STUDIA INFORMATICA 2002
Volume 23 Number 2B (49)

Marek J. FIUK, Robert E. HAUSMAN, Siddartha R. DALAL
Telcordia Technologies
Stanisław WIDEŁ
Politechnika Śląska, Instytut Informatyki

DEVELOPING A TOOLKIT FOR EXTRACTING AND MAINTAINING DATA FROM WEB PAGES

Summary. In this paper we presented a toolkit for extracting and maintaning data from web pages. There are two situations in which the automation may prove useful. First, for those WEB pages in which there is repeating structure, e.g., a large table of data in which each row is a collection of similar attributes for a different object. Second, those WEB pages in which the data changes often, but for which the structure stays relatively unchanged.

METODY POZYSKIWANIA I AKTUALIZACJI DANYCH ZE STRON WWW

Streszczenie. W artykule przedstawiono metodę automatycznego pozyskiwania i aktualizacji danych pobieranych w witryn WWW zapisanych w postaci HTML. W ramach prowadzonych prac autorzy opracowali szereg prototypowych aplikacji. W zaprojektowanym oprogramowaniu wyróżniono warstwę bazodanową zależną od zastosowania oraz warstwę ekstrakcji danych. Metodę extrakcji danych oparto na opracowanym specjalnie przez autorów zapisie nazwanym LPath (Location Path).

1. Introduction

Every day, more and more data is posted to the WEB. Alas, most of it is presented in a manner that is optimized for visual perusal rather than systematic structured analyses. One approach to facilitating such analyses is to automatically extract data from WEB pages and use it to populate local databases upon which such analyses may be performed easily. Our

objective was to create a robust tool for automating such data extraction from WEB pages and then populating databases with as much ease as possible please find fig. 1.

Not all WEB data is equally amenable to such automated collection. If the data on the page is relatively unstructured (e.g., numbers in a paragraphs of text) and the contents of the page never change, then there is no point in training any automated system to go out and collect the information. Any such training will be at least as time-consuming as simply copying the data into the database once.

On the other hand, there are two situations in which the automation may prove useful. First, for those WEB pages in which there is repeating structure, e.g., a large table of data in which each row is a collection of similar attributes for a different object. Second, those WEB pages in which the data changes often, but for which the structure stays relatively unchanged. In both these cases, it is the repetitive nature (repeating structure on the page in the first case, and repeating structure over time in the second) on which we hope to capitalize. Of course, the greatest benefits will accrue for pages that exhibit both forms of repetitive structure.

One application that we constructed had both of these attributes. This application tracked bids and offers in various bandwidth auctions. The pages containing the data typically had tables in which each row represented a bid or an offer. These tables had many rows and were updated frequently. Our application would track the changes by frequently revisiting and reanalyzing the pages.

It is important to notice that capitalizing on repetitive structure requires that the structure be recognized by the automated procedures as repetitive. That is, the ability to recognize structure must be robust. For example, if a particular piece of data moves from one location on a page on one day to another on the next day, we would like our procedures to recognize that and continue to pick up that piece of data correctly.

We began this work in 1999. In the future, we expect that XML will make these goals much easier to achieve. Data will often be labeled and easy to pick out. However, at this time, most pages still consist simply of text formatted by HTML. Therefore, our methods were designed to support any tree-structured format for documents and our implementation was based on DOM[1] trees constructed from HTML.

Also, we were most interested in collecting numbers, often with units, and our work reflects that orientation.

Finally, many of the pages of interest could not be accessed by simply providing a URL. For example, many of them required that a login procedure be executed before access was

[1] The Document Object Model (DOM) is an API from the W3C (World Wide Web Consortium) for facilitating the manipulation of HTML and XML documents. DOM trees have nodes corresponding to the HTML or XML elements.

granted. To deal with this issue, we used a tool developed by Telcordia Technologies. To train this tool to get to a particular page, the user simply navigates through whatever screens or procedures are necessary while the tool keeps track of the process. A unique ID is then associated with this process and the process can be played back automatically as needed by simply supplying that ID to the tool.

2. Application Structure

Over the course of this project, we considered and built prototypes for several different applications of the technology. Anticipating such evolution, we initially chose to split each application into two pieces, 1) an application-specific database piece implemented in Microsoft Access and 2) the data extractor itself, a DLL written in C++. Although the data extractor included a rudimentary means of navigating to an arbitrary URL, we found that to be insufficient in that it did not provide a means for automatically navigating through logon screens and the like. As a result, in the later applications, we integrated as a third piece a more general web navigation engine developed by Telcordia.

