University of Pennsylvania
ScholarlyCommons

Database Research Group (CIS) | Department of Computer & Information Science

January 1999

Taming Web Sources with "Minute-Made" Wrappers

Fabien Azavant
École Nationale Supérieure des Télécommunications

Arnaud Sahuguet
University of Pennsylvania

Follow this and additional works at: http://repository.upenn.edu/db_research

Azavant, Fabien and Sahuguet, Arnaud, "Taming Web Sources with "Minute-Made" Wrappers" (1999). *Database Research Group (CIS)*. 39.
http://repository.upenn.edu/db_research/39

Database Research Group.

This paper is posted at ScholarlyCommons. http://repository.upenn.edu/db_research/39
For more information, please contact libraryrepository@pobox.upenn.edu.

Taming Web Sources with "Minute-Made" Wrappers

Abstract
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 and this proportion keeps increasing. In some cases, database access is only granted through a Web gateway using forms as a query language and HTML as a display vehicle. In order to permit inter-operation (between Web sources and legacy databases or among Web sources themselves) there is a strong need for Web wrappers.

Web wrappers share some of the characteristics of standard database wrappers but usually the underlying data sources offer very limited query capabilities and the struc- ture of the result (due to HTML shortcomings) might be loose and unstable. To overcome these problems, we divide the architecture of our Web wrappers into three components: (1) fetching the document, (2) extracting the information from its HTML formatting, and (3) mapping the information into a structure that can be used by applications (such as mediators).

Comments
Database Research Group.

This working paper is available at ScholarlyCommons: http://repository.upenn.edu/db_research/39

Taming Web sources with "minute-made" wrappers

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 A need for Web wrappers

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 and this proportion keeps increasing. In some cases, database access is only granted through a Web gateway using forms as a query language and HTML as a display vehicle. In order to permit inter-operation (between Web sources and legacy databases or among Web sources themselves) there is a strong need for Web wrappers.

Web wrappers share some of the characteristics of standard database wrappers but usually the underlying data sources offer very limited query capabilities and the structure of the result (due to HTML shortcomings) might be loose and unstable. To overcome these problems, we divide the architecture of our Web wrappers into three components: (1) fetching the document, (2) extracting the information from its HTML formatting, and (3) mapping the information into a structure that can be used by applications (such as mediators).

W4F is a toolkit that allows the fast generation of Web wrappers. Given a Web source, some extraction rules and some structural mappings, the toolkit generates a Web wrapper (a Java class) that can be used as a stand-alone program or integrated into a more complex system.
W4F provides a rich language (HEL: HTML Extraction Language) to express declaratively extraction rules and mappings, as well as a wysiwyg interface that allows the creator of the wrapper to pick relevant pieces of information just by clicking on them, as he sees them in his Web browser.

As an illustration, we present the TV-Guide Agent that allows users to query TV movie listings by time scheduled (date, time, channel) and program content (movie genre, rating, year, cast, country, etc.). This example demonstrates real inter-operability between TV-listing information (http://tv.yahoo.com) and movie information (Internet Movie Database).

2 The architecture

The architecture of our wrapper "factory" identifies three separate components: retrieval, extraction and mapping. This structure 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 reuse the same components.

As presented in Figure 1, an HTML document is first retrieved from the Web according to one or more **retrieval rules**. Currently, a retrieval rule simply consists of the url[1] of the remote document.
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.
Extraction rules are then applied on the parse tree and the extracted information is stored in an internal format based on *nested string lists* (NSL), the datatype defined by $NSL = null + string + listof(NSL)$.
Finally, NSL structures are mapped to structures exported by the wrapper to the upper-level application, according to **mapping rules**.

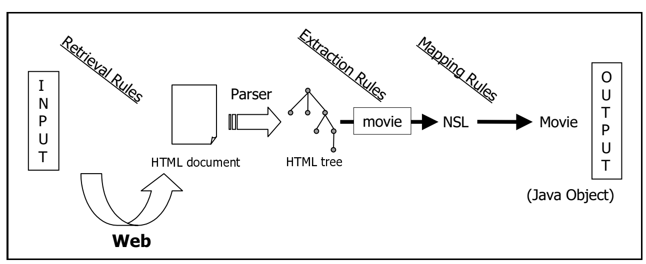

Figure 1: W4F information flow

This process is repeated for each Web document.

