W4F: a WysiWyg Web Wrapper Factory for Minute-Made Wrappers

!!! Draft: Do Not Distribute !!!

Fabien Azavant
Télécom Paris (E.N.S.T.), France
azavant@poly.polytechnique.fr

Arnaud Sahuguet
Computer & Information Science Department
University of Pennsylvania
sahuguet@cheops.cis.upenn.edu

August 11, 1998

1 Introduction

The Web environment has become a de-facto standard to publish information. Content providers can take advantage of the wide deployment of browsers on computer systems; end users can benefit from the user-friendly interface and navigation capabilities.

It allows at the same time individual, companies, independent and governmental organizations to publish information (for research, fun, profit, etc.) at a very low cost.
Individuals create Web sites dedicated to their hobbies. Companies put on-line annual reports, catalogues, marketing brochures, product specifications. Government agencies publish new regulations, tax forms, etc. Independent organizations make available latest research results (e.g. *the Human Genome Project*).
As of today, for some specific domains, the "reference" information can *only* be found on the Web and this is even truer for real-time data such as stock-market (e.g. The New-York Stock Exchange or Nasdaq), weather forecasts, etc.

These information sources however live isolated, with no real connections with one another. Each service exists independently. There is no *Web-awareness*. Only human-beings seem to establish connections between services (say a travel-agency and a weather forecast institute). The next challenge of the Web is interoperability.

In many cases (most likely professional information sources), information comes from *hidden* underlying legacy databases and is published on the Web. However, the structure of the database has been lost because HTML is ill-suited for representing complex data (nested structures, types, inheritance, etc.): this seems to be the price to pay!

On the one hand, the problem has been partially solved for human-beings. Navigation along hyper-links and forms help them to access pieces of relevant information and it is easy for them to infer the structure from the context (TO BE DETAILED). On the other hand, it is not that easy for machines. The up-coming XML seems to be one solution.

In order to offer interoperability, one requirement now seems to be able to extract information from HTML pages and recover the underlying structure. Each Web source has to be somehow *reverse-engineered*. From a database point of view, it really means *wrapping* the Web source to offer an abstract view of it that offers a standardized access, structure, etc.

W4F is a toolkit that permits to design wrappers for Web sources. The resource is described in terms of how to access the resource, what pieces information to extract and what target structure to use.
The toolkit generates some Java code that is in charge of issuing the HTTP request, performing the extraction and returning the structured result. W4F is written completely in Java and can also be used to build Web APIs (WAPI) that can be called from other Java programs.

The rest of the paper is organized as follows. Section 2 presents the concepts related to wrappers and the architecture of the system. The principles of W4F are exposed in Section 3 with the abstract representation of HTML documents. The extraction layer with the HEL language are detailed in Section 4. Section 5 and Section 6 present the mapping and the retrieving layers. The architecture of the W4F system is exposed in Section 7. Section 8 provides some real-life case studies to illustrate the use of W4F. Some related work is briefly presented in Section 9. Section 10 contains our conclusions.

In the rest of the paper, we assume the reader familiar with the HTML language (see [MK98]), regular expressions using Perl syntax (see [WCS96]).

2 Everything You Always Wanted to Know About Wrappers

2.1 What is a wrapper

A *wrapper* (aka *translator*) is a software component that converts data and queries from one model to another. In the case of Web sources, the role of the wrapper is to convert information implicitly stored as an HTML document (or a set of documents) which consists of plain text with some tags, into information explicitly stored as part of a data-structure. HTML documents do not contain any type information.

In the case of the Web, the wrapper also has to deal with the retrieving of information, via the HTTP protocol (through a GET or a POST method). The knowledge of the capabilities of source (specially if it consists of an underlying legacy database) can increase tremendously the performance of the system.

2.2 How the job is done today

Most people use ad-hoc *wrappers* to deal with Web information sources.

The wrapper is usually a C-program, a Perl-script (or whatever your favorite language is) that fetches the information from the Web and processes it. The description of the extraction is hidden inside the code as well as the target structure (if any). Therefore, writing a new wrapper requires a lot of skills and it is not obvious to derive and maintain new wrappers.

As already mentioned, HTML has not been designed to carry structure and is mostly used for presentation/display purposed. Web publishers tend to frequently modify page lay-out, while

they wish to represent the same structure. Such wrappers do not distinguish extraction and data conversion.

3 W4F manifesto

We think the design of Web wrappers should follow some guidelines.

3.1 Extraction strategies

The content retrieved from the Web is an HTML document from which structure has to be "inferred". Extraction should follow a *multi-granularity* approach. A Web document contains various levels of structure.

The first level is the level of HTML tags. The *head* of the document is separated from its *body*; the document is split into paragraphs, tables, bullet list, etc. These explicit separations are often used by content providers to identify some specific information. HTML tagging divides the document into spans and each span can be uniquely identified along the HTML hierarchy. The extraction should be able to operate on the content enclosed by HTML tags as well as the properties of the tags themselves.

The second level concerns the content of these spans. Inside a span, some patterns can represent structure. For instance, a sequence of words separated by commas would represent an enumeration. This kind of structure can often be expressed in terms of regular expressions.

The last level concerns tokens. A token can represent a int, a string, a phone number, etc. This structure can often be represented as regular expression patterns.

3.2 Design Guidelines

The Web wrapper itself should be split into three separate layers: (1) retrieval, (2) extraction, (3) mapping.

The **retrieval layer** deals with accessing the source through a GET or a POST method. This layer is in charge of building the correct URL to access a given service and to pass the correct parameters. It should also handle redirection and failures, authorization, etc. The retrieval layer might require some input from the user or none.

The **extraction layer** deals with extraction per-se. Extraction should take advantage of the HTML grammar as well as of regular expressions patterns. The extraction language should be expressive enough to capture the structure expressed by the document.
The extraction layer is really specific to the source. At this level, every extracted information should be regarded as a *string*: the coercion shall be done by the mapping layer.
The extraction layer is described by a set of extraction rules.

The **mapping layer** is in charge of transforming the extracted information into the target structure that has to be used outside of the wrapper according to the user's needs. It should take advantage of generic conversion tools that can directly map extracted string into say dates, SS or phone numbers.

We think this separation is useful because the 3 layers are independent and can be reused independently. For instance the same retrieval layer can be used for various Web sources that use the same input FORM. Or for a given Web source, one might want to have different target structures (say a relational view and an object-oriented view). Some cases might require a retrieval layer to use another Web wrapper.

