Empirical Study on Screen Scraping Web Service Creation

Case of Rhein-Main-Verkehrsverbund (RMV)

Mohamadou Nassourou
Department of Computer Philology & Modern German Literature
University of Würzburg Am Hubland D - 97074 Würzburg
mohamadou.nassourou@uni-wuerzburg.de

Abstract

Internet is the biggest database that science and technology have ever produced. The world wide web is a large repository of information that cannot be used for automation by many applications due to its limited target audience. One of the solutions to the automation problem is to develop wrappers. Wrapping is a process whereby unstructured extracted information is transformed into a more structured one such as XML, which could be provided as webservice to other applications. A web service is a web page whose content is well structured so that a computer program can consume it automatically.

This paper describes steps involved in constructing wrappers manually in order to automatically generate web services.

Keywords: HTML, XML, Wrapper, Web service

1. Introduction

One of the biggest disadvantages of the enormous size of information on the Internet is that, users can no longer easily locates what they need. Information agents have been developed to solve this problem by assisting users through some filtering interfaces. Many agents generally rely on wrappers to extract information from semistructured web pages such as HTML pages.

HTML (HyperText Markup Language) is mainly concerned with the presentation aspect of a page. Constucting a wrapper by hand involves analyzing documents to be processed, determining the rules that apply for these documents and finally generating the wrapper procedure.
A wrapper is used to transform an HTML document into an XML document which could be reused for machine-to-machine communication. XML (Extensible Markup Language) has been design for describing, storing and transmitting data.

This paper outlines in details the main steps for constructing a wrapper by hand in order to create an XML document to be delivered as web service to other applications.

The Rhein-Main-Verkehrsverbund (RMV) [4] website is used as the source from which to create the web service. PHP programming language is employed for parsing and generating XML documents to be served as web services.

2. Tasks of a Wrapper

A wrapper performs its task by implementing the following steps:

1. download a web document
2. extract required data from the web page
3. map the data into an XML format or other format

Extraction is done after parsing the HTML document. Usually some basic string matching rules such as regular expressions are applied.

2.1. Type of wrappers

Web pages are rendered for human consumption. They often exhibit some hierarchical structure. Information on web pages are usually separated by delimiters such as tags. With these observations in mind, and considering [1,2], there are basically two types of wrappers:

a) string based wrappers
b) tree based wrappers

2.1.1. String Based Approach

In string based wrappers a web page is viewed as a sequence of tokens and markup tags. The beginning and end boundaries of a given string are used to extract it.

Example:

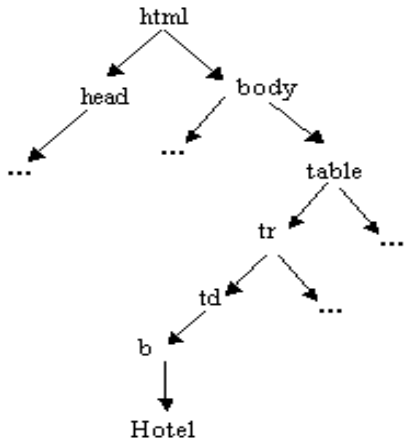
```
 <html><body>
<p><a>Hotel Welcome<i>Germany</i>(German)</a>
<b>Tel: +49</b>(0)xxxxxx</p>
</body></html>
```

2.1.2. Tree Based Approach

In Tree based wrappers web pages are viewed as trees. A simplified figure of a tree representation of an HTML document is shown in fig.2. The nodes in the tree are HTML tags, while the leaves are made of textual content between the opening and closing of the tags.

Example:

```
<html>
<head></head>
<body><table><tr><td><b>Hotel</b></td></tr></table></body>
</html>
```



3. Wrapper for Rhein-Main-Verkehrsverbund (RMV) website

The RMV website provides ample information about transportation in Germany. It offers users the possibility to plan their journey well in advance before departing. Departure time, arrival time, locomotives and other information are available on the website [4].
The wrapper downloads the page using its URL and then extracts needed information such as departure time, arrival time, locomotives. Then it saves the information in XML format for reusability and automation purposes.

A sample of the website is shown in Figure 1.

Übersicht

Details	Bahnhof/Haltestelle	Weg/Karte	Datum	Zeit	Dauer	Umst.	mit	Tarif*
		« früher						
☐	Darmstadt-Wixhausen Bahnhof Darmstadt Nordbahnhof	Umgebungskarte Umgebungskarte	02.02.09	ab 15:49 an 16:11	0:22	1	S	1,95 € 1,15 €
☑	Darmstadt-Wixhausen Bahnhof Darmstadt Nordbahnhof	Umgebungskarte Umgebungskarte	02.02.09	ab 16:19 an 16:36	0:17	1	S	1,95 € 1,15 €
☑	Darmstadt-Wixhausen Bahnhof Darmstadt Nordbahnhof	Umgebungskarte Umgebungskarte	02.02.09	ab 16:49 an 17:06	0:17	1	S	1,95 € 1,15 €
		später »						

Details für Auswahl | Details für alle
Rückfahrt | Weiterfahrt | Erste Fahrt | Letzte Fahrt
Vielfahrer | Verbindungsplaner | Mobil
Druckvorschau Übersicht | Druckvorschau Detailansicht

* Preise für Einzelfahrkarten Erwachsene/Kinder (6 bis einschließlich 14 Jahre). Bitte beachten Sie: Lokale Regelungen können zu Fahrpreisabweichungen führen.