The applications comprise two major processes, training and retrieving. The training process proceeds as follows. Upon creating a new entry in the database, the user browses the web using the web navigation engine to bring up the relevant page. The navigation engine then stores the procedure to reach that page and returns a unique page ID to the main application. That page ID is stored with the entry.

The web page is then passed to the data extractor and the user uses the mouse to identify fields on the page. The data extractor creates an LPath (Location Path) for each field and returns it to the main application where it is stored with that field entry. The LPath specifies where to find the data on the page. XPath is a similar language, standardized by the W3C, for addressing parts of an XML document. We developed LPath as a simpler alternative. LPath has the ability to easily use information in one part of the page to index amongst entries in another part of the page. It also has a variety of specialized text parsing operators.

The user also has an opportunity to edit the LPath, typically in an effort to make it more robust against future modifications of the web page. Automating this aspect of the process remains an open issue. In addition, the user may replace an index by a special variable, indicating that the LPath can be used to identify multiple similar pieces of information (e.g., entries from the second column for each row in a table).

The retrieving process may be initiated manually, or in an automated periodic fashion. Based on the data stored while training, it proceeds as follows.

The entry page ID is passed to the navigation engine, which opens a browser and navigates to the appropriate page. The LPaths for each field are then passed to the data extractor, which evaluates each one in the context of the current page and returns the resulting data for storage in the main application.

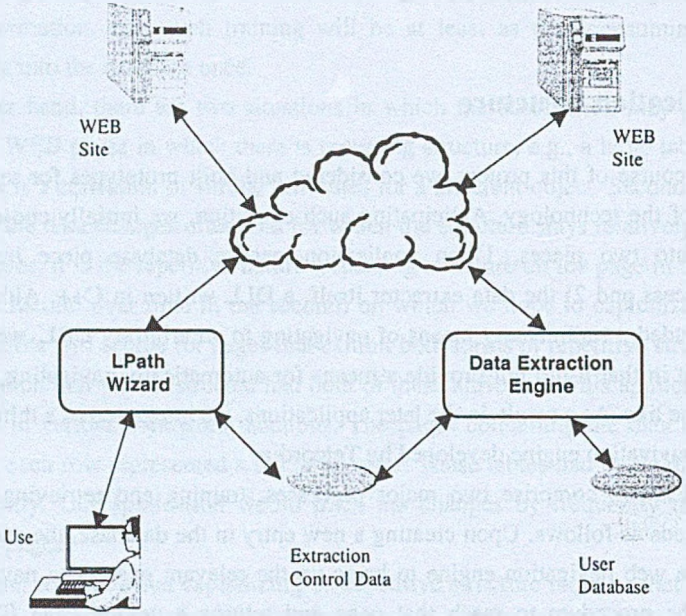

Fig. 1. Architecture of the WEB data extraction system
Rys. 1. Architektura systemu pozyskiwania danych z witryn WWW

3. LPaths

As described in the previous section, LPaths are used to identify a unique DOM node or block of text. For example, suppose that in the middle of a web page, we have the following table 1.

Table 1

Example of the HTML table

Mid Size Cars			
Model	Engine	Weight	MSRP
Ford Taurus	3.0 l	3354 lbs, 1651 kg	$21,500
Accura Legend	3.2 l	3287 lbs, 1576 kg	$26,000
Ford Focus	2.6 l	2475 lbs, 1207 kg	$14,250

The HTML code producing this table you can find on fig. 2.

```
<HTML>
<BODY>
<TABLE>
      <CAPTION><B>Mid Size Cars</B></CAPTION>
      <TR>
         <TH>Model</TH>
         <TH>Engine</TH>
         <TH>Weight</TH>
         <TH>MSRP</TH>
      </TR>
      <TR>
         <TD>Ford Taurus</TD>
         <TD><CENTER><FONT COLOR="darkgray">3.0 l</FONT></CENTER></TD>
         <TD><B>3354 lbs, 1651 kg</B></TD>
         <TD><FONT COLOR="gray"><B>$21,500</B></FONT></TD>
      </TR>
      <TR>
         <TD><CENTER>Accura Legend</CENTER></TD>
         <TD><B><FONT COLOR="darkgray"><CENTER>3.2 l</CENTER></FONT></B></TD>
         <TD>3287 lbs, 1576 kg</TD>
         <TD><B><FONT COLOR="gray">$26,000</FONT></B></TD>
      </TR>
      <TR>
         <TD><B>Ford Focus</B></TD>
         <TD><FONT COLOR="darkgray"><CENTER>2.6 l</CENTER></FONT></TD>
         <TD>2475 lbs, 1207 kg</TD>
         <TD>$<B><FONT COLOR="black">14,250</FONT></B></TD>
      </TR>
</TABLE>
</BODY>
</HTML>
```