3.3 A Declarative Language

The description of these different layers should be declarative in order to make the language independent, understandable/maintainable, reusable and exchangeable. One advantage of a declarative language is that it ignores the underlying implementation. In the case of W4F, the execution is done through some Java code, but it could be done through any programming language.
Having an understandable/declarative language makes it easier for users to maintain the Web wrapper in case the HTML layout of the content has been changed (say the HTML now uses Cascading Style Sheets).
Another major advantage is the capability of imitation. Part of the success of HTML comes from the fact that you can learn it by looking at it and you can just go to a Web page and use the HTML source as is. It is important when designing wrappers because very often the same patterns occur.

3.4 Nested String List

Extraction is proper to one document and is somehow unique. In order to capture the structure expressed by the document, we use *nested string list*s (NSL) that are similar to LISP parenthesized expressions. A NSL can be formally defined as follows:

```
NSL  :=     null
          |  string
          |  list of (NSL)
```

As already mentioned above, from the extraction layer's point of view, everything is a *string* or a list of NSL. A mapping to a target structure represents one way to consume this structure.

Example 3.1 *The Yahoo directory*

As an illustration, the classification of Yahoo![1] web site corresponds to the following NSL.

```
[ [ "Arts & Humanities", [ "Literature", "Photography" ] ],
  [ "Business & Economy", [ "Companies", "Finance", "Jobs" ] ], ...  ]
```

This structure is intrinsic to the document and there is no other way to extract it.
However, it might not be suitable for the user's specific application. Some people would like to see the information as part of a relation, some other people as part of a collection of complex objects, etc. In any case, all of them should be able to use the NSL, whatever their representation is.

[1] http://www.yahoo.com

Following the previous example, the information could be mapped into a relation:

```
[ [ "Arts & Humanities", "Literature" ] [ "Arts & Humanities", "Photography" ] [ "Business & Economy",
"Companies" ] [ "Business & Economy","Finance" ] [ "Business & Economy", "Jobs" ], ...]
```

or into a OQL record structure:

```
set(
struct(name:  "Arts & Humanities", teams:  list( "Literature", "Photography")),
struct(name:  "Business & Economy", teams:  list( "Companies", "Finance", "Jobs")), ...
)
```

These are two different ways to consume the NSL structure.

4 W4F Extraction Language

This section describes the language used by W4F to define extraction rules, called HEL for *HTML Extraction Language*. An extraction rule is an assignment between a variable name and a path-expression. At run-time, the result of the assignment is a NSL.

4.1 The Object Model

For HEL, an HTML document is represented as a tree-structure based on the JavaScript (see [Fla98]) page object model. It represents a document according to its HTML tag hierarchy (see [MK98]): an inner tag is a child of its closest outer tag.

It is important to remark that there is a 1-to-1 mapping between a valid[2] HTML document and its tree-structure.

There are 3 kinds of nodes in a tree: (1) the **root**, (2) **internal nodes** and (3) **leaves**. All nodes carry a label that corresponds to the HTML tag they represent. Text chunks are labeled as PCDATA.
The root has no parent and its label is `html`.
Internal nodes may have children and attributes. They represent *closed* HTML tags. Children of an internal node are accessed by their label and their index, because a node can have many children of the same label.
Leaf nodes have no children: they represent either *bachelor* HTML tags[3] and may have attributes; or they represent PCDATA values with just some text value.

As we can see on Figure 1, this representation really describes the hierarchical structure of the HTML page: for instance, a row `<TR>` inside a table `<TABLE>` is represented as a child of the table. Another important point is that the original HTML document can be recovered via a *depth-first* path in the tree: the depth-first orientation represents the continuous flow of the document.

Given this HTML tree-structure, a path-expression is a very convenient way to identify a location (*an information token*) along this tree. Labels names used inside a path-expression are the same as the names of the node they represent.

[2] in the sense that the document is compliant with the grammar
[3] like `<IMG>` or `<BR>` that do not require any closing tag.

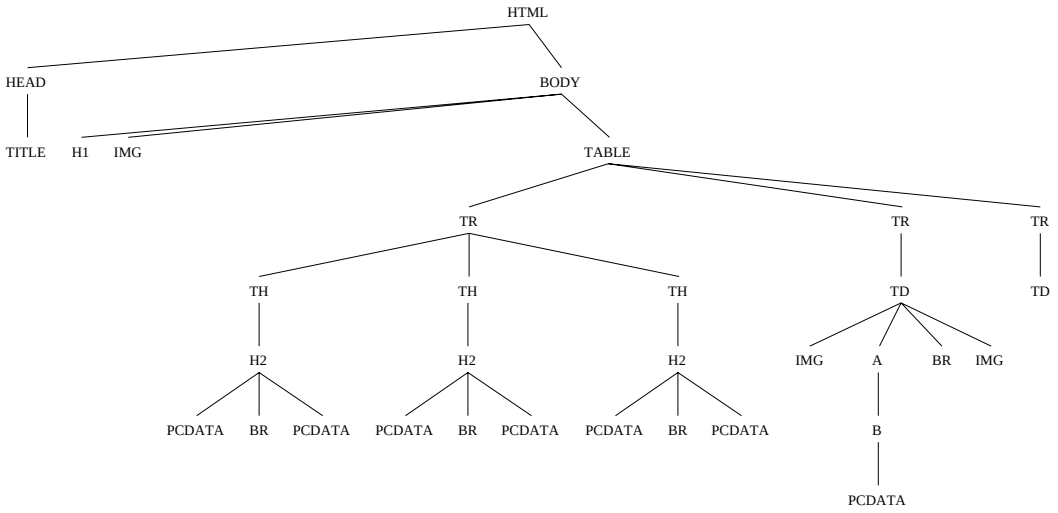


Figure 1: A portion of the HTML tree of http://www.cis.upenn.edu/~db

4.2 Navigating the HTML tree

One of the goal of W4F is to offer a rich language to identify location of information inside the page, but to remain simple to write and to evaluate. HEL tries to capture the many different ways a user would like to specify the location of the information he is interested in. It means that the language should offer more than one way to navigate through the tree structure.

