, the world's largest hypertext structure, has already in the last years a huge expansion. Many worldwide consistent or semi-consistent collections of scientific data, in particular disciplines, have become accessible on Internet, especially via several types of Web sites: forums, portals, wikis, Weblogs, etc. [2] These sources of information comply with a standard for interoperability. The data may also conform to a common semantics (each item of particular data has a precise formal definition).
Regardless of many theoretical and technical advances, it is harder to discover the existence of appropriate data needed for a particular problem and the most frequent approach in searching information, on Web mainly, is the keywords based method. Different Web robots perform this search activity. The growing of information available on Internet shows the weakness of the traditional Web search techniques. As a result, increased usage of online search by non-specialists has increased the need for a more effective and friendlier search experience.
Search services generally can be distinguished according to how they accumulate and organize their meta-information [4, 6]: *automatic acquisition and indexing* (performed by Web robots) and *manual acquisition and categorization* (accomplished by trained information personnel). Actually, these methods are not adequate.
The paper presents a Web agent designed to search semi-structured documents. These documents consist in a set of (X)HTML or XML mark-ups (tags).
To properly store the found documents, it can be used our defined XML-based language called *WQFL* (*Web Query Formulating Language*) – detailed in [3] and [4]. Also, this language can be used to formulate structured queries for Web search activity.
The Web agent is implemented in Java language that assures the platform-independence of the application. Also, the agent can use different modules written in *Jess* – a script language and a development environment for expert systems. To process WQFL documents the agent uses *JDOM* library [9], an open-source implementation of Web Consortium's DOM (Document Object Model) [10] recommendation.

Semi-Structural Search: General Presentation

The following practical situations are observed [3-4, 6]:

- Compound queries are using Boolean connectors (such as *and*, *or*, *near* or *not* operators used by all actual search Web engines).
- The users would like to retrieve particular Web documents that have diverse data structures (e.g. without tables, only seven pictures to be placed on bottom of the desired Web page and so on).
- The search activity can be more productive by using different AI methods.

In the first phase of search process, the user is going to formulate a query. By using different Web user-interfaces, we can design textual or graphical complex queries. The users need to be able to formulate complex and flexible queries such as “conference” + “proceeding” + without applets + without sounds + with <3 tables on top + <7 paragraphs. This query can be formulated by an XML-based language called WQFL (Web Query Formulating Language) [4]. This markup language will be presented later in this paper. For each found page, the agent will generate a WQFL document.

The given keywords (e.g. “conference”) will be used effectively by the search engine (for example, Google), thru the available Web service interface to return first *N* significant pages and the remaining expressions (e.g., with <3 tables on top) will be processed in the activity of semi-structural search by the Web agent.
The second part’s objective is to encode the Web pages structural information. According to the given potential of WQFL language, some users would like the graphical content to be found on top of the Web pages and maximum 7 paragraphs etc. That information is stored into WQFL documents. The agent will process each found Web document and it will retain only the position (top, middle, and bottom) and the occurrences of some (X)HTML elements and attributes (e.g., <p>, <table>, <img>, etc.).
For example, the software agent can use the following HTML elements (tags) to perform the semi-structural search activity [6]:

- <p> (paragraph);
- <img> (still image; JPEG, GIF, or PNG file image formats);
- <object> (multimedia – sound, movie, animation, 3D virtual world – or generic object, such as ActiveX component);
- <table> (tabular data);
- <a> (anchor: denotes a hyperlink);
- <script> (script code, such as JavaScript or VBScript programs);
- <applet> (Java applet).

For each element, the agent will retain three values that represent the occurrences of that element on top, middle and bottom of the Web page. The generated WQFL documents will be stored locally.
Later, another software tool can perform a semi-structured search by using a query language such as XQuery [10].

