A Prototype System for Retrieving Dynamic Content

Denis Shestakov

School of Computer Engineering,
Nanyang Technological University,
Singapore 639798
shestakov@pmail.ntu.edu.sg

Abstract. With the advances in web technologies, web pages are no longer confined to static HTML files that provide direct content. This leads to more interactivity of web pages and at the same time to ignoring a significant part of the Web by search engines (or web crawlers) due to their inability to analyze and index most dynamic web pages. In this paper, we present a prototype system for retrieving dynamic content from pages returned by web forms. The system is based on the form query language that allows to query forms, retrieve data from dynamically generated web pages and store them.

1 Introduction

Current-day web crawlers retrieve content only from a portion of the Web, called the *publicly indexable Web* (PIW) [1]. This refers to the set of web pages reachable exclusively by following hypertext links, ignoring search forms and pages required authorization or registration. However, recent studies [2, 4] observed that a significant fraction of web content lies outside the PIW. A great portion of the Web is hidden behind search forms (lots of databases are available only through HTML forms). This portion of the Web was called the *hidden Web* in [3] and *deep Web* in [4]. Pages in the hidden Web are dynamically generated in response to queries submitted via the search forms.

The dynamic content retrieving process requires the user to perform three steps, i.e., downloading the web page, providing input values and submitting the form (possibly, more than once), and extracting data from the web pages containing the search results. A web application that supports querying of web forms must also perform similar steps: (1) Forms should be stored in some database, after that web applications can process any stored form in the same manner, (2) A high level declarative form query language should be provided to free web applications from relying on form-specific algorithms to query forms, (3) Form query results should be stored in a database to allow us to perform future queries on the basis of stored data, and (4) A powerful extraction tool will be required to extract data from the dynamically generated web pages.

As an example, we consider the task of finding the "best" car dealers in specified cities on the basis of the data available from the car search web site(s).

"Best" dealers may mean dealers that provide lesser prices for interesting car models in comparison with other dealers from the same city. As a rule, a typical web user visiting the car web site tries to find "best" dealers in one specific city and uses a list with the limited number of wanted car models. Thus, a car expert's task of finding the "best" dealers in some cities is only a logical development of the typical user's search task. The car expert could have the list of cities, the list of interesting car models (viz, car make, car model and vehicle age). We assume that these lists are stored in relational tables. The goal of the querying car site(s) is to retrieve hyperlinks to web pages with the "best" dealers contact information for each city from the said list. It should be mentioned that the proposed task can be efficiently accomplished by using an automatic form querying system supported by a robust data extraction technique. Indeed, even for one city and a list with a small number of car models, the process of filling out the car search form(s), their submission and looking through returned results is a very tedious and time-consuming affair.

In this paper we present a prototype system for retrieving content from dynamically generated web pages. The system is aimed at providing such applications as automated web agents searching for specific domain information, hidden web crawlers [3], etc. with expressive query interface to data in hidden Web. The rest of the paper is organized as follows. Section 2 describes how the dynamically generated pages are represented. Section 3 unveils the syntax of a web form query language called `FOQUEL`. In Section 4, we discuss the prototype system's implementation and highlight some experimental results. Finally, Section 5 discusses related work and Section 6 concludes the paper.

2 Result Page Representation

In this section, we describe how the web pages (called *result pages*) returned by a submission of a web form are represented in our proposed system. The result pages are intended to be browsed by humans, and hence their content cannot be easily accessed and manipulated by computer applications. On the other hand, a dynamically generated page represents itself a regular HTML code with a dataset embedded into the code by the web server. Clearly, the successful data extraction tool should extract embedded datasets from generated pages that also contain menu, banners, ads and other irrelevant elements produced by the server.

To overcome the difficulties of dealing with variety of search interfaces provided by the forms, we extract HTML forms and store them in a database specially designed for form storage in accordance with the data model discussed in [5]. Thus, a web agent or application may retrieve a required web form from the *Form Database*. In addition, the web applications are likely to query similar forms multiple times. Hence, storing forms in the database can greatly quicken the whole querying process.