4.2.1 Following the document hierarchy... (dot operator)

The first type of navigation is **hierarchy based**. From one node of the tree, it is possible to reach one of its children, given its label and its index. This type of navigation uses the *dot operator* (`'.'`). In the previous example (see Figure 1), from node `HTML` we can access node `HEAD` or the image node `IMG` inside the table by simply writing ⟨⟨ `html.head` ⟩⟩ or ⟨⟨ `html.body.table[0].tr[1].td[0].img` ⟩⟩.

The HEL language is *node based* and the square brackets identify the index of the child, when there are more than one (when there is no ambiguity, the index can be skipped and is assumed to be 0).

It is important to note that, when using only the dot operator, there is a *unique* path for each information token of the document.

4.2.2 Following the document flow... (arrow operator)

The other type of navigation respects the **flow of the document** and ignores/overrides the hierarchical structure. This type of navigation is very useful because many times the HTML hierarchical structure is purely artificial and misleading (the content of a section with a `<H1>` heading is not represented by the children of `<H1>` but by its siblings). As mentioned previously, the flow of the document is represented by a depth-first traversal of the tree structure.

Our language provides the *arrow operator* (`'->'`). As an illustration, the second picture of the

page (if any) can be reached through: ⟨⟨ `html->img[1]` ⟩⟩. The semantics is to traverse the tree depth-first from node `html` and get the second node with label `img`. It is clear that there are many "equivalent" ways to identify the same item: ⟨⟨ `html.body->img[1]` ⟩⟩, ⟨⟨ `html.head.title->img[1]` ⟩⟩. The last path is equivalent provided that the evaluation of ⟨⟨ `html.head.title` ⟩⟩ is successful.

The user has to be careful when using the arrow operator, because indices then become *relative*. ⟨⟨ `html.body->img[1]` ⟩⟩ and ⟨⟨ `html.body->img[0]->img[0]` ⟩⟩ reference the same item, since the second image is actually the first, when the traversal starts from the first one.

4.3 Node properties

The HEL language is concerned with extraction: an extraction rule cannot return a node itself but a value related to this node. We now present properties we can get out of a node. Table **??** summarizes them.

We can extract the **text content** (`'.txt'`) of a node.
The text content of the leaf is empty for a bachelor tag and corresponds to its text value for PCDATA. The text content of an internal node is defined as the recursive concatenation of its children text content according to a depth-first ordering. The text content is what is seen by the user on his browser.

From a node, we can also extract the **source content** (`'.src'`) which corresponds to the underlying HTML source. The source content corresponds to what is seen by the user when choosing "View Source" on his browser.

Attributes of a node (if any) can be extracted using the `'getAttr(<label>)'` method which takes a node label in input. ⟨⟨ `html.body->a[0].getAttr(href)` ⟩⟩ will return the url of the first anchor of the document.
Similarly, the `'numberOf(<label>)'` method returns the number of children for a given label. ⟨⟨ `html.body.numberOf(a)` ⟩⟩ will return the number of anchors inside the document that are children of node node `BODY`. `'numberOf(all)'` will return the number of all children of any label.

HEL offers some regular expressions capabilities (with operators `match` and `split`) to capture some extra structure related to node properties.

	Children	Attributes	Text value	`.txt`	`.src`	`getAttr(.)`	`numberOf(.)`
Root	√	N/A	N/A	√	√	N/A	√
Internal nodes	√	√	N/A	√	√	√	√
Bachelor tags	N/A	√	N/A	N/A	√	√	N/A
PCDATA	N/A	N/A	√	√	√	N/A	N/A

Table 1: HTML nodes and their properties

4.4 Extracting NSL structures

As we mentioned in a previous section, the result of an HEL extraction rule is a *nested string list*. The exact nature of the NSL depends on the rule itself and the NSL can have an arbitrary

dimension/depth/level of nesting. The dimension is influenced by (1) the use of wild cards (`*`) or variables as index values, (2) the use of pattern matching operators (`match` and `split`) and (3) the use of the fork operator.
These cases are presented below.

The simplest rule returns one single string value. ⟨⟨ `html.head.title.txt` ⟩⟩ will simply return the string that contains the title of the Web page.

When accessing children from internal nodes, HEL allows the use of *range specifiers* to define a subset of children (children must have the same label). The `'*'` symbol means in the index context `'<label>[*]'` the list of all the children of the given label. HEL also allows the use of patterns like `'<label>[1,2-5,9-]'` to denote the children of index 0, of index between 2 and 5 or greater or equal to 9.
⟨⟨ `html.body->a[*].getAttr(href)` ⟩⟩ will return the list of urls found inside `<A>` tags in the document. The arrow operator indicates to return the value of attribute `HREF` for all `<A>` tags inside the body of the document. The returned NSL is a list of strings (dimension 1).
⟨⟨ `html.body->ul[*].li[*].txt` ⟩⟩ will return a list where each element is the list of text values that correspond to the text content of list items. The returned NSL is a list of list of strings (dimension 2).

Pattern matching operators can also modify the dimension. They can only be used on text values (text or source content). The syntax is as follows: `'<text value>, <op> /<Regular Expression>/'`, where `op` is either `match` or `split`. These operators can be used in cascade: in this case, the operator is applied each element of the result. For instance a `match` following a `split` will be applied to each element of the result of the `split`.

The `split` operator takes a string and a separator as inputs and returns a list of substrings. It always increase the dimension of the result by 1.
⟨⟨ `html.body.ul[0].li[0].txt, split/,/` ⟩⟩ will return the NSL: `[ "red", "orange", "blue", "pink", "purple", "yellow" ]`.

The `match` operator takes a string and one pattern/regular expression 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.
Expression ⟨⟨ `html.body.ul[0].li[1,2].txt, match/([A-Zaz]+) born ([0-9]+-[0-9]+-[0-9]+)/` ⟩⟩ will return the NSL: `[ ["Arnaud", "14-3-72"], ["Fabien", "4-6-75"] ]`.

Example 4.1 *Using* `match` *and* `split` *operators*

```
<HTML>
<BODY>
<UL>
<LI>red, orange, blue, pink, purple, yellow</LI>
<LI>Arnaud born 14-3-72
<LI>Fabien born 4-6-75
</UL>
```

The fork operator (`'#'`) permits to follow a path and then fork into various sub-paths. Sub-paths are separated by the `'#'` symbol and can themselves be forked. Results will be grouped into a list.

