Building light-weight wrappers for legacy Web data-sources using W4F

Arnaud Sahuguet
Department of Computer and Information Science
University of Pennsylvania
sahuguet@saul.cis.upenn.edu

Fabien Azavant
École Nationale Supérieure des Télécommunications
Paris, France
fabien.azavant@enst.fr

1 Introduction

The Web has become a major conduit to information repositories of all kinds. Today, more than 80% of information published on the Web is generated by underlying databases (however access is granted through a Web gateway using forms as a query language and HTML as a display vehicle) and this proportion keeps increasing. But Web data sources also consist of stand-alone HTML pages hand-coded by individuals, that provide very useful information such as reviews, digests, links, etc. As for the information that also exists in underlying databases, the HTML interface is often the only one available for many would-be clients.

1.1 A need for HTML wrappers

As soon as we want to go beyond the basic mode of a *browsing human*, for example to achieve Web-awareness among services (services taking advantage of one another) or interoperability (between Web sources and legacy databases or among Web sources themselves), we need software applications and these need to access HTML data. HTML was really designed to display information to a human user, so applications need **HTML wrappers** that can make the content of HTML pages directly available to them.

The purpose of the World-Wide Web Wrapper Factory (W4F) toolkit that we propose to demonstrate here is the rapid design, generation, and integration in applications of such wrappers. Specifically, W4F's key features are: fully declarative specifications, light-weight components, rapid development, robustness, direct integration into Java programs and re-usability.

This work is presented at an interesting time because of the on-going emergence of XML as a more application-friendly standard for Web pages. Some people have already argued that there is no more need for HTML wrappers because data sources will soon serve XML documents. In fact, there already are countless HTML pages on the Web and the information that many of them contain will have to be displayed in XML in a relatively near future. This demonstration will show that our W4F toolkit, among other things, is an efficient instrument for facilitating the migration from HTML to XML.

More generally, we believe light-weight HTML wrappers are currently indispensable for Web inter-operation and Web information integration. In particular, such wrappers turn out to be also an excellent testbed for the construction of smarter customized applications for e-commerce, digital libraries, etc.

1.2 Outline of the demonstration

To keep our demonstration focused, we will show how to build a wrapper for the the Internet Movie Database (IMDb) and create a *gateway* to serve these HTML pages as XML documents. First we will explain how to specify what information to extract from HTML pages using our wysiwyg extraction wizard and our extraction language. Then we will display and test the wrapper using our visual interface. The next step will be to define an XML mapping for the extracted information. Finally, we will deploy the wrapper as a standard CGI-based Web interface that serves transparently and on-the-fly HTML pages as XML documents with their corresponding DTD.

However, we hope that the demonstration will argue for the effectiveness of the toolkit on any Web data source and for a broader range of applications.

2 The W4F toolkit in a nutshell

W4F (World-Wide Web Wrapper Factory) is a toolkit to generate Web wrappers. Our wrappers consist of three independent layers: retrieval, extraction and mapping. The **retrieval layer** is in charge of fetching the HTML content from a Web data source. From the user's point of view it means to provide the location of the document. The **extraction layer** extracts the information from the document. It is important here to note that we can extract complex structures and not just atomic elements from the page. The user provides extraction rules to identify relevant pieces of information. The **mapping layer**'s role is to specify how to export the data.

2.1 Architecture

This architecture is motivated both by the particularities of Web data sources and by the desire to take advantage of re-usable functionalities. For example, wrappers for Web sources that use the same query form or that feed into the same application could re-use the same components. The key point here is that each layer remains simple.

Now, for a given Web source, some extraction rules and some structural mappings, the toolkit generates a Java class that can be used as a stand-alone program or directly integrated into a more complex application.

A wrapper processes a Web source in the following way, as presented in Figure 1. An HTML document is first retrieved from the Web according to one or more retrieval rules. Once retrieved, the document is fed to an HTML parser that constructs a corresponding parse tree. Given the permissiveness of HTML, the parser has to recover from badly-formed documents using various heuristics.
Extraction rules are then applied on the parse tree and the extracted information is stored in our internal format.
Finally, information is mapped to structures exported by the wrapper to the upper-level application, according to mapping rules.

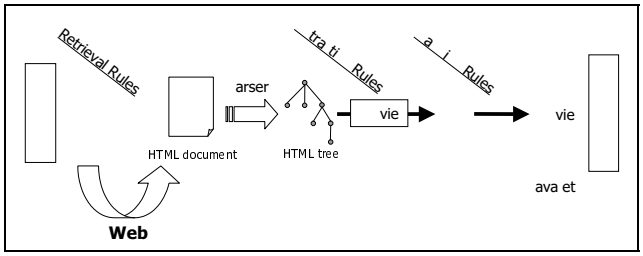

Figure 1: W4F information flow.