Fig. 2. HTML implementation of the example table
Rys. 2. Kod w języku HTML dla przykładowej tabeli

If this is the third table on the page, then we can refer to the weight of the Taurus as

```
/TABLE[2]/TR[1]/TD[2]/TEXT/STR()
```

which means navigate to the third table (indices are zero-based), the second row, third cell and select all text found there. This is the LPath that would be produced automatically by the data extractor.

This LPath would work fine on the web page described, but what if the author were to rearrange the tables on the page? We would like to make our specification of the data robust enough to adjust for such movements. So rather than specify the table number, we would rather specify the table of interest as that with the caption "Mid Size Cars". In LPath notation, we write

```
/TABLE(CAPTION/TEXT/STR("Mid Size Cars"))/TR[1]/TD[2]/TEXT/STR()
```

The modifier for TABLE (the condition expression) must be an LPath, in this case CAPTION/TEXT/STR("Mid Size Cars"). This selects the first TABLE node that has a CAPTION with the text string " Mid Size Cars". In general, the modifying LPath will evaluate to true or false and the table selected is the first for which it is true.

If the author were to rearrange the columns in the table, the LPath would need to include a mechanism to determine the index of the desired column by, for example, using its heading. This can be achieved using the query path mechanism as follows

```
/TABLE[2]/TR[1]/TD[^TABLE/TR[0]/TH[?]/TEXT/STR("Weight")]/TEXT/STR()
```

Instead of indexing the cell explicitly with 3, we use an LPath that evaluates to that value. That LPath, ^TABLE/TR[0]/TH[?]/TEXT/STR("Weight"), may be interpreted as "move up to the enclosing table, go to the first row and then try various cells in that row until you find the string "Weight". Since that string is found in the third cell, a 2 needs to be used in place of the ?, and so the LPath returns the value 2 which is used as the index to TD.

Very often information elements on the WEB page can be conveniently identified using some surrounding text or some encompassing HTML tags. For example, the engine size of the Taurus can be uniquely identified as the text in the third table that is enclosed in the <CENTER> tag and is not enclosed in the <FONT> tag. This description can be expressed as the following LPath.

```
/TABLE[2]/(!FONT)$CENTER/TEXT/STR()
```

The expression (!FONT)$CENTER is interpreted to mean travel downward in the DOM tree to the next CENTER node in a depth-first manner ($ denotes depth-first traversal) without passing through any FONT nodes on the way (which is denoted by the exclusion-inclusion expression !FONT).

LPath also includes mechanisms to identify information elements relatively to the surrounding text using a predefined set of higher level textual constructs, such as numbers (can include sign and the decimal point), measures (numbers with certain pre/post-fixes), dates, money (numbers with currency pre/post-fixes), text lines etc. For example, the path

```
/TABLE[2]/TEXT/MESR(POSTFIX/STR("lbs"))[1]/NUMB/STR()
```

identifies the numerical part of the 2nd measure that has postfix "lbs" and is contained in the third table. In the context of our example, it will return the string "3287".

4. Database Application

As the database applications have evolved, we have faced a variety of design decisions. The first was, obviously, the database structure. The first application was designed with a particular set of pages and fields in mind. Our tables were therefore very specific to that particular need. As is typical, over time, we have moved to a much more general structure, in which we have metatables that define the objects, object sets, fields, etc. used to capture the end data. Thus, we can easily create a new application by merely changing the contents of those metatables rather than having to define new tables and relations and modify the code that manipulates them.

We refresh the data by revisiting the pages referenced by the database and performing the extraction again. In order to minimize the effort, we keep a checksum for each page and only perform the extraction if the checksum changes.

Depending on the needs of the application, varying amounts of history need to be maintained. At a minimum, we simply keep the most recent data along with the date and time at which it was obtained. At the other end of the spectrum, we can retain the date and time of each visit to the page along with the values at that time. Typically, we opt for an intermediate level at which we only store the date, time and values whenever the values change.