3 Extracting information

In this section, we glimpse at HEL, the language used in W4F for declaring extraction rules. Full details can be found in [12].
A declarative specification of the wrapper for the movie

[1] Both GET and POST methods are supported.

database is presented in Figure 2 and will be used in this section to illustrate some features of the language.

3.1 Building the HTML parse tree

Each Web document is parsed into a parse tree corresponding to its HTML hierarchy. The parse tree follows the *Document Object Model* [13].
A tree consists of a root[2], some internal nodes and some leaves. Each node corresponds to an HTML tag (text chunks correspond to PCDATA nodes). A leaf can be either a PCDATA or a *bachelor* tag[3]. Given this, it is important to note that there is a 1-to-1 mapping between a valid HTML document and its tree. Non-leaf nodes have children that can be accessed by their label (the label of the HTML tag) and their index (the order of appearance).

3.2 Two ways to navigate the tree

Navigation along the abstract tree is performed using path-expressions [4, 1].

The first way is to navigate along the **document hierarchy** with the "." operator.
The path 'html.head[0].title[0]' will lead to the node corresponding to the <TITLE> tag, inside the <HEAD> tag, inside the <BODY> tag. This type of navigation offers a "canonical" way to reach each information token.

The second way is to navigate along the **flow of the document**, with the "->" operator.
The path 'html->table[0]' will lead to the first <TABLE> tag found in the depth-first traversal of the abstract tree starting from the <HTML> tag. This operator increases considerably the expressivity of HEL, since it permits to cope with irregularities of structure. It is also useful to create navigation shortcuts.

Both operators apply to an internal node of the tree and return one (or more) child according to a label name (e.g. html, title, etc.) and an index value. Index ranges can also be used to return array of nodes, like [1,2,3], [7-] or the wild-card [*]. When there is no ambiguity, the index value can be omitted and is assumed to be zero.

3.3 Extracting node information

Extraction rules do not operate on the nodes themselves but on the information they carry.
From a tree node, we can extract its text value ".txt". The text content of a leaf is empty for a bachelor tag and corresponds to the chunk of text for PCDATA. For internal nodes, the text value corresponds to the recursive concatenation of the sub-nodes, in a depth-first traversal.
The underlying HTML source is extracted using ".src".
Some properties of the node like the value of some attributes as well as the number of children can be retrieved using "getAttr" and "numberOf".

The detail of node information is presented in Table 1.

3.4 Using regular expressions

The relevant information might not be entirely captured by the HTML structure (e.g. an enumeration inside a table cell): that's where regular expression patterns can be useful.

[2] The root is labeled html.

[3] A bachelor tag is a tag that does not require a closing tag, like <IMG> or
.

	Root	Int. nodes	Bachelor tags	PCDATA
.txt	√	√	N/A	√
.src	√	√	√	√
.getAttr	N/A	√	√	N/A
.numberOf	√	√	N/A	N/A

Table 1: Tree nodes and their properties

```
SCHEMA ::

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

EXTRACTION_RULES ::
movie = html.body (
  ->h1.txt, match /(.*)[(](19[0-9]{2})[)]/  /* title, year */
# ->table[i:0].tr[j:*].td[0].txt            /* cast        */
WHERE   html.body->table[i].tr[0].td[0].txt =~ "cast"
AND     html.body->table[i].tr[j].getNumberOf(td) = 3;
```

Figure 2: A W4F wrapper for the Internet Movie Database

HEL provides two operators match and split that follow the Perl syntax (see [14]).
The match operator takes a string and a pattern, and returns the result of the matching. Depending on the nature of the pattern[4], the result can be a string or a list of strings.
The split operator takes a string and a separator as inputs and returns a list of substrings.

These operators can be used in cascade: the operator is applied to each element of the previous result.

3.5 Enforcing constraints

As mentioned in 3.2, array elements can be specified using wild-cards or index values. They can also be defined using variables to which conditions can be attached by introducing a WHERE clause. Conditions cannot involve nodes themselves but only their properties. Various comparison operators are offered by the language. Constraints are another feature of HEL that gives a lot of freedom to the user when he writes wrappers; it also permits to deal with irregularities in the structure of the document.

3.6 Creating nested structures

The language also provides the fork operator "#" to construct NSLs by following multiple sub-paths at the same time. This is particularly useful when information spread across the page need to be put together. For a movie (see Figure 2), we put together the title and the cast.