For the construction of the wrapper per-se, the toolkit provides a compiler that generates Java classes from a specification file, as well as some wizard applications to assist in the design of the wrapper layers.

2.2 HEL: HTML Extraction Language

A key feature of the toolkit is the HTML Extraction Language (HEL[11]) that permits a declarative specification of information extraction. HEL is a DOM-centric [12] language where an HTML document is represented as a labeled graph. It comes with two ways to navigate the tree. The first one follows the hierarchy of the document, implied by tags. This navigation is extremely useful in table-based documents for instance. The second one follows the flow of the document, i.e. a depth-first traversal of the document tree which corresponds to the way a document is read by a human. Using both navigation styles, most structures can be easily identified as extraction paths. To the best of our knowledge, HEL is the only language that captures both *shapes* of a document.

The language also offers conditions and extraction features based on regular expressions à la Perl (like `match` and `split`). The first allow the definition of robust extraction rules where conditions are resolved at run-time, on a per document basis. The second permit to capture a finer granularity inside the document.

Moreover, the language has been designed to extract complex structures from HTML documents and not just isolated pieces. For instance, the language can extract *as a whole* a movie with its title, genre and cast (see Figure 3, using the "`#`" operator, a record constructor in a sense). The implicit structure of the document does not have to be reconstructed from scratch but is extracted as is.

2.3 NSL, our internal data-model

Another interesting aspect is that the extracted information is stored using an anonymous and language neutral representation called NSL *nested string lists* (NSL), the data-type defined by $NSL = null + string + listof(NSL)$. NSL can represent complex structures (unlimited level of nesting) and then be mapped easily to the desired data structure. The user can take advantage of an automatic mapping to Java base types (`String, int, float`, etc. and their array extensions); he can also provide his own Java classes by writing a valid constructor that consumes the NSL; finally he can specify XML mappings using the XML declarative specification. Some new mapping extensions are to be included in future releases of the toolkit.

3 The demonstration

The demonstration will go through the various steps of the building and deployment of a wrapper. As *advertised* in the introduction, we want to "wrap" the Internet Movie Database. IMDb is the biggest information repository about movies and is freely available. Its underlying information system is a big file system[1] that serves HTML pages.

3.1 Building a wrapper, step by step

Building a wrapper consists of the following steps:

1. define each layer (retrieval, extraction, mapping)
2. test
3. refine
4. compile the wrapper into a Java class
5. include the Java class as part of an application

The toolkit provides wizards to help the user write, test and refine the definition of the wrapper.

3.2 Defining extraction rules using the extraction wizard

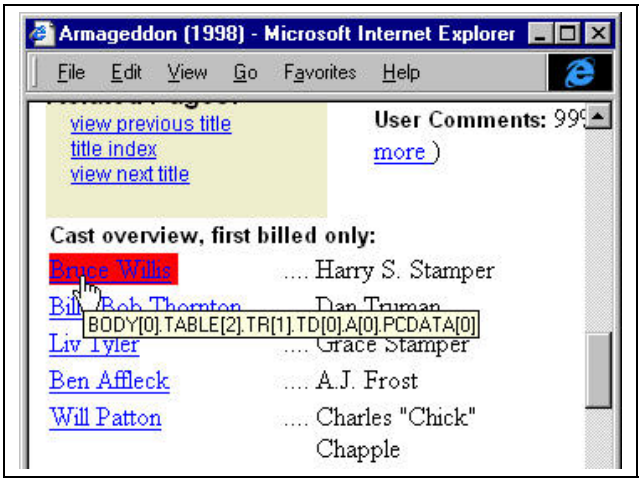

Figure 2: the extraction wizard.

The most critical part of the design of the wrapper is the definition of extraction rules. The role of the extraction wizard (see Figure 2) is to help the user write such rules. Instead of forcing the user to mess up with the HTML code, W4F adopts an annotation approach.

[1] See http://www.imdb.com/interfaces#plain for more details.

For a given HTML document, the wizard feeds it into W4F and returns the document to the user with some invisible annotations (the document appears exactly as the original).

On Figure 2, when the user points to "Bruce Willis", the corresponding text element gets high-lighted and the canonical[2] extraction rules pops-up. Even if the wizard is not capable of providing the best extraction rule, it is always a good start. Compare what is returned by the wizard and what we actually use in our wrapper (figures 2 and 4).

3.3 Testing

Figure 3 presents the wizard (a Java applet here) that assists the user when writing and testing the wrapper. The applet layout represents the 3-layer architecture of the wrapper.
In the top layer, the user will input the location of the Web source and the retrieval method (here a `GET`).