[4] The number of parenthesized sub-patterns indicates the number of items returned by the match.

Should we want to return the list of the hyper-links inside a page, the best way so far seems to extract 2 lists (⟨⟨ `html.body->a[*].txt` ⟩⟩ for link names and ⟨⟨ `html.body->a[*].getAttr(href)` ⟩⟩ for link urls) and join them later. But from the structure of the page, there should be only one list, where each element is a pair (name, url).
⟨⟨ `html.body->a[*]( .txt # .getAttr(href) )` ⟩⟩ will return a list (because of the `a[*]`) of elements, where each element is a pair (a list of 2 elements) where the the first one is the text content of the link and the second one is the corresponding url.
The rule indicates to explore the HTML tree for `<A>` tags. Then the exploration is split into two sub-exploration. One returns the text content, while the other returns the value of attribute node `href`. The result is grouped into a list (hence the increase of dimension).

The fork operator can be used more than once inside a path expression and forked paths can be of any complexity. The fork operator is different from the other operators because it permits to build irregular structures where elements of a list can have different value dimension/depth. It is up to the mapping layer to know how to deal with this irregular nested structures.

4.5 Enforcing constraints

As mentioned above, array elements can be specified using wild-cards or the index values. They can also be defined using variables and conditions can be attached to these variables. Now extraction rules are composed of two parts: the path itself and a set of conditions. Conditions are introduced by the `where` keyword and separated by `and` if needed. Conditions are in conjunctive form only. Conditions do not operate on node themsleves but on node propeties.
It is worth noting that the structure of the result (dimensions, etc.) is fully defined by the path itself and is not influenced by the set of conditions.

First let us look at a motivating example from the French white pages (http://www.pageszoom.com). This Web service returns information about people in a given *département* and we want to extract the people grouped by county. The table below represents the rough structure of the HTML document returned. Names is upper-case and italics represent county names (followed by the zip code). Other entries correspond to people with their full name followed by their phone number and address.
On the original page, each line is actually an HTML table itself and the first column is empty and is just used as a margin.

We now look at how we can use HEL to solve the problem.
⟨⟨ `html.body.table[*].tr[0].td[1].txt` ⟩⟩ will return the list of counties mixed with the names of people. If we want to get only the name of counties, we can ask for tables where the first element is in italics. If we want the names of people only, we can ask for table where the first element is not in italics. The corresponding expressions are:

⟨⟨ `html.body.table[i:*].tr[0].td[1].txt WHERE html.body.table[i].tr[0].td[1].b[0].numberOf(I) != 0` ⟩⟩
⟨⟨ `html.body.table[i:*].tr[0].td[1].txt WHERE html.body.table[i].tr[0].td[1].b[0].numberOf(I) == 0` ⟩⟩

The syntax for the use of variables is as follows.
In the path, a variable (say `i`) appears followed by a range specifier: `[i:*]` means all index values

ESPALION(12500)		
Sahuguet Sylvie	05 65 44 79 11	21 av St Côme
GABRIAC (12340)		
Sahuguet Jean	05 65 44 90 33	Le Bourg
Sahuguet Paul	05 65 48 52 33	rte Espalion
Sahuguet Pierre-Marie	05 65 48 51 62	lot Causse
MURET LE CHATEAU(12330)		
Sahuguet Henri	05 65 46 94 31	Le Bourg
RODEZ(12000)		
Sahuguet Nadine	05 65 68 59 34	49 r Grandet
Sahuguet Paul	05 65 42 06 17	18 r Chapelle
Sahuguet Rémi	05 65 42 17 57	La Parisienne 6 av Paris
SAINT GENIEZ D'OLT (12130)		
Sahuguet Philippe	05 65 47 41 73	Champ de la Place 253 av Espalion
SALLES LA SOURCE (12330)		
Sahuguet Bernard	05 65 74 97 75	Solsac

Figure 2: Screen shot from the French White Pages (http://www.pageszoom.com)

that satisfy the condition, `[i:0]` means the first one, `[i:1,3-5]` means the 2^{nd} and the 4^{th}, 5^{th} and 6^{th}. In the condition, the range specifier does not need to be repeated. Conditions cannot involve nodes themselves but values such as text content, source content, getAttr() or numberOf(). Conditions can deal with usual arithmetic comparison operators ("`==`", "`!=`") as well as some regular expression pattern matching for strings ("`=~`", "`!~`").
More than one variable can appear in an expression and conditions are grouped using the `and` keyword.

But let us go back to our example. In order to solve the problem we would like to identify the counties and then look for the following tables that are not county names. The problem is that using the arrow operator we cannot stop after the last person of a given county: the search will go on for all the persons that can be found in the document. We would like to have a way to stop the search when the condition fails for the first time (like a cut in Prolog).

```
html.body.table[i:*](    .tr[0].td[1].txt
                      # ->table[j:*].tr[0](    .td[1].txt
                                            # .td[2].txt
                                            # .td[3].txt )  )
WHERE html.body.table[i].tr[0].td[1].b[0].numberOf(i) != 0              // condition (1)
  AND html.body.table[i]->table[j].tr[0].td[1].b[0].numberOf(i) = 0, !; // condition (2)
```

This expression should be understood as follows. The result returned is a list (because of `table[i:*]`) of pairs (the pair comes from the first fork). Elements of the list must satisfy condition (1) on `i` which says that the table should represent a county. From our example, it means that index `i` ranges in {0,2,6,8,12,14}. Now, for each value of `i`, we look for the next tables (`->table[j:*]`) such that it represents a person and not a county according to condition (2). Condition (2) ends with the cut symbol "`!`" which means that we look for the next tables until we fail. Then, for each table satisfying condition (2), we extract the text content of the three columns.
The result of the extraction is:

```
[ ["Espalion(12500)", [ [ "Sahuguet Sylvie",       "05 65 44 79 11", "21 av St Come"] ],
  ["Gabriac(12340)",  [ [ "Sahuguet Jean",         "05 65 44 90 33", "Le Bourg"     ],
                        [ "Sahuguet Paul",         "05 65 48 52 33", "rte Espalion" ],
                        [ "Sahuguet Pierre-Marie", "05 65 48 51 62", "lot Causse"   ] ], ...
]
```

4.6 Handling failure

We now explain the semantics of rule evaluation as far as failure is concerned.