4 Mapping information

The information obtained from the execution of the extraction rules is stored as a NSL. The structure of the NSL (levels of nesting, etc.) is fully defined by the rules themselves. The use of an index range or a split in a rule will result in one extra level of nesting for the result.

[4] The number of parenthesized sub-patterns indicates the number of items returned by the match. In the example (see Figure 2), the match will return a pair (title, year).

```
SCHEMA ::
Movie movie;

public class Movie
{
  String title; String [] cast; int year;
  public Pointer(NestedStringList nsl)
    {
      title = nsl[0][0];
      year  = (int) nsl[0][1];
      cast  = nsl[1];
    }
}
```

Figure 3: User-defined mapping for IMDB wrapper

The philosophy of W4F is to return anonymous NSLs that can be freely consumed by some upper-level applications. The default mapping consists of mapping NSL into Java base types (string, int, float) and their array extensions. W4F performs the conversion automatically. In Figure 2, there is no specified mapping and the default mapping is used.

The user can also define his own mappings. In the current implementation, this means to provide a Java class with a proper constructor. A possible mapping for the movie database wrapper is presented in Figure 3.

As of today, W4F offers a generic mapping to the K2 object model [6]; the XML mapping is almost achieved.

5 Wysiwyg interface

The real work in the creation of a Web wrapper is in the specification of the extraction rules. An expert could write them down by just looking at the HTML source. However, in order to increase accessibility, W4F offers an interface in which the user is presented with the document and he simply clicks on the pieces of information he wants to collect. The system then “magically” returns the corresponding extraction rule (as shown in Figure 4).

Figure 4: The Wysiwyg interface

The Wysiwyg interface takes a URL as an input, parses the corresponding HTML document and returns a new “annotated” HTML document. The following annotation is performed: each PCDATA leaf corresponding to an outer tag is transformed. Basically, a new tag `<SPAN>` is inserted within the outer tag. The benefit of it is that the chunk of text can now be given a specific behavior that can be used by the HTML browser.

Assuming that the path-expression corresponding to this tag in the HTML abstract tree is `"html.tag[n].txt"`, we have the following transformation:

```
<TAG> stuff </TAG>
```
becomes
```
<TAG><SPAN ID="html.tag[n].txt"> stuff </SPAN></TAG>
```

The new `<SPAN>` tag created now carries some information about the path that leads to this specific piece of information.

It is important to note that the path-expressions used for `ID` (i.e. returned by the interface) are *canonical* and only use the `"."` operator. This extraction rule might not be the most robust one [5], but it is a good start anyway.

6 Examples of applications

For the TV-Guide Agent service, the TV-listing information is first extracted from http://tv.yahoo.com. Then for each movie title, a query is sent to the Internet Movie Database to retrieve information about the movie (see figure 5). Because of title mismatches, we have first to get a list of matching titles before proceeding to the movie itself.

The service uses one wrapper for the TV-listings and two wrappers for the movie database (one for matching titles, one for the movie itself).

From a declarative description like the one presented in Figure 2, a Java class is generated and compiled in order to be directly used by the main application.

Even if a lot of inter-operation mismatches still have to be resolved by program code, the extraction part is now fully declarative.

The W4F toolkit has been successfully used to build various kinds of applications (data-warehousing of Web data sources, Web agents, etc.), extracting information very diverse and versatile sources (CIA World Factbook, Med-Line, on-line stores).

7 Conclusion and future work

Wrapper construction is a key issue in the implementation of mediator-based architectures [15]. Several approaches to wrapper generation use procedural descriptions (eg. [8] using *configurable extraction programs*) or grammars [9], but none of them “speaks” HTML and therefore they must rely on adhoc approaches to Web sources. Web-OQL [3] maps HTML documents generically to an object-oriented data model and then uses OQL to extract information.
The extraction per-se often requires considerable expertise and maintenance is poorly supported. Strategies that use machine-learning techniques like [10] aim at solving this latter issue.

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 [8], Kleisli [7], Garlic [11], YAT [5], etc.

We try to make the most out of the implicit HTML hierarchy through the use of the Document Object Model (like in [2]) but we also provide regular expression operators to capture finer granularity. This design permits the

[5] As a matter of fact, the extraction rule presented in Figure 5 is a refinement of the one proposed by the interface in figure 4.