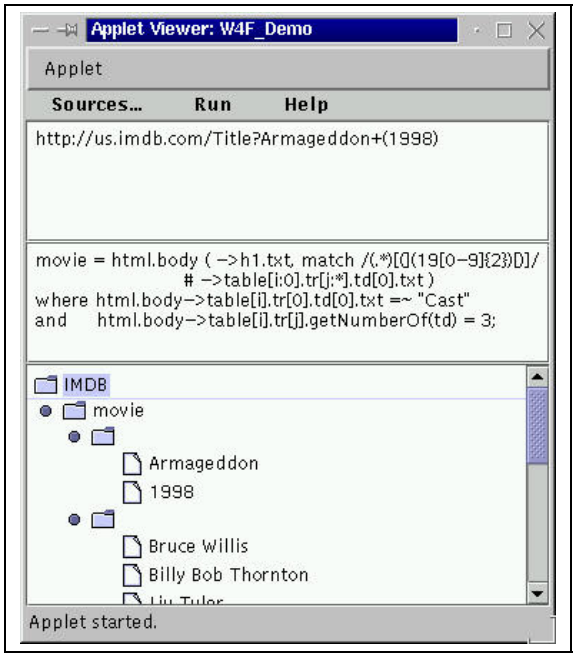

Figure 3: a visual view of a wrapper with its 3 layers.

The middle layer displays the extraction rule – expressed in the HEL language – to be applied on the retrieved HTML page. In this example, the rule will extract the title, the year and the cast of the movie (list of actors).
The bottom layer represents the structure of the information extracted as an NSL data-structure. This is the default mapping exported by the toolkit.

3.4 Refining the wrapper with a mapping to XML

The NSL structure extracted from the HTML pages can be used as is – the toolkit provides printing methods – or can be mapped to other data structures. For our target application, we can take advantage of a declarative specification for an XML mapping (see figure 4). This XML mapping is very close to the corresponding extraction rule (it can be seen as an annotation of the extraction or a template). Strings indicate tag or attributes names; dots indicate tag nesting;

[2]By canonical we mean that it uses only hierarchy based navigation.

```
RETRIEVAL_RULES ::
  getMovie(String title)
   GET "http://us.imdb.com/M/title-substring/title=$title$";

EXTRACTION_RULES ::
movie = html.body(
  ->h1.txt, match/(.*?) [(]/           // title
# ->h1.txt, match/.*?[(]([0-9]+)[)]/   // year
# ->td[i:0].a[*].txt                   // list of genre
# ->table[ii:0].tr[jj:*].td[0].txt, match/(\S+)\s(.*)/
                                       // first, last name
                  )
where html.body->td[i].b[0].txt = "Genre"
and   html.body->table[ii].tr[0].td[0].txt =~ "Cast"
and   html.body->table[ii].tr[jj].getNumberOf(td) = 3;
}

XML_MAPPING ::
movie_XML_template = Movie ( .TITLE^
                   # .YEAR^
                   # CATEGORIES*.Genre
                   # ACTORS*.Actor ( .FN # .LN )
                           );

JAVA_CODE ::
public static void main(String[] args)
{
 IMDB wrapper = IMDB.get(args[0]);
 print( wrapper.movie, wrapper.movie_XML_template );
}
```

Figure 4: `IMDB.w4f`, the wrapper description file.

"#" indicate concatenation; "*" indicates lists; "^" indicates that the value as to be used as an attribute of the parent tag.
The benefit of this approach is two-fold: (1) the semantics of the mapping from the extraction-rule to XML are straightforward; (2) the DTD of the corresponding XML document can be generated.

3.5 The final wrapper as a self-contained file

The wrapper itself is self described by a single file presented in Figure 4. For a full description of the extraction and mapping syntax, refer to [11]. Such files can later be copied, modified and re-used. Moreover, since each layer is independent, it can be developed separately and replaced later on.

The wrapper description file `IMDB.w4f` is then compiled into a Java class `IMDB.class` (less than 5 kb)[3] that can be used as a stand-alone application or integrated into another Java application.

3.6 Deploying the wrapper

For the XML-Gateway service, the only thing we have to do is to make the wrapper available as a CGI script. The result of the migration from HTML to XML is presented in Figure 5. The documents generated on the fly can then be sent to an XML query language like XML-QL [5].

4 Conclusion

Wrapper construction is a key issue in the implementation of mediator-based architectures [13]. And with more and more data sources being available on the Web, it is important to have a convenient framework to build, test and deploy

[3]The footprint of the entire W4F package is less than 200kb.