Perhaps the most common case is that a web server returns results a bit at a time, showing ten or twenty result matches per page. Usually there is a hyperlink

or a button to get to the next page with results until the last page is reached. We treat all such pages as part of one single document by concatenating all result pages into one single page. Specifically, we will consider all the result web pages as one web page containing N result matches, where, N may be specified in the web form query by one of the following special keywords: (1) `ALL` (default keyword) - all result matches from each page; (2) `FIRST(`x`)` - first x matches starting with the first result page; (3) `FIRSTP(`y`)` - all matches from first y result pages.

2.1 Result Matches

A code related to a result match is often organized as web table using different web styles, fonts, colors, images and so on. The reason is that most server-side programs generate result pages for human browsing, and hence, are intended to produce an eye-pleasant HTML code. As a matter of fact, only text elements and hypertext links of a result match are informative. Thereby, we ignore an HTML layout of a result match and focus on text strings and hyperlinks embedded in its HTML code. A code that corresponds to a result match can be extracted from a result page (and then stored in the *Results Database*) using an idea of the regularity of HTML patterns related to matches.

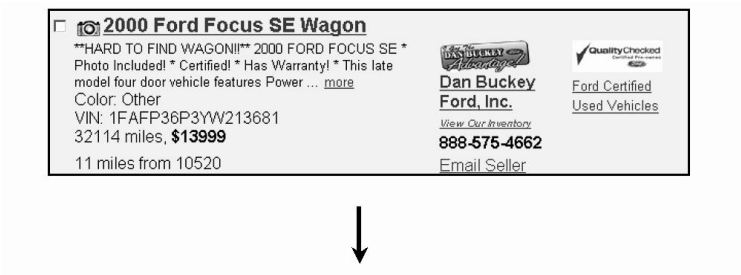
Thus, each result match is represented as a set of text strings and links. Links have their own internal structure similar to the structure of the HTML hyperlink, that is, the link label and the URL of the link[1]. Figure 1 shows the result match of some result page, and the text strings and links corresponding to this match. Here, each result match can be considered as a single row in a table with attributes of two types: *text* and *link*. Any value corresponding to the link type attribute consists of a hyperlink label and the URL of the hyperlink. The default attribute names are $text_i$ or $link_j$, where i corresponds to the number of occurrence of the text element or hyperlink correspondingly in the HTML code related to result match.

Since result matches from the same result page may have different structure (in partucular, different number of text strings or links), the representation of several matches in one table is ambiguous. Actually, the finding of common attributes for all result matches extracting from result pages is a very complicated problem. We suppose a solution of this problem is a theme of additional research. In particular, the work [6] is directly devoted to the problem. We use a modified version of the approach in [6]. The reader may refer to [5] for further details.

Another way is using DEFINE operator that defines extraction conditions for result pages generated by a particular web server. The syntax of the operator is as follows:

```
DEFINE ATTRIBUTE  <type>  <set of attribute names>
       CONDITION  <condition on text> | <condition on label>
             FOR  <form label>
```

[1] Note that if an hyperlink label is an image, the corresponding link label is the text string defined by the `alt` attribute of the `IMAGE` tag or simply the word "*Image*".



$link_1$	$link_2$	$text_1$	$link_3$	$text_2$	$text_3$	$text_4$	$text_5$	$link_4$	$link_5$	$link_6$	$text_6$	$link_7$	$link_8$	$link_9$
Image	2000 Ford Focus SE Wagon	**HARD TO FIND WAGON!!** 2000 FORD FOCUS SE * Photo Included! * Certified! * Has Warranty! * This late model four door vehicle features Power ...	more	Color: Other	VIN: 1FAFP36P3YW213681	32114 miles, $13999	11 miles from 10520	Image	Dan Buckey Ford, Inc.	View Our Inventory	888-575-4662	Email Seller	Image	Ford Certified Used Vehicles

Fig. 1. Result Match Representation

First the operator specifies the type of attribute(s) (`TEXT` and `LINK` correspond to the text and link types respectively). Since more than one link or text may satisfy the extraction conditions, several attribute names may be specified. However, the operator requires at least one attribute name to be specified. The **CONDITION** clause specifies conditions on text strings or hyperlinks respectively of each result match. The satisfied text string or hyperlink will be presented in the *result table* as the value of the column specified by the attribute name. The syntax assumes that each web form has its own set of the result table's attributes. As mentioned earlier, the results database stores an HTML code of each result match. This allows us to define data for extraction from stored results anytime using of the `DEFINE` operator.