5 Mapping the extracted information

The philosophy of W4F is to separate the wrapper into 3 distinct layers. The extraction layer tries to capture as much structure as it can from the HTML document. The result is a NSL object that may contain very irregular nested structures (because of the use of the fork operator mainly).

The role of the mapping layer is to explain how to consume the NSL structure and return another one, more suitable for further use.
A mapping is defined *for only one extraction rule*. There is no way to express a structure out of more: it is beyond the scope of the mapping layer.
W4F offers some default mapping but also allows the user to define his own.

5.1 Default mapping

W4F knows how to handle base types such as *string*, *int* and *float* as well as list structures involving these types. They are the only types known by W4F. Should the user want to use other ones, he would have to provide them (see Custom Mapping). From the previous examples, we could write:

```
String[][] pointers;                                    // mapping layer
pointers = html.body->a[*]( .txt # .getAttr(href) );    // extraction layer
```

W4F will automatically convert the NSL into a Java object of type `String[][]`. For non string types, it tries to perform a conversion using Java parsing routines. When no schema is enforced, it assumes everything as a string.

5.2 Custom mapping

Default mapping is very useful for simple cases, but many times the end user really want to build a complex structure out of the extraction. Mapping rules can reference Java classes (including array classes) as the target of extraction rules. The only requirement is that the class contains a constructor that knows how to build an instance out of a NSL.

Going back to the previous example, we would like to get back from W4F an array of `Pointer` objects, where a `Pointer` object encapsulates a name and the corresponding url.

```
Pointer[] pointers;                                     // mapping layer
pointers = html.body->a[*]( .txt # .getAttr(href) );    // extraction layer
```

The user has to provide the corresponding `Pointer` Java class with a constructor that knows how to consume the NSL in order to build a Java object. Here follows a possible piece of Java code.

```
public class Pointer
{
  String name, url;
  public Pointer(NestedStringList in)
    {
      nsl  = (NSL_List) in;                                    // the NSL is really a list
      name = ( (NSL_String) nsl.elementAt(0) ).toString();     // list item 1 is a string
      url  = ( (NLS_String) nsl.elementAt(1) ).toString();     // list item 2 is a string
    }
}
```

It is worth noting that W4F knows how to deal with arrays. The user just has to tell W4F how to build a class instance out of a NSL.

6 The Retrieving Layer

The retrieving layer is in charge of issuing an HTTP request to a remote server and fetching the corresponding HTML document. The request might be always the same (fixed URL) or might be different. In this case, the user has to provide input to the retrieval layer.
There are two types of HTTP requests (GET and POST) that differs on the way parameters are sent. The GET method appends the parameters at the end of the url, while the POST method attach the variables to the request itself. From the end-user's point of view they are identical. From the W4F developer they are different.

The retrieving layer is described by a `.rtv` file of *retrieval rules*. A retrieval rule looks like an interface definition. The name of the retrieval rule is followed by the list of parameters it takes (for the current version of W4F, only strings are allowed). Then, the type of the method (GET or POST) as well as the corresponding url. The url might contain some variables to be replaced by their string value.
For the POST method, parameters can be specified using the `PARAM` keyword. Parameters can be fixed or can correspond to string value of input variables.

Here is a portion of the `.rtv` file used for the French White Pages.

```
RETRIEVAL_RULES {
  getURL(String lName,
         String fName,
         String address,
         String locality,
         String department)
    {
    METHOD: POST ;
    URL: "http://www.pageszoom.com/wbpm_pages_blanches.cgi";
    PARAM:  "TYPE_ACTION"           =       "decode_input_image",
            "DEFAULT_ACTION"        =       "bf_inscriptions_req",
            "LANGAGE_TYPE"          =       "francais",
            "PAGE_TYPE"             =       "PAGES_BLANCHES",
```

```
            "FRM_NOM"                =       lName,
            "FRM_PRENOM"             =       fName,
            "FRM_ADRESSE"            =       address,
            "FRM_LOCALITE"           =       locality,
            "FRM_DEPARTEMENT"        =       department,
            "BF_INSCRIPTIONS_REQ.x" =       "58",
            "BF_INSCRIPTIONS_REQ.y" =       "16";
    }
}
```