One other issue that arose in the design was at what level to integrate certain features, the database application or the data extractor? For example, initially we chose to have the extractor return a single value in all cases. So in order to return entries from all (similar) rows of a table, we used %ROW% in the stored LPath to represent an arbitrary row index. Then, in order to update the database, the main application would call the data extractor repeatedly with increasing integers substituted for %ROW% until the call failed. Later, we integrated this iterative ability into the data extractor itself (see comments on %DELTA% below).

5. LPath Wizard

As described previously, the LPath Wizard is used to create the descriptions of the location of the information elements on the WEB page (the LPaths). The LPath Wizard is an interactive tool built around the Microsoft WEB Browser control. Fig. 3. presented the screenshot of the LPath Wizard's main window.

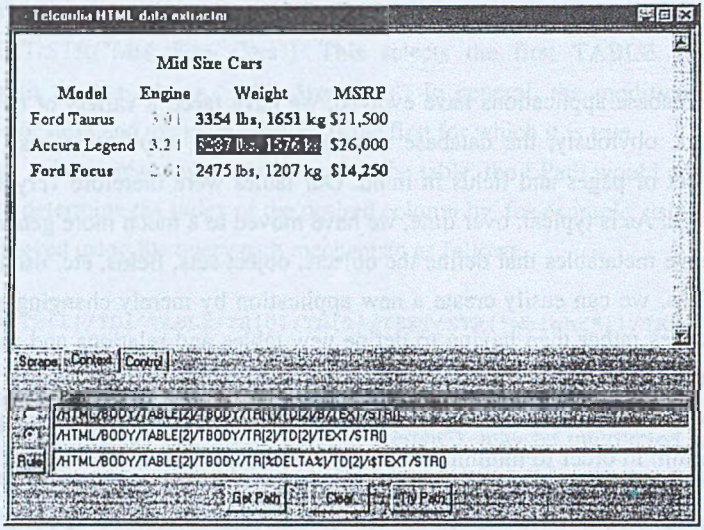

Fig. 3. Screenshot of the LPath Wizard's main window
Rys. 3. Ekran programu kreatora ścieżki LPath

The simplest way of creating an LPath with the LPath Wizard is to highlight the appropriate information element (text) in the browser window and click the GetPath button. This will produce an LPath that evaluates to the innermost HTML node that encompasses the selected text. This LPath can be further manually edited in the path edit control located under the browser window.

In addition to a simple LPath pointing to a single information element, an IteratedLPath can be constructed by replacing exactly one of the indices with %DELTA%. This IteratedLPath can then be used to retrieve multiple information elements from the WEB page.

The LPath Wizard also offers a mechanism for automated creation of IteratedLPaths. First, the LPaths for two example information elements must be obtained (by highlighting) in two path edit controls. Then pressing the Rule button will generate the corresponding IteratedLPath in the third path edit control by replacing the single differing index with %DELTA%. For example, selecting the two following example LPaths:

```
/TABLE[2]/TR[1]/TD[1]/TEXT/STR()
/TABLE[2]/TR[2]/TD[1]/TEXT/STR()
```

will produce (after pressing the 'Rule' button) the following IteratedLPath:

```
/TABLE[2]/TR[%DELTA%]/TD[1]/TEXT/STR()
```

There are often cases in which the two example LPaths are very similar but differ by more than just an index. The LPath Wizard has built-in path transformation rules that allow it to construct (in certain cases) an IteratedLPath despite the additional differences.

One such transformation rule is ignoring or dropping "ornamental" HTML tags such as I (italics) or B (bold). Another is to replace a one node traversal with a possibly multinode, depth-first order, descendants only traversal (i$). If these rules are applied to the following two example LPaths:

```
/TABLE[2]/TR[1]/TD[2]/B/TEXT/STR()
/TABLE[2]/TR[2]/TD[2]/TEXT/STR()
```

then the following IteratedLPath will be produced:

```
/TABLE[2]/TR[%DELTA%]/TD[2]/i$TEXT/STR()
```

Yet another transformation rule is to combine the differing condition expressions. If this rule is applied to the following two example LPaths:

```
/TABLE[2]/TR[2]/TD(B)/i$CENTER/TEXT/STR()
/TABLE[2]/TR[3]/TD(FONT)/i$CENTER/TEXT/STR()
```

then the following IteratedLPath will be produced:

```
/TABLE[2]/TR[%DELTA%]/TD(B|FONT)/i$CENTER/TEXT/STR()
```

Another variation of this rule is to combine the differing nodes using the exclusion-inclusion expression. If this rule is applied to the following two example LPaths:

```
/TABLE[2]/TR[2]/TD/CENTER/i$TEXT/STR()
/TABLE[2]/TR[3]/TD/B/i$TEXT/STR()
```

then the following IteratedLPath will be produced:

```
/TABLE[2]/TR[%DELTA%]/TD/(CENTER|B)i$TEXT/STR()
```

It should be noted that these automatically produced IteratedLPaths are the data wizard's best guess, though they may not always be exactly what is desired. For that reason, we have implemented the Try Path button which will highlight the text indicated by the LPath (with any %DELTA% replaced by 0).