Web Query Formulating Language (WQFL)
This section describes first the syntactic constructs (elements and attributes) of the WQFL language, by using the DTD formal rules.
The following DTD defines some of the constituents of the WQFL language:

```
<!-- WQFL elements -->
<!ELEMENT webquery
        (engine+,query,structure,page*)>
<!-- web query information -->

<!ELEMENT engine  (#PCDATA)>
<!-- search engine -->
<!ELEMENT query   (#PCDATA)>
<!-- query expression -->

<!ELEMENT structure        (element*)>
<!-- structural data -->

<!ELEMENT element(occur*)>
<!-- elements data -->

<!ELEMENT occur   EMPTY>
<!-- position occurrences -->

<!ELEMENT page             (#PCDATA)>
<!-- found page(s) information -->

<!-- WQFL attributes -->

<!ATTLIST webquery
  timestamp   PCDATA      #IMPLIED
  <!-- query timestamp -->
  maxpages    NUMBER      #IMPLIED
  <!-- maximum number of found pages -->
  language    PCDATA      #IMPLIED
  <!-- Web pages language(s) -->
>
```

```
<!ATTLIST engine
  url           PCDATA      #REQUIRED
  <!-- search engine URL -->
  info          PCDATA      #IMPLIED
  <!-- additional information
     (e.g. use a specific language) -->
>

<!ATTLIST element
  name          PCDATA      #REQUIRED
  <!-- stored element name (e.g. <a>) -->


 order          NUMBER      #IMPLIED
 <!-- order of relevance -->
 appear         yes|no  yes
 <!-- the element must appear
     (position don't matters) -->
 context        PCDATA      #IMPLIED
 <!-- context of element occurrence
     (e.g. parent tag) -->
>

<!ATTLIST occur
  top           NUMBER      #IMPLIED
  <!-- position occurrences -->
  middle        NUMBER      #IMPLIED
  bottom        NUMBER      #IMPLIED
>

<!ATTLIST page
  url           PCDATA      #REQUIRED
>
```

For each found Web page, the agent automatically generates a WQFL document. The document root element <*webquery*> will include a least one <*engine*> element, a <*query*> element, a <*structure*> element, and a zero or more <*page*> elements.
The <*query*> element will store the effective query expression to be sending to a search Web engine and the <*structure*> element will contain the semi-structural information of the found document (Web page). Some attributes (e.g., *name* or *url*) are mandatory and other can optionally appear (such as *maxpages*, *language* or *context*). The order of significance of an element is given by the value of *order* attribute of <*element*> tag. Other details are presented in [4] and an extension of this language is detailed in [3].

Examples An example of generated WQFL fragment of document follows:

```
<!DOCTYPE webquery
      PUBLIC "-//WQFL 1.0//EN">
<?xml version="1.0" ?>
<webquery
  timestamp="18.04.2006 10:33"
  location=
  ”http://www.site.ro/~document”>
 <engine
   url="http://www.google.com/">
   Google
 </engine>
 <query>conference proceeding</query>
 <structure>
  <element name="p">
   <occur top="7" />
  </element>
  <element name="img" order="2">
   <occur middle="5" bottom="3" />
  </element>
  <element name="a" appear="no">
   <occur top="0"
        middle="0" bottom="0" />
  </element>
  <element name="table">
   <occur top=”1” />
  </element>
 </structure>
</webquery>
```

To validate the document it can be used an XML processor such as Apache Xerces [8].
The found page contains 7 paragraphs on top, 5 images on bottom, no other hyperlinks and only one table.

Benefits The WQFL documents can store (within <*page*> element) the additional information about a certain Web page: location, size, metadata (originator, copyright, owner and so on), language etc. Different XML vocabularies, such as RDF (Resource

Description Framework) or RSS (Really Simple Syndication) [5, 10] can be included, too.

The main benefits are the following: the WQFL documents can store metadata information about found Web pages and WQFL language can used as a standard format for interchange HTML and XML information between Web agents.
Instead of HTML element names, the WQFL documents can include position occurrences information of any XML tags (such as SVG, SMIL, XHTML or MathML elements) and WQFL can be viewed as a shallow query language for XML data.

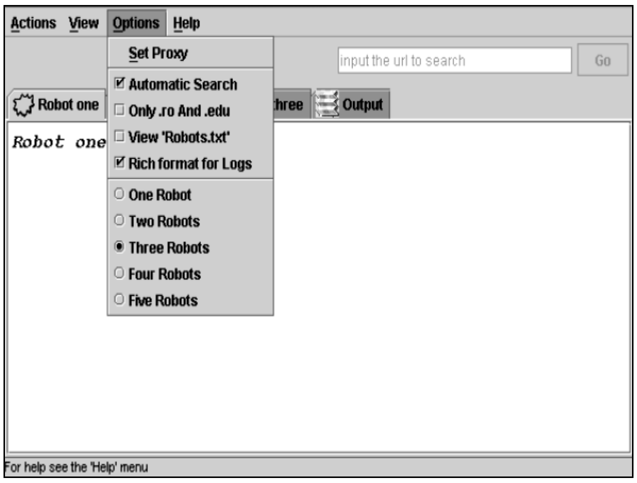

Figure 1. Interface of the Web agent.