Dieses Symbol zeigt an, dass aktuelle Informationen zu Fahrplanänderungen, Baustellen etc. vorliegen.

Detailansicht

Bahnhof/Haltestelle	Datum	an	ab	Gleis	mit	Bemerkungen
Darmstadt-Wixhausen Bahnhof	02.02.09		16:19	1	S 3	S-Bahn Richtung: Darmstadt Hauptbahnhof Fahrradmitnahme begrenzt möglich
Darmstadt Hauptbahnhof		16:25		3		5 alternative Abfahrten (mit S-Bahn, alle 30 Minuten) Frühere Ankunft
Darmstadt Hauptbahnhof			16:32	8	RB 63/75	RB Richtung: Aschaffenburg Hauptbahnhof Fahrradmitnahme begrenzt möglich, Fahrzeuggebundene Einstiegshilfe: Anmeldung 01805-512512 *, (*14 ct/Min. aus dem Festnetz, Mobilfunk ggf. abweichend), Klimaanlage
Darmstadt Nordbahnhof		16:36		3		7 alternative Abfahrten (mit RB, alle 2 - 60 Minuten) Spätere Abfahrt

Dauer: 0:17; Verbindung besteht: Mo - Sa, nicht 10., 13. Apr, 1., 21. Mai, 1., 11. Jun, 3. Okt

Tarifinformation: 1,95 € Erwachsene, 1,15 € Kinder | Details zum Tarif

Figure 1. RMV web page to be wrapped

Fig. 2 and 3 display the source code of the web page that was wrapped and the resulting XML document respectively.

```
<table class="hafasContentTable" cellspacing="0" summary="HAFAS Routenplane
<tr>
<td class="hafasContentTable">
<form name="tp_results_form" action="/auskunft/bin/jp/query.exe/dn?ld=14.4
<table class="separator1" style="margin-bottom:5px;" cellspacing="0" summar
<tr><td>Ihre Anfragedaten</td><td class="right"><a href="http://www.rmv.de/
<table class="hafasResult" cellspacing="0" summary="Ihre Anfrage">
<tr>
<th class="querysummary">Von</th>
<td class="querysummary screennowrap">Darmstadt-Wixhausen Bahnhof</td>
<th class="querysummary">
Datum
</th>
<td class="querysummary nowrap right">Mo, 02.02.09</td>
<th class="querysummary">Zeit</th>
<td class="querysummary nowrap">
16:11
(Abfahrt)
</td>
</tr>
<tr>
<th class="querysummary nowrap">Nach</th>
<td class="querysummary screennowrap">Darmstadt Nordbahnhof</td>
<td class="querysummary" colspan="4"> </td>
</tr>
<tr>
<td colspan="6">
<div class="hafasHint2" style="padding:0px 2px 0px 2px; margin-left: 0px; f
<div style="width:100%;"></div>
</td>
</tr>
</table>
<div class="separatingLine" style="margin-top:5px;"><hr style="display:none
```

Figure 2. RMV HTML web page source code

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <root>
  - <fahrt>
    - <verbindung>
        <abfahrt>Darmstadt-Wixhausen Bahnhof</abfahrt>
        <abfahrtszeit>10:19</abfahrtszeit>
        <abfahrtsgleis />
        <mittel>S 3</mittel>
        <ankunft>Darmstadt Hauptbahnhof</ankunft>
        <ankunftszeit>10:25</ankunftszeit>
        <ankunftsgleis />
      </verbindung>
    - <verbindung>
        <abfahrt>Darmstadt Hauptbahnhof</abfahrt>
        <abfahrtszeit>10:32</abfahrtszeit>
        <abfahrtsgleis />
        <mittel>RB 63/75</mittel>
        <ankunft>Darmstadt Nordbahnhof</ankunft>
        <ankunftszeit>10:36</ankunftszeit>
        <ankunftsgleis />
      </verbindung>
      <gesamtdauer>Dauer: 0:17</gesamtdauer>
      <preis />
    </fahrt>
```

Figure 3. Resulting XML document after wrapping

3.1. Implemented Algorithm

The wrapper implemented the string based approach to extract the relevant information. It is able to follow links just like crawler functions do.

The following algorithm was used to write the wrapper

1. retrieve initial HTML page from the given URL
2. parse the fetched HTML page
3. extract some of the URLs from the page according to some criteria
4. foreach link in (extracted list of URLs)
    a. fetch the HTML page
    b. parse the HTML page
    c. extract relevant text from the page according to some criteria
5. store extracted data

To retrieve an HTML page I mainly used the CURL library. The syntax of CURL is as follows:

curl_setopt(curl init, curl option, value).

More information on how to use CURL library could be found at [5].

3.2. Practical Example with Sample PHP code

First I defined some variables for the departure and arrival stations:

```
$from = "Darmstadt Hauptbahnhof" ;   // a given departure station called  'Darmstadt Hauptbahnhof'
$to = "Darmstadt Nordbahnhof";        //  a given arrival station called  'Darmstadt Nordbahnhof'
```

then I constructed a query URL for RMV webserver as follows:

```
$URL="http://www.rmv.de/auskunft/bin/jp/query.exe/dn?L=vs_rmv&REQ0JourneyStopsN=-1&REQ0JourneyStopsS0A=255&REQ0JourneyStopsS0G=$from&REQ0JourneyStopsZ0A=255&REQ0JourneyStopsZ0G=$to&start%26date%3D%2B0%26times%3D%2B0%26dummy=jetzt";
```

The follwing CURL code does the connection and the retrieval of the HTML source code

```
$cinit = curl_init();
curl_setopt($cinit, CURLOPT_URL, $URL);
curl_setopt($cinit, CURLOPT_REFERER, 1);
curl_setopt($cinit, CURLOPT_VERBOSE, 1);
curl_setopt($cinit, CURLOPT_FOLLOWLOCATION, 1);
curl_setopt($cinit, CURLOPT_RETURNTRANSFER, 1);
$response = curl_exec($cinit);
curl_close($cinit);
```

The query response is assigned to the PHP variable $response. Then I extracted the relevant information.

As mentioned earlier the string based approach requires the beginning and end boundaries of a string before extracting it.

For instance to retrieve a station name from the query response, I identified the delimiters from the RMV HTML page as follows:

$delimiter_start = 'title="Haltestelleninformation:'

$delimiter_end = '">'

$string_startPosition = strpos($response,$delimiter_start);

$string_endPosition = strpos($response, $delimiter_end);

$station = substr($string, $string_position,$string_endPosition-$string_startPosition);

For each relevant information I had to identify its delimiters then extract it with the above code by substituting its string delimiters in the variables accordingly.

The extracted data is then saved as XML. The names of tags are derived from the semantics of the extracted data.

Example of such an XML document looks like this one:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <root>
  - <fahrt>
    - <verbindung>
        <abfahrt>Darmstadt Hauptbahnhof</abfahrt>
        <abfahrtszeit>10:32</abfahrtszeit>
        <abfahrtsgleis />
        <mittel>RB 63/75</mittel>
        <ankunft>Darmstadt Nordbahnhof</ankunft>
        <ankunftszeit>10:36</ankunftszeit>
        <ankunftsgleis />
      </verbindung>
      <gesamtdauer>Dauer: 0:04</gesamtdauer>
      <preis />
    </fahrt>
  </root>
```

4. Performance

The efficiency of a wrapper must be tested because of its dynamical and regular task to retrieve updated information. It has been found that the time of the wrapping itself (extraction) was the significant part of the total time including the server access and the download of the pages.

5. Robustness

Generally web pages change every month. Wrappers are vulnerable to changes and can be broken, due to their reliance on the HTML tag structure and their content to locate information. However for simple application they could be efficiently used.

6. Conclusion and Future Work

The aim of this paper was to describe the main steps for constructing a wrapper by hand. Manually writing wrappers is a tedious and time consuming task. However the limited number of websites that provide webservices as well as their temporarily unavailability are an undeniable motivation to learn how to write wrappers. In fact most of the websites do not provide webservice due to its complicated technical implementations for an elementary web developer. They prefer publishing all the information on their websites. Therefore wrappers are a good solution to retrieve these information and provide them as webservices to other applications.

The developed wrapper needs some improvements in order to deal with frequently changing websites. Possible further work could be on automatic wrapper generation and induction.

References

[1] Jussi Myllymaki (2001), "Effective Web data Extraction with Standard XML Technologies", in 10th International World Wide Web Conference (WWW10), Hong Kong, May 2001.

[2] L.Liu, W. Han, D. Buttler, C.Pu, and W.Wang (1999), "An XML-based Wrapper generator for Web Information Extraction", in *Sigmod'99*, Philadelphia, 1999.

[3] Kushmerick, N., Weld, D.S., Doorenbos, R.B. Wrapper Induction for Information Extraction, In *International Joint Conference on Artificial Intelligence*. 1997

[4] www.rmv.de

[5] www.php.net