3 Web Form Query Language

We now present the syntax and semantics of FOQUEL(**FO**rm **QUE**ry **L**anguage). The FOQUEL is SQL-like and based on a data model presented in [5]. It allows a user to assign more values to form fields than it is possible when a form is manually filled out and perform form queries that require input values from other form queries or relational tables.

The syntax of the FOQUEL is as follows:

```
<query>::= SELECT  [<number of results>]  <set of attribute names>
                   [<set of assigned values>]  [AS <query label>]
           [ FROM  <source set>  ]
           [ WHERE  <assignment set>  ]
           [ CONDITION  <condition set>  ]
```

The `SELECT`, retrieval operator of the FOQUEL[2], consists of four parts: *ex-*

[2] The complete grammar is given in [5].

traction, *source specification*, *assignment*, and *condition* part. The number of results specifies how many result matches should be extracted from the result pages for each submission data set defined in the assignment part. The set of the attribute names can be specified if the result table has been created before by `DEFINE` operator. However, the default attribute names $link_i$, $text_j$ may be used in case the result table has not been created. The **AS** clause specifies that the results of this query will be stored and defines the reference to these results.

Form(s) to be queried, relational table(s) used as a source of input data, the form URL(s) if form(s) is not pre-stored in the form database, and names of stored query results must be specified after the **FROM** clause in the "source specification" part of the `SELECT` operator.

The next section of the `SELECT` statement, the **WHERE** clause defines values to be assigned to the form fields (the fields must pertain to forms specified in the **FROM** clause). Lastly, conditions on the data extracted from the result pages are specified in the **CONDITION** clause.

Example (Query: *Given a list of senior researchers from some Graduate School related to natural sciences*[3]*. Find all works published by these researchers in 2002.)* Suppose all researchers' names are stored in the relational table shown in Figure 2. The `PubMed` form[4] is used to search for published works. The query **Q** can be formulated as follows:

`SELECT` *authors, work, published, pubmed.TEXT*
`FROM` *pubmed, researcher*
`WHERE` *pubmed.db = "PubMed"* `AND` *pubmed.TEXT* = {*reseacher.name,all*}
`CONDITION` *published = (text contains "2002")*

According to the query, results are presented in the four-column table with attributes *authors, work, published, pubmed.TEXT*. The values assigned to the *pubmed.TEXT* define values of the fourth column. These values are taken from the specified relational table `researcher`. We can also specify how many values are used as input to the "*TEXT*" field of the `pubmed` form. `all` defines that all corresponding values of the relational table are assigned to the "*TEXT*" field. In the case of the relational table above the specifying *pubmed.TEXT* = {*reseacher.name,2*} is equivalent to the following assignment: *pubmed.TEXT* = {"*Coffey ET*", "*Kulomaa MS*"}. Figure 2 shows the results of the query Q.

4 Experimental Results

We have created a prototype system based on the methodology described in previous sections. The prototype system performs queries formulated on the FOQUEL and stores query results in the results database. The prototype consists of the following components: *User Interface*, *Web Document Loader*, *HTML Parser*, *Query Processor*, *Extraction Module*, and *Storage/Retrieval Manager* (see Figure 3).