An example of a retrieval rule for a GET method can be found in Section **??**

7 Putting it all together

the compiler
the run-time
the wysiwyg

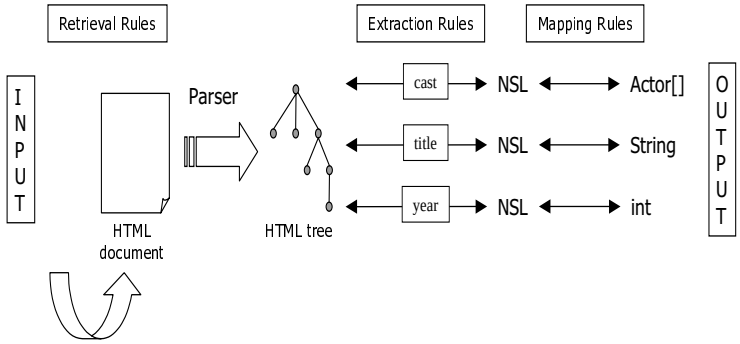


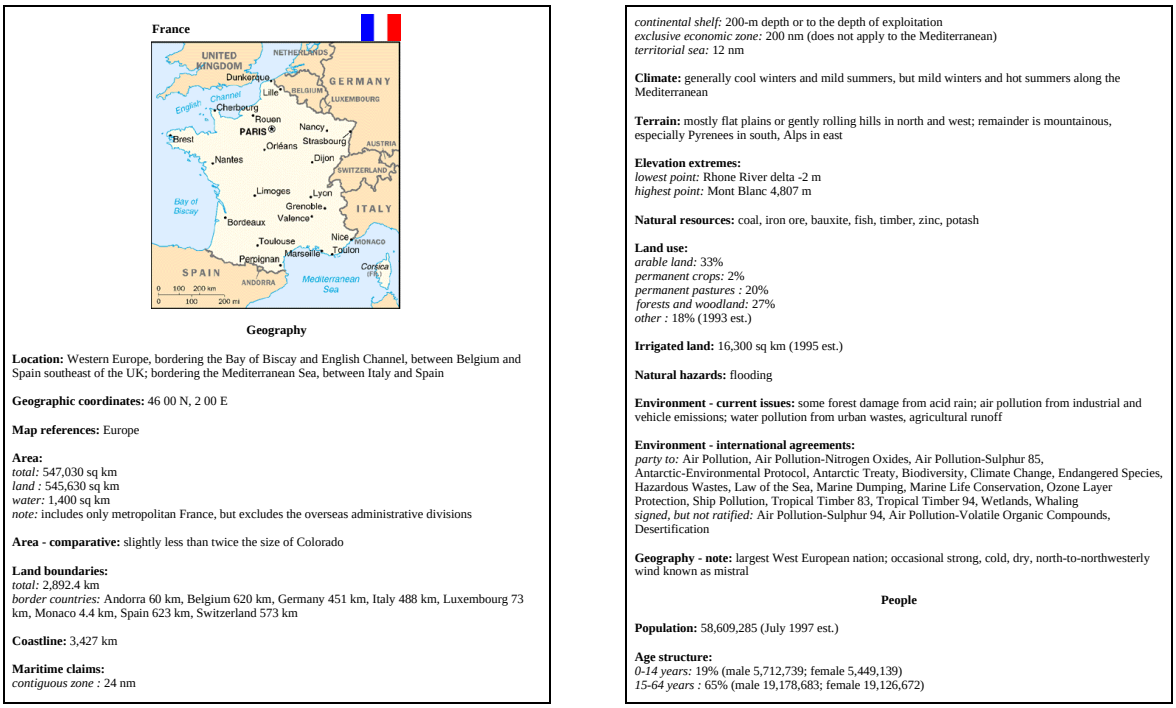
Figure 3: W4F architecture

8 Case studies

This section presents some case studies where W4F has been used successfully to solve some real problems. Examples have been chosen to demonstrate its simplicity, expressivity and ease of reuse. They also represent Web sources that offer some irregularities in the structure of the information they provide.

8.1 The CIA *World Factbook*

The *World Factbook* [5] gathers information about 266 countries. The Factbook contains one document listing all the country and then one profile per country. Country profiles share the same structure [6].



France

Geography

Location: Western Europe, bordering the Bay of Biscay and English Channel, between Belgium and Spain southeast of the UK; bordering the Mediterranean Sea, between Italy and Spain

Geographic coordinates: 46 00 N, 2 00 E

Map references: Europe

Area:
total: 547,030 sq km
land : 545,630 sq km
water: 1,400 sq km
note: includes only metropolitan France, but excludes the overseas administrative divisions

Area - comparative: slightly less than twice the size of Colorado

Land boundaries:
total: 2,892.4 km
border countries: Andorra 60 km, Belgium 620 km, Germany 451 km, Italy 488 km, Luxembourg 73 km, Monaco 4.4 km, Spain 623 km, Switzerland 573 km

Coastline: 3,427 km

Maritime claims:
contiguous zone : 24 nm
continental shelf: 200-m depth or to the depth of exploitation
exclusive economic zone: 200 nm (does not apply to the Mediterranean)
territorial sea: 12 nm

Climate: generally cool winters and mild summers, but mild winters and hot summers along the Mediterranean

Terrain: mostly flat plains or gently rolling hills in north and west; remainder is mountainous, especially Pyrenees in south, Alps in east

Elevation extremes:
lowest point: Rhone River delta -2 m
highest point: Mont Blanc 4,807 m

Natural resources: coal, iron ore, bauxite, fish, timber, zinc, potash

Land use:
arable land: 33%
permanent crops: 2%
permanent pastures : 20%
forests and woodland: 27%
other : 18% (1993 est.)

Irrigated land: 16,300 sq km (1995 est.)

Natural hazards: flooding

Environment - current issues: some forest damage from acid rain; air pollution from industrial and vehicle emissions; water pollution from urban wastes, agricultural runoff

Environment - international agreements:
party to: Air Pollution, Air Pollution-Nitrogen Oxides, Air Pollution-Sulphur 85, Antarctic-Environmental Protocol, Antarctic Treaty, Biodiversity, Climate Change, Endangered Species, Hazardous Wastes, Law of the Sea, Marine Dumping, Marine Life Conservation, Ozone Layer Protection, Ship Pollution, Tropical Timber 83, Tropical Timber 94, Wetlands, Whaling
signed, but not ratified: Air Pollution-Sulphur 94, Air Pollution-Volatile Organic Compounds, Desertification

Geography - note: largest West European nation; occasional strong, cold, dry, north-to-northwesterly wind known as mistral

People

Population: 58,609,285 (July 1997 est.)

Age structure:
0-14 years: 19% (male 5,712,739; female 5,449,139)
15-64 years : 65% (male 19,178,683; female 19,126,672)

Figure 4: CIA World Factbook: France.

For the CIA Factbook we chose not to enforce any target structure. W4F will automatically return NSL structures. The retrieval rule is simple since all the countries can be reached through a Web with a standardized name (1 code for each country).

[5] The *World Factbook* is prepared by the Central Intelligence Agency for the use of US Government officials, and the style, format, coverage, and content are designed to meet their specific requirements.[..] The *Factbook* is in the public domain. Accordingly, it may be copied freely without permission of the Central Intelligence Agency (CIA)." (from http://www.odci.gov/cia/publications/factbook)

[6] The template is available at: http://www.odci.gov/cia/publications/factbook/guide.html

