http://webdam.inria.fr

Web Data Management

Putting into Practice: Wrappers and Data Extraction with XSLT

Serge Abiteboul
INRIA Saclay & ENS Cachan

Ioana Manolescu
INRIA Saclay & Paris-Sud University

Philippe Rigaux
CNAM Paris & INRIA Saclay

Marie-Christine Rousset
Grenoble University

Pierre Senellart
Télécom ParisTech

Copyright @2011 by Serge Abiteboul, Ioana Manolescu, Philippe Rigaux, Marie-Christine Rousset, Pierre Senellart;
to be published by Cambridge University Press 2011. For personal use only, not for distribution.

`http://webdam.inria.fr/Jorge/`

Contents

1 Extracting Data from Web Pages 2

2 Restructuring Data 5

Besides languages to extract information such as XPath or XQuery, languages for *transforming* XML documents have been proposed. One of them, XSLT, is very popular. The goal of this PiP is to expose the reader to this aspect of XML and to languages based on tree-pattern rewriting. A presentation of XSLT is beyond the scope of this book. The reader can read the present PiP to get a feeling on standard tasks that are commonly performed with XSLT programs. Of course, realizing the project that is described requires a reasonable understanding of the language. Such an understanding can be obtained, for instance, from the companion Web site of the book, i.e., at `http://webdam.inria.fr/Jorge/`. More references on XSLT may be found there.

XSLT is an XML transformation language. Its principles are quite different from that of XQuery, although they may roughly serve the same purpose: accessing and manipulating XML content and producing an XML-formatted output. In practice, XQuery is used to extract pieces of information from XML documents, whereas XSLT is often used to restructure documents, typically for publishing them in different forms, different dialects. We show in the present PiP chapter how XSLT can serve to write simple "wrappers" for XML pages. This is taking us back to data integration. To integrate a number of data sources, the first step is typically to wrap them all into a uniform schema. Since most data source now export XML, the wrapping technique considered here can be used in a wide variety of contexts. We focus in the PiP on HTML pages from the Web, after transforming them into XML.

Any XSLT processor can be used for the exercises of this chapter. Using an XSLT 2.0 processor, however, will make things much easier: features such as grouping or regular expression matching are of great help in writing wrappers. Therefore, we recommend, for instance, the open-source version of SAXON that is available at `http://saxon.sourceforge.net/` and on the companion Web site. Applying a stylesheet `yin.xsl` to a document `yang.xml` with SAXON is done with the following command line:

```
java -cp saxon8.jar net.sf.saxon.Transform yang.xml yin.xsl
```

1 Extracting Data from Web Pages

We first focus on the extraction of data from Web pages:

1. Choose a Web site that presents semi-structured information about some *entities* such as products, movies, books, or persons. You should choose a collection of pages where data follow a fixed template. To simplify your task, properties of these entities should (at least partly) be clearly presented within the structure of the Web page, as in:

   <li> <b>Chez Chen</b> Chinese food, <i>excellent Beijing Duck</i></li>

   rather than simply given in text:

   Chez Chen. Chinese food, excellent Beijing Duck.

Here are a few ideas, but you can also select your favorite Web sites:

- The Internet Movie Database (IMDb, `http://www.imdb.com/`);
- Amazon (`http://www.amazon.com/`) or any other e-commerce Web site;
- Ethnologue (`http://www.ethnologue.com/`), a resource on all languages of the world;
- The Mathematics Genealogy Project (`http://genealogy.math.ndsu.nodak.edu/`) that gives the scientific adviser of a given researcher in mathematics and related fields;
- DBLP (`http://www.informatik.uni-trier.de/~ley/db/`), or some other research publication database;
- The Yellow Pages service, or other kinds of phone directories.

A solution is proposed on the companion Web site. More precisely, we provide Web pages, wrappers, and extracted data for the IMDb Web site. Depending on your experience in XSLT, you may wish to study them before implementing your own wrapper, refer to them as you progress, or consult them after finishing the exercises to compare the approaches.

2. Select in the Web site you chose a few pages with the same structure presenting different entities. Save these Web pages on disk. Make sure your browser does not try reformatting the Web page: request saving the *Web page only* or *only the HTML*.

3. With rare exceptions, HTML pages from the Web, even when supposedly written in XHTML, the XML-ized variant of HTML, are not well-formed XML documents and cannot be directly transformed by an XSLT processor. We will use the open-source *tidy* utility, available as a command-line tool[1] or through a Web interface[2], to transform HTML Web pages from the wild into well-formed XML documents valid against the XHTML DTD.

   To do this cleanly, we need to set up some options, notably to remove the document type declaration in the output and to replace all named references with numeric references. This is necessary so that the XSLT processor will not need to perform complex tasks such as downloading and analyzing the XHTML DTD. If you use the command line, the syntax is:

```
tidy -q -asxhtml --doctype omit
     --numeric-entities yes file.html > file.xhtml
```

   If you use the Web interface, be sure to set the same options. Do this on all the Web pages you saved. Do not be rebuked by the (usually high) number of warnings!