5. Summary

Despite its power, the LPath specification is simple and an interpreter for it can be implemented relatively easily.

While we implemented an LPath interpreter to work on HTML web pages, the specification can just as easily be applied to XML, or, by extension, virtually any tagged tree-based document representation.

Since LPaths are simply declarative strings, they are amenable to programmatic manipulation. For example, two similar LPaths may be used to create an IteratedLPath.

It should also be stated that in an anticipation of the need for the mechanized application of the path transformation rules we have designed the syntax of the LPath notation to be declarative in nature, so that automated analysis and comparison of paths can be performed. This may be more difficult with notations that take form of the computer program (multiple statements).

Collection elements which appear similar on the page (e.g., cells in a table row) are not necessarily similarly situated in the HTML tree. For example, an extra ... tag pair within a table cell will lower the data of interest by one level for that cell. It is important that the specification of data location be robust against such anomalies.

Some form of automated generation of LPaths is necessary. Manual generation simply takes too long and is extremely error prone.

In the future, there are several extensions that we would like to explore. The first of these is to integrate regular expressions into the TEXT section of LPaths.

Secondly, we would like to allow indirect navigation through hyperlinks embedded in the document. For example, in a table of auction bids, one column might contain hyperlinks to pages with more details on each bid. We would like to be able to construct an LPath that would lead to one of these hyperlinks and then continue from the top of that linked page to a particular piece of information on that page.

Finally, we would like to further explore methods of constructing iterated LPaths from example LPaths. In particular, we are interested in developing the concept of equivalence classes of LPaths with or without respect to a particular tree.

REFERENCES

1. Azavant F., Sahuguet A.: World Wide Web Wrapper Factory (W4F) User Manual, January 30, 2000.

2. Hammer J., Garcia-Molina H., Cho J., Aranha R., Crespo A.: Extracting Semistructured Information from the Web. In Proceedings of Workshop on Management of Semistructured Data, June 1997.
3. Lim S.-J., Ng Y.-K.: Extracting Structures of HTML Documents. In Proceedings of the 13th International Conference on Information Networking (ICOIN '98), 1998.
4. XML Path Language (Xpath) Version 1.0, W3C Recommendation 16 November 1999.

Recenzent: Dr inż. Lech Znamirowski

Wpłynęło do Redakcji 30 kwietnia 2002 r.

Streszczenie

W artykule przedstawiono metodę automatycznego pozyskiwania i aktualizacji danych pobieranych z witryn WWW. Metodę ekstrakcji danych oparto na opracowanym przez autorów zapisie LPath (Location Path). Notacja LPath jest relatywnie prosta w implementacji i może być stosowana zarówno do stron zapisanych w postaci HTML jak i języka XML. Najprościej notację LPath można sobie wyobrazić jako sposób adresowania danych w dokumentach HTML. Ponieważ notacja jest ciągiem wyrażeń tekstowych jako taka może być przedmiotem przetwarzana i analizy. Dla przykładu podobne dane zapisane w tablicy HTML mogą być wskazane za pomocą jednej iterowanej ścieżki (IteratedLPath). Notacja została zaprojektowana w celu umożliwienia dokonania automatycznej analizy i porównania ścieżek do danych. Dla typowego zapisu programistycznego czyli sekwencyjnego ciągu instrukcji wykazanie adresowania tych samych danych czy porównanie ścieżek może być zadaniem bardzo trudnym.

Autorzy zdają sobie sprawę, że do praktycznego wykorzystania notacji LPath niezbędne jest zastosowanie odpowiednich narzędzi wspomagających generowanie ścieżek, gdyż ręczne ich wprowadzanie jest podatne na błędy i zabiera zbyt dużo czasu. W ramach prowadzonych prac autorzy opracowali szereg prototypowych aplikacji. W zaprojektowanym oprogramowaniu wyróżniono warstwę bazodanową zależną od zastosowania oraz warstwę ekstrakcji danych.