[3] We use data available at `http://www.abo.fi/isb/research_groups.html`.
[4] Available at `http://www.ncbi.nlm.nih.gov/entrez/query.fcgi`

researcher:

id	name
1	Coffey ET
2	Kulomaa MS
3	Lahti R
4	Lahesmaa R
5	Kilpelainen I

authors	work	published	pubmed.TEXT
Coffey ET, Smiciene G, Hongisto V, Cao J, Brecht S, Herdegen T, Courtney MJ.	c-Jun N-terminal protein kinase (JNK) 2/3 ...	J Neurosci. 2002 Jun 1;22 (11):4335-45.	Coffey ET
Karisola P, Alenius H, Mäkkola J, Kalkkinen N, Helin J, Pentikainen OT, Repo S, Reunala T, Turjanmaa K, Johnson MS, Palosuo T, Kulomaa MS.	The major conformational IgE-binding epitopes ...	J Biol Chem. 2002 Jun 21;277 (25):22656-61.	Kulomaa MS
Laitinen OH, Hytonen VP, Ahlroth MK, Pentikainen OT, Gallagher C, Nordlund HR, Ovod V, Marttila AT, Porkka E, Heino S, Johnson MS, Airenne KJ, Kulomaa MS.	Chicken avidin-related proteins show altered biotin-binding and ...	Biochem J. 2002 May 1;363(Pt 3):609-17.	Kulomaa MS
Tirola MA, Mannisto MK, Puhakka JA, Kulomaa MS.	Isolation and characterization of Novosphingobium ...	Appl Environ Microbiol. 2002 Jan;68(1):173-80.	Kulomaa MS
...	...	...	...
...	...	...	...
Karkonen A, Koutaniemi S, Mustonen M, Syrjanen K, Brunow G, Kilpelainen I, Teeri TH, Simola LK.	Lignification related enzymes in Picea abies suspension cultures.	Physiol Plant. 2002 Mar;114(3):343-353.	Kilpelainen I

Fig. 2. Relational Table `researcher` and Results of Query **Q**

The user interface allows a user to specify URLs or local filenames of web pages containing one or more web forms. It calls the web document loader to fetch a web page of some specified URL from the Web. The HTML parser constructs a tree representation of the downloaded web page and passes it to the extraction module. The latter is responsible for extracting HTML forms from a given web page and storing extracted form data in the form database. The FOQUEL part of the user interface allows a user to specify queries on web forms using FOQUEL language. The formulated query is passed to the query processor that parses the query and determines the steps of query execution. Final step transforms the query into HTTP requests which are passed to the web document loader. The query processor is also responsible for deriving different sets of input values from the *Relational Database*, Results Database, and Form Database. The results database stores results of form queries. The returned web page is parsed by the HTML parser, and its corresponding HTML tree is forwarded to the extraction module to retrieve all web pages linked to the returned page. The extraction module analyzes result pages, retrieves result matches, and extracts data from the matches. Finally, the results database is populated by the data extracted from the result pages.

The implementation of our prototype system was conducted on a SUN workstation working under Solaris operational system using Perl and employing MySQL DBMS as the data storage. Totally, 58 forms were stored in the form database. We have been successful with submission of 52 forms. That is, the system was able to process issued query (in average, approximately 105 successful queries per form were issued) in such way that the pages containing relevant query results were generated. The automatically submission of the remaining forms is predominatingly failed due to the number of built-in client-side scripts that are not currently supported by our prototype. In the case of web forms such scripts, often written in Javascript, usually validate the correctness of user input before submitting to the server or define the dependencies between form fields. At the same time, the result match extraction technique requires the further development. The main disadvantage is that each result page is analyzed apart from other connected pages. Thus, the system is sometimes unsuccessful

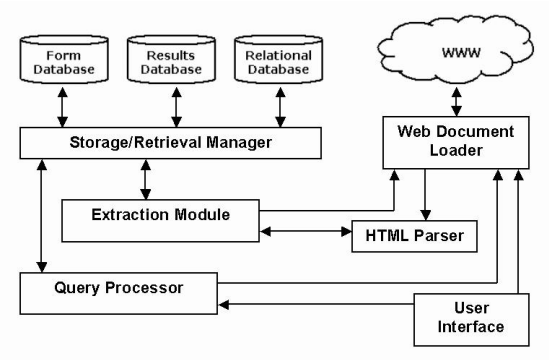


Fig. 3. Architecture of Prototype System

in retrieving content from pages containing a few result matches. However, the extraction module successfully extracts all matches from all connected result pages that contain more than 10 result matches. In the final analysis, our experiments showed that automatic retrieving dynamic content is feasible, and that relatively few forms are queried incorrectly.