Most of the extraction rules are similar. They first identify the node with the corresponding heading and then take the second PCDATA node.
Rule 2 uses a match to get rid spurious information and only return the string that represents a number. This field is looks usually like: `<b>Population:</b> 58,609,285 (July 1997 est.)` in the source code.
Rule 3 is interesting because it involves two conditions. The information about *area* is subdivided into *total*, *land*, *water area* and some extra comments. Here we are only interested in the total area.
Rule 6 takes advantage to transform a sequence of items separated by commas into a list.

```
SCHEMA {}
EXTRACTION_RULES
{
/*(1)*/ geo            = html.body.p[i].b[0]->pcdata[1].txt
                   WHERE html.body.p[i].b[0].txt =~ "Geographic coor";
/*(2)*/ pop            = html.body.p[i].b[0]->pcdata[1].txt, match /([0-9,]+)/
                   WHERE html.body.p[i].b[0].txt =~ "Population";
/*(3)*/ landArea       = html.body.p[i]->i[j]->pcdata[1].txt
                   WHERE html.body.p[i].b[0].txt =~ "Area"
                     AND html.body.p[i]->i[j].txt =~ "land";
/*(4)*/ landBoundary   = html.body.p[i]->i[j]->pcdata[1].txt
                   WHERE html.body.p[i].b[0].txt =~ "Land boundaries"
                     AND html.body.p[i]->i[j].txt =~ "total";
/*(5)*/ coastLine      = html.body.p[i].b[0]->pcdata[1].txt
                   WHERE html.body.p[i].b[0].txt =~ "Coastline";
/*(6)*/ resources      = html.body.p[i].b[0]->pcdata[1].txt, split /,/
                   WHERE html.body.p[i].b[0].txt =~ "Natural resources";
/*(7)*/ capital        = html.body.p[i].b[0]->pcdata[1].txt
                   WHERE html.body.p[i].b[0].txt =~ "National capital";
}
RETRIEVAL_RULES
{
  getCountry(String ciaCode)
        {
          METHOD: GET; URL: "http://www.odci.gov/cia/publications/factbook/$ciaCode$.html";
        }
}
```

The result is of this extraction (setting `$ciaCode$` equals to `"fr"` for France) is presented below:

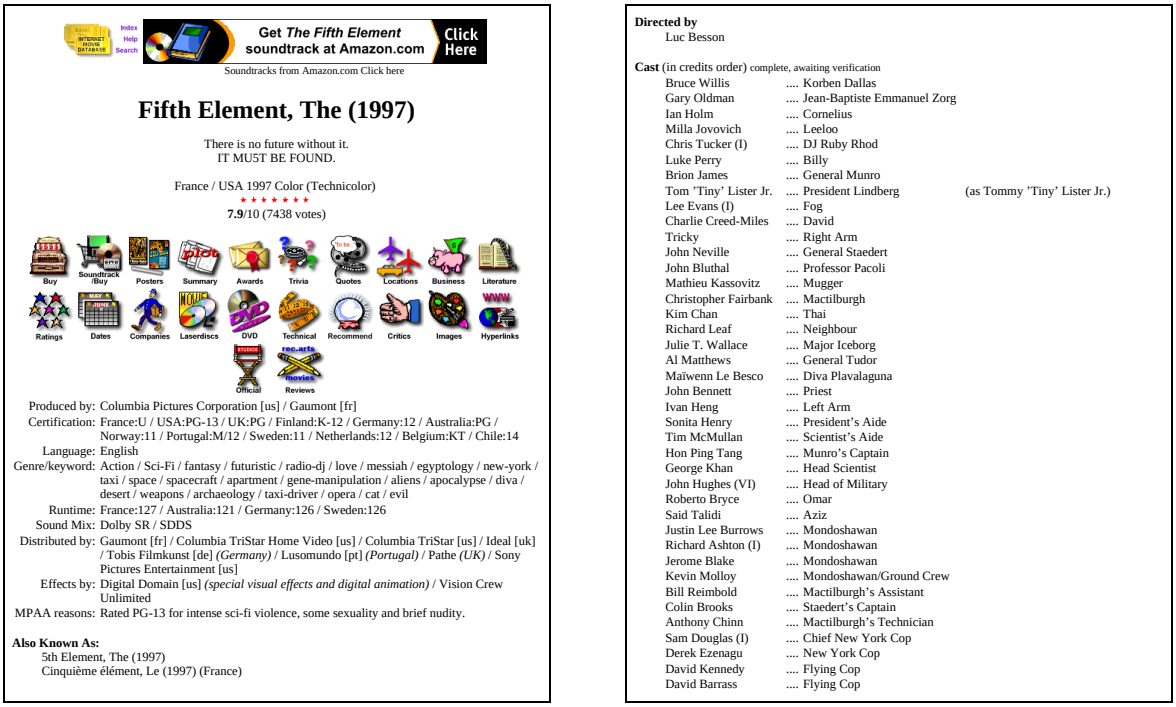
```
capital : Paris
pop : 58,609,285
landArea : 545,630 sq km
resources : [ coal, iron ore, bauxite, fish, timber, zinc, potash ]
geo : 46 00 N, 2 00 E
coastLine : 3,427 km
landBoundary : 2,892.4 km
```

A similar wrapper has been built to extract the url corresponding to each country name. Using both wrappers, it is now possible to write programs that can take advantage of the knowledge of the CIA Factbook.

This application demonstrated that it would be useful to enrich[7] W4F with automatic format conversion in order to directly map a number represented using commas into an integer.

8.2 The Internet Movie Database

The Internet Movie Database[8] is a database that gathers information about movies, actors, directors, etc. The database is updated regular, as new movies are released. It offers entries by movie title, actor name and some more advanced query forms. Movie entries point back to actors. Actor entries point back to movies. Actor entries also to biographical information.



Get *The Fifth Element* soundtrack at Amazon.com Click Here

Soundtracks from Amazon.com Click here

Fifth Element, The (1997)

There is no future without it.
IT MU5T BE FOUND.

France / USA 1997 Color (Technicolor)

7.9/10 (7438 votes)

Buy Soundtrack /Buy Posters Summary Awards Trivia Quotes Locations Business Literature
Ratings Dates Companies Laserdiscs DVD Technical Recommend Critics Images Hyperlinks
Official Reviews

Produced by: Columbia Pictures Corporation [us] / Gaumont [fr]
Certification: France:U / USA:PG-13 / UK:PG / Finland:K-12 / Germany:12 / Australia:PG / Norway:11 / Portugal:M/12 / Sweden:11 / Netherlands:12 / Belgium:KT / Chile:14
Language: English
Genre/keyword: Action / Sci-Fi / fantasy / futuristic / radio-dj / love / messiah / egyptology / new-york / taxi / space / spacecraft / apartment / gene-manipulation / aliens / apocalypse / diva / desert / weapons / archaeology / taxi-driver / opera / cat / evil
Runtime: France:127 / Australia:121 / Germany:126 / Sweden:126
Sound Mix: Dolby SR / SDDS
Distributed by: Gaumont [fr] / Columbia TriStar Home Video [us] / Columbia TriStar [us] / Ideal [uk] / Tobis Filmkunst [de] *(Germany)* / Lusomundo [pt] *(Portugal)* / Pathe *(UK)* / Sony Pictures Entertainment [us]
Effects by: Digital Domain [us] *(special visual effects and digital animation)* / Vision Crew Unlimited
MPAA reasons: Rated PG-13 for intense sci-fi violence, some sexuality and brief nudity.