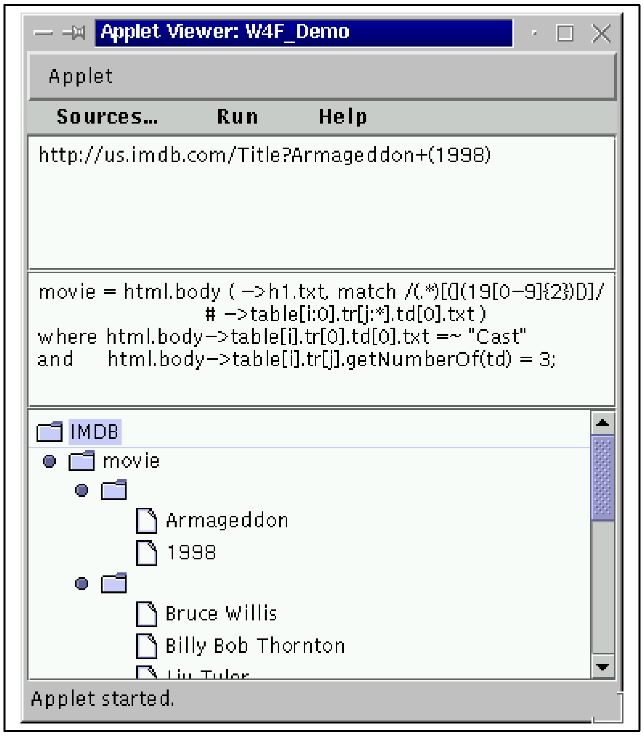

Figure 5: W4F in action, extracting movie information

semi-automatic generation of robust extraction rules that relate perspicuously to the navigation of the HTML parse tree. Moreover, it provides, essentially for free (!), a true wysiwyg interface that returns a default path for any piece of information identified by the user. In order to offer reusability, the result of the extraction is stored in an anonymous structure that can be mapped into various user-defined structures, some of them already built into the system.
We think that the splitting of Web wrappers into independent components (layers) and the use of declarative specifications will make it easier to design such wrappers in order to make Web sources inter-operate with one another.

Our future work will focus on a default mapping to a data-model for querying, on an improvement of the user interface to deal with more complicated retrieval methods, and on the notion of Web services in terms of interface, caching and object identity.

The W4F toolkit has been developed under JDK-1.1.5, using JavaCC[6] to generate the the HTML parser and the parser for the HEL language; regular expressions are evaluated using PAT[7]. The footprint of the toolkit is less that 200kb.

On-line examples of W4F applications (including the TV-Guide Agent presented here) can be found at the *Penn Database Research Group* web site[8].

Acknowledgements:
We would like to thank Jérôme Siméon and Val Tannen for some early comments on a draft of this paper.

[6] http://www.suntest.com/JavaCC
[7] PAT, the regular expression package: http://www.javaregex.com
[8] http://db.cis.upenn.edu.

References

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L. Wiener. The lorel query language for semistructured data. *Journal on Digital Libraries*, 1997.

[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] Vassilis Christophides. *Documents structurés et bases de données objet*. PhD dissertation, Conservatoire National des Arts et Metiers, October 1996.

[5] Sophie Cluet, Claude Delobel, Jérôme Siméon, and Katarzyna Smaga. Your Mediators Need Data Conversion! In *Proc. SIGMOD Conference*, Seattle, 1998.

[6] Johnatan Crabtree, Scott Harker, and Val Tannen. An OQL interface to the K2 system. Technical report, University of Pennsylvania, Department of Computer and Information Science, 1998. To appear.

[7] 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.

[8] 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.

[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] Naveen Ashish and Craig A. Knoblock. Semi-automatic Wrapper Generation for Internet Information Sources. In *Proc. Second IFCIS Conference on Cooperative Information Systems (CoopIS)*, Charleston, South Carolina, 1997.

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

[12] Arnaud Sahuguet and Fabien Azavant. W4F: the WysiWyg Web Wrapper Factory. Technical report, University of Pennsylvania, Department of Computer and Information Science, 1998. To appear.

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

[14] Larry Wall, Tom Christiansen, and Randal L. Schwartz. *Programming Perl*. O'Reilly & Associates, 1996.

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