5 Related work

In recent years, there has been significant interest in the study of web crawlers. These studies have addressed various issues, such as performance, scalability, freshness, and so on, in the design and implementation of crawlers [9, 10]. However, all of this work has focused solely on the PIW.

W3QS (WWW Query System) [7] is a project to develop a flexible, declarative, and SQL-like Web query language, W3QL. W3QS offers mechanisms to learn, store, and query forms. However, it supports only two approaches to complete a form: either using past form-fill knowledge or some specific values. No other form-filling methods, such as filling out a form using multiple sets of input values or values obtained from queries to relational tables are provided. Furthermore, W3QS is not flexible enough to get all result pages returned by a form.

In [8], Davulcu et al. proposed a three-layered architecture for designing and implementing a database system for querying forms. Our work differs from the work by Davulcu et al. in two aspects. First, we propose more advanced web form representation and user-friendly language for defining form queries. Second, we do not treat the form query results as relations. We represent result pages as containers of result matches, each of which containing informative text strings and hyperlinks.

Raghavan and Garcia-Molina [3] propose a way to extend crawlers beyond the publicly indexable Web by giving them the capability to fill out web forms automatically. Starting with a user-provided description of the search task,

HiWE (Hidden Web Exposer) learns from successfully extracted information and updates the task description database as it crawls. There are some limitations of the HiWE design that if rectified, can significantly improve performance. One of them is lack of support for data extraction from the result pages. Another limitation is lack of query language to query web forms.

6 Conclusion

This paper describes a web form query language for retrieving data from the hidden Web and storing them in the format convenient for additional processing. We presented a prototype system for retrieving dynamic content. Our experiments showed that automatic form querying is feasible and the proposed query language may be used by web applications that retrieve data from dynamically generated pages. Improving the process of filling out forms and more advanced data extraction are essential directions for future work.

Notes and Comments. The paper is based on the Master's thesis [5].

Acknowledgement: I would like to thank a reviewer for valuable comments on this paper.

References

1. Steve Lawrence and C. Lee Giles. Searching the World Wide Web. *Science*, 280(5360):98-100, April 1998
2. Steve Lawrence and C. Lee Giles. Accessibility of Information on the Web. *Nature*, 400:107-109, July 1999
3. Sriram Raghavan and Hector Garcia-Molina. Crawling the Hidden Web. In *Proceedings of the 27th International Conference on Very Large Data Bases(VLDB 2001)*, September 2001
4. Michael K.Bergman. The Deep Web: Surfacing Hidden Value, September 2001. www.brightplanet.com/deepcontent/tutorials/DeepWeb/deepwebwhitepaper.pdf
5. Denis Shestakov. Modeling and Querying Web Forms. *Master's thesis*, School of Computer Engineering, Nanyang Technological University (Singapore), 2002
6. Valter Crescenzi, Giansalvatore Mecca and Paolo Merialdo. RoadRunner: Towards Automatic Data Extraction from Large Web Sites. *The VLDB Journal*, 109-118, 2001
7. David Konopnicki and Oded Shmueli. Information Gathering in the World-Wide Web: The W3QL Query Language and the W3QS System. *ACM Transactions on Database Systems*, 23(4):369-410, 1998
8. H.Davulku, J.Freire, M.Kifer and I.V.Ramakrishnan. A Layered Architecture for Querying Dynamic Web Content. *ACM Conference on Management of Data (SIGMOD)*, 1999
9. Soumen Chakrabarti, Martin van den Berg and Byron Dom. Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery. *8th World Wide Web Conference*, 1999
10. Allan Heydon and Marc Najork. Mercator: A Scalable, Extensible Web Crawler. *World Wide Web*, 2(4):219-229, 1999