Implementation Details regarding the Web Agent

The Web agent will act like a proxy. First, the user's query will be used to interrogate a main search engine (e.g., *Google*, *Altavista*, or *Excite*) and a number *N* of found pages will be returned. For each found page, the agent will generate a WQFL document. These documents can be stored locally, in a native XML database.
Later, the agent can perform a semi-structured search by using a query language, such as XQuery, to discover the desired information.
The Web agent is implemented in Java language that assures the platform-independence of the application. For processing WQFL documents, the Web agent uses *JDOM* (*Java Document Object Model*) library [9], an open-source and flexible implementation of the Web Consortium's DOM recommendation [10].
To process the stored WQFL documents the robot uses *Jess* – a script language and a development environment for expert systems, written in Java and compatible to CLIPS – details in [7]. Jess can be used as a rule engine and can access Java native classes and interfaces.
An example of a Jess program used to perform Web searching follows:

```
; (A_leaf_url url) - just to check
; (A_node_url url) - to check and parse
; A_tmp - internal temp var
; A_this - stores the current Agent object
; C_string - stores the searched string
; C_list_model - the list model
; (watch all)

(set_list_model (fetch C_list_model))
(set-strategy breadth)

(defrule process_node
  (A_node_url ?node_url)
  (not (processed ?node_url))
  =>
  (assert (processed ?node_url))
  (bind ?url (check_url
    (string_to_url ?node_url)))
  (if (neq nil ?url)
    then
      (bind ?new_url (url_to_string ?url))
      (if (neq ?new_url ?node_url)
        then
          (assert (processed ?new_url))
      )
      (log status "Searching:" ?new_url)
      (bind ?http_client
        (get_http_client ?url))
      (parse_url ?http_client)
      (if (contains ?http_client
        (fetch C_string))
        then
          (if (add_address ?http_client)
            then
              (log info "Found:" ?new_url)
            else
              (dispose_http_client
                ?http_client)
        )
       else
        (dispose_http_client
          ?http_client)
     )
  )
)
(run)
```

The interface of the Web agent is presented in figure 1 and uses *Swing*.

Conclusions

A Web agent was presented in this paper. The agent is designed to search semi-structured documents. To properly store the found documents it can be used our defined XML-based language called *WQFL* (*Web Query Formulating Language*). Also, this language can be used to formulate structured queries for Web search activity.

The Web agent is implemented in Java language and can use different modules written in *Jess* – a script language and a development environment for expert systems. To process WQFL documents, the agent uses the *JDOM* library.

Our approach uses an AI approach to enhance the query activity on semi-structural Web resources.

Further work will be focused on automatic categorization of the documents regarding their internal structure expressed by the WQFL constructs.

References

[1] Bray, T. *et al.* (eds.) (2004) *Extensible Markup Language (XML) 1.0, W3C Recommendation*: http://www.w3.org/TR/REC-xml/.

[2] Buraga, S. C. (2005) *Web Site Design* (in Romanian), Polirom, Iaşi.

[3] Buraga, S. C., Brut, M. (2002) *Different XML-based Search Techniques on Web*, Transactions on Automatic Control and Computer Science, vol. 47 (61), No. 2, Politehnica Press, Timişoara.

[4] Buraga, S.C., Rusu, T. (2000). *An XML-based Query Language Used in Structural Search Activity on Web*. The 4th Conference on Technical Informatics – CONTI 2000 Proceedings.

[5] Geroimenko, V. (2004) *Dictionary of XML Technologies and the Semantic Web*, Springer.

[6] Gogan, O., Buraga, S.C. (2000) *The Use of Neural Networks for Structural Search on Web*. In: The 10th International Symposium on Systems Theory – SINTES 10 Proceedings. ROM TPT, Craiova.

[7] Watson, M. (1997) *Intelligent Java Applications for the Internet and Intranets*. Addison-Wesley, Reading MA.

[8] * * *, *Apache Xerces* (2006) http://xml.apache.org/.

[9] * * *, *JDOM* (2006) http://www.jdom.org/.

[10] * * *, *World-Wide Consortium's Technical Reports* (2006) http://www.w3.org/TR/.