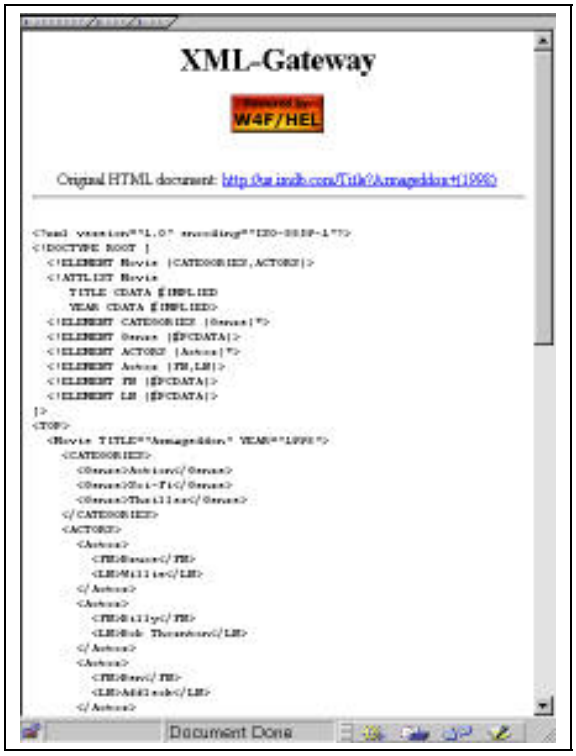

Figure 5: the XML document generated on-the-fly.

such wrappers in order to make the content of Web sources directly available to applications.

In this demonstration, by using the W4F toolkit we have managed to build quickly a wrapper for a large HTML-based Web data source and deploy it as a gateway that serves on-the-fly these pages as XML documents.

The main contributions of W4F are: (1) Wrappers are split into 3 separate layers. (2) The description of a wrapper is fully declarative. (3) Entire structures can be extracted from HTML pages and not just single "atomic" pieces. (4) The toolkit comes with visual wizards to help the user define extraction rules and test the wrapper before deployment. (5) Generated wrappers are ready to be integrated in any Java application.

Compared to other approaches [8, 9], we do not use a grammar-based approach for extraction but rely on the DOM object-model, which gives us for free some wysiwyg visual tools like [1].

With rich features like hierarchical and flow-based navigations, conditions and nested constructs, our extraction language is more expressive and robust than [6, 2]. Unlike [3], we do not try to query the Web but simply extract structure from Web information sources: querying is the concern of the application.

In W4F, we do not address problems that are specific to mediators but we believe that our wrappers can be easily included into existing integration systems like TSIMMIS [7], Kleisli [4], Garlic [10], etc.

Our goal with W4F is to offer something simple yet powerful enough to handle any kind of HTML structure. From our point of view, our Web wrappers are similar to small shell programs that perform simple tasks and can be combined at will. They can be easily written to automate tedious tasks, help interface huge legacy HTML data, and make information processing and inter-operability easier, faster and smarter.

The toolkit is freely available at http://db.cis.upenn.edu/W4F. On-line examples of W4F applications (including the wrapper presented here) can be found at the same location.

References

[1] Brad Adelberg. NoDoSE – A Tool for Semi-Automatically Extracting Semi-Structured Data from Text. In *Proc. of the SIGMOD Conference*, Seattle, June 1998.

[2] Charles Allen. WIDL: Application Integration with XML. *World Wide Web Journal*, 2(4), November 1997.

[3] Gustavo Arocena and Alberto Mendelzon. WebOQL: Restructuring Documents, Databases, and Webs. In *Proc. ICDE'98*, Orlando, February 1998.

[4] Susan Davidson, Christian Overton, Val Tannen, and Limsoon Wong. Biokleisli: A digital library for biomedical researchers. *Journal of Digital Libraries*, 1(1):36–53, November 1996.

[5] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. XML-QL: A Query Language for XML, 1998. http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/.

[6] Jean-Robert Gruser, Louiqa Raschid, M. E. Vidal, and L. Bright. Wrapper Generation for Web Accessible Data Sources. In *COOPIS*, 1998.

[7] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting Semistructured Information from the Web. In *Proceedings of the Workshop on Management of Semistructured Data. Tucson, Arizona*, May 1997.

[8] Gerald Huck, Peter Fankhauser, Karl Aberer, and Erich J. Neuhold. JEDI: Extracting and Synthesizing Information from the Web. In *COOPIS*, New-York, 1998.

[9] G. Mecca, P. Atzeni, P. Merialdo, A. Masci, and G. Sindoni. From Databases to Web-Bases: The ARANEUS Experience. Technical Report RT-DIA-34-1998, Universita Degli Studi Di Roma Tre, May 1998.

[10] Mary Tork Roth and Peter Schwartz. A Wrapper Architecture for Legacy Data Sources. Technical Report RJ10077, IBM Almaden Research Center, 1997.

[11] Arnaud Sahuguet and Fabien Azavant. W4F, 1998. http://db.cis.upenn.edu/W4F.

[12] W3C. The Document Object Model, 1998. http://www.w3.org/DOM.

[13] Geo Wiederhold. Intelligent integration of information. In *ACM Sigmod*, Washington, DC, USA, 1993.