4. The documents are now ready to be processed. Before doing something too elaborate, write a minimalistic XSLT stylesheet that for instance outputs the title (that can be retrieved with `/html/head/title`) of the XHTML documents you want to process. Remember that XHTML elements live in the XHTML namespace `http:`

[1]`http://tidy.sourceforge.net/`
[2]`http://infohound.net/tidy/`

@ Serge Abiteboul, Ioana Manolescu, Philippe Rigaux, Marie-Christine Rousset, Pierre Senellart, 2011; to be published by Cambridge University Press 2011.

`//www.w3.org/1999/xhtml`. You should therefore either declare an XHTML namespace associated with some prefix and use this prefix before every element name in XPath expressions, or (only in XSLT 2.0) use the `xpath-default-namespace` attribute of the <**xsl:stylesheet**> element to specify the default namespace used in XPath expressions. Test the stylesheet.

5. You should now decide what information to extract from the Web pages. Do not be overly ambitious, start with simple things (e.g., for movies, title, and name of the director). You can do this in an iterative manner (extracting one piece of information at a time, testing it, and then adding another one). Design a DTD for the XML document that will contain the extracted information.

6. For each piece of information, look into the XHTML source for robust ways of identifying where the information is located, using an XPath pattern (since we will use this pattern in a `match` attribute of a template, the context node does not need to be the document root). Element names, *class* attributes, position predicates, and the like are especially useful. For example,

   if the movie title is given inside a `<span class="title"></span>` element, the pattern `span[@class='title']` can be used.

   Write then a template definition in your stylesheet that makes use of this XPath expression to produce a new output element containing the extracted information. Because of the restrictions on the kind of axes that may occur in a `match` attribute, it may be necessary to put part of the pattern there and part of it in an <**xsl:value−of** `select=""`/>.

   In some cases, there is not enough XHTML structure to precisely identify the location of a particular property. When this happens, "regular expressions" can be used in the template definition to only extract only relevant data. For regular expressions,

   - XPath 2.0 provides the `tokenize()` function and
   - XSLT 2.0, the `<xsl:analyze-string>` element.

7. Run your template on each XHTML document and check that each time the output document is valid against the DTD you designed. If not, adapt your stylesheet accordingly.

8. We are now going to modify the stylesheet a little bit (and the DTD) so that it processes all Web pages at a time, resulting in a single output document containing all information. Create a file `list.xml` listing all your XHTML documents with the following structure:

```
<files>
  <file href="a.xhtml" />
  <file href="b.xhtml" />
  <file href="c.xhtml" />
</files>
```

   Modify the stylesheet so that it takes as input document this `list.xml` file and processes in turn (using the `document()` function) each referenced file. The result should be an XML document `data.xml` collecting extracted information from all different Web pages. Adapt the DTD as needed.

2 Restructuring Data

Up to now, the data we obtained closely follows the structure of the original Web source: each item is described one after the other, with all its properties. But it is often needed to restructure extracted information (e.g., to present a list of movies without their cast, and then a list of actors referencing all movies this actor has played in). We now write a second XSLT program to restructure data in this way.

1. Choose one of the properties of your items according to which data will be regrouped (e.g., actors of movies, publication year of books). This will be called the *grouping key*. The final XML file you need to produce should have a structure similar to what follows:

```
<data>
  <item id="id1">
    <property1> ... </property1>
    <property2> ... </property2>
  </item>
  ...
  <item id="id9">
    <property1> ... </property1>
  </item>
  <grouping-key name="...">
    <item-ref ref="id3" />
    <item-ref ref="id7" />
  </grouping-key>
  <grouping-key name="...">
    <item-ref ref="id2" />
  </grouping-key>
</data>
```

   In other words, all items are listed one after the other as before, but the values of the grouping key are not listed among their properties. They are listed separately after the item descriptions, and items that share a given grouping key value are referred to in the description of this grouping key. Write a DTD for the final document. Obviously, choose more explicit element names than `grouping-key` or `item`.

2. Write an XSLT stylesheet that transforms the XML document previously obtained into an identical document, with two exceptions:
   - elements representing items now have an `id` attribute that uniquely identifies them (one can use the XSLT function `generate-id()` for that);
   - information about the grouping key is removed from the document.

3. Modify this stylesheet to add after the list of items the list of all values of the grouping key, without duplicates. This can be easily done in XSLT 2.0 with the <**xsl:for−each−group**> element. In XSLT 1.0, this is more intricate. It can be done, for instance, by expressing in XPath the fact that a specific occurrence of the grouping key value is the first in the document.

@ Serge Abiteboul, Ioana Manolescu, Philippe Rigaux, Marie-Christine Rousset, Pierre Senellart, 2011; to be published by Cambridge University Press 2011.

4. Add to each element representing the value of a grouping key the list of items that have this specific value. Test that the resulting document is valid against the DTD you have designed.

@ Serge Abiteboul, Ioana Manolescu, Philippe Rigaux, Marie-Christine Rousset, Pierre Senellart, 2011; to be published by Cambridge University Press 2011.