Also Known As:
5th Element, The (1997)
Cinquième élément, Le (1997) (France)

Directed by
Luc Besson

Cast (in credits order) complete, awaiting verification

Bruce Willis	.... Korben Dallas	
Gary Oldman	.... Jean-Baptiste Emmanuel Zorg	
Ian Holm	.... Cornelius	
Milla Jovovich	.... Leeloo	
Chris Tucker (I)	.... DJ Ruby Rhod	
Luke Perry	.... Billy	
Brion James	.... General Munro	
Tom 'Tiny' Lister Jr.	.... President Lindberg	(as Tommy 'Tiny' Lister Jr.)
Lee Evans (I)	.... Fog	
Charlie Creed-Miles	.... David	
Tricky	.... Right Arm	
John Neville	.... General Staedert	
John Bluthal	.... Professor Pacoli	
Mathieu Kassovitz	.... Mugger	
Christopher Fairbank	.... Mactilburgh	
Kim Chan	.... Thai	
Richard Leaf	.... Neighbour	
Julie T. Wallace	.... Major Iceborg	
Al Matthews	.... General Tudor	
Maïwenn Le Besco	.... Diva Plavalaguna	
John Bennett	.... Priest	
Ivan Heng	.... Left Arm	
Sonita Henry	.... President's Aide	
Tim McMullan	.... Scientist's Aide	
Hon Ping Tang	.... Munro's Captain	
George Khan	.... Head Scientist	
John Hughes (VI)	.... Head of Military	
Roberto Bryce	.... Omar	
Said Talidi	.... Aziz	
Justin Lee Burrows	.... Mondoshawan	
Richard Ashton (I)	.... Mondoshawan	
Jerome Blake	.... Mondoshawan	
Kevin Molloy	.... Mondoshawan/Ground Crew	
Bill Reimbold	.... Mactilburgh's Assistant	
Colin Brooks	.... Staedert's Captain	
Anthony Chinn	.... Mactilburgh's Technician	
Sam Douglas (I)	.... Chief New York Cop	
Derek Ezenagu	.... New York Cop	
David Kennedy	.... Flying Cop	
David Barrass	.... Flying Cop	

Figure 5: The Internet Movie Database: "The Fifth Element"

Wrappers have been generated for movie, actor and actor biography entries. For the purpose of this paper we will only present the wrapper for movies.

```
SCHEMA
{
String title;
```

[7] The user can always define his own Java classes to do the job.

[8] The Internet Movie Database (IMDb: http://www.imdb.com) is an international organization whose objective is to provide useful and up to date movie information freely available on-line, across as many systems and platforms as possible. It currently covers over 150,000 movies with over 2,000,000 filmography entries and is expanding continuously.

```
String[] genres;
String[] cast;
float review;
}
EXTRACTION_RULES
{
/*(1)*/ title   = html.head.title.txt;
/*(2)*/ genres  = html.body.table[0].tr[3].td[1].a[*].txt;
/*(3)*/ cast    = html->a[i]->table[0].tr[j:*].td[0].a[0].getAttr(href)
            WHERE html->a[i].getAttr(name) = "cast"
            AND   html->a[i]->table[0].tr[j].numberOf(td) = 3;
/*(4)*/ review  = html.body.p[1].b[0].txt;
}
RETRIEVAL_RULES
{
  getMovie(String title)
        {
        METHOD: GET; URL: "http://us.imdb.com/Title?$title$" ;
        }
}
```

The interesting extraction rule is 3. Here, the cast is located as the first table following an anchor with a name equal to "cast". Inside the table, we have to extract the first column of all the rows to get the name of actors and actresses. The listing is usually sorted by credits order. For some movies, the listing is interrupted to switch to alphabetical order. From an extraction point of view, we do not want to get this spurious line. In the case of IMDb, regular entries have 3 columns while spurious one have less. The condition simply checks the number of columns.
Another interesting point is the automatic conversion of the review into a float.
The retrieval part is a regular GET. It is worth noting that the retrieving layer assumes that the URL is a valid one. In our case, the calling application has to make sure the title is a valid IMDb title. To do so, the application could make use of another IMDb wrapper that gets a string and returns a list of matching movie titles.

The result is of this extraction (setting `$title$` equals to `"Fifth+Element,+The+(1997)"`) is presented below:

```
review : 7.9
title : Fifth Element, The (1997)
genres : [ Action, Sci-Fi, fantasy, futuristic, radio-dj, love, messiah, egyptology, new-york,
taxi, space, spacecraft, apartment, gene-manipulation, aliens, apocalypse, diva, desert,
weapons, archaeology, taxi-driver, opera, cat, evil ]
cast : [ Bruce Willis, Gary Oldman, Ian Holm, Milla Jovovich, Chris Tucker (I), ... ]
```

8.3 The GIST TV-Guide

The GIST-TV Listings (http://www.gist.com/previewlistings) is a very good illustration of the fact that HTML is clumsy to represent complex information.
Inside the TV-Guide, programs are displayed by slices of three hours. They are displayed in tables, one row for each channel. Columns corresponds to time slots and columns have various width depending on the duration of the program. Moreover, some programs overlap time slices and start on one to finish on another.

The nature of programs is represented by a color code (movies have a blue background). Movies are identified with their year of release, their rating and certification. When the program started in a previous time slice, the starting time is indicated and the program is preceded by the < sign. Obviously we should not expect extraction to capture all the semantics of the page (it is even hard for human beings the first time); however, we should be able to extract enough information to reconstruct the TV-Guide.

←	6pm	6:30pm	7pm	7:30pm	8pm	8:30pm	→
VC1 21	<<Home Alone 3 (1997) ** (PG) (04:30)	Switchback (1997) ** (R)				Spice World (1997) ** (PG)	VC1 21
COMEDY 22	Norm MacDonald	Make Me Laugh	The Daily Show	Win Ben Stein's Money	National Lampoon's Vacation (1983) *** (R)		COMEDY 22
PBS 23	News	Images/Imagenes	State of the Arts	News	This Old House	About Your House With Bob Yapp	PBS 23
VC4 24	As Good as It Gets (1997) *** (PG-13)					As Good as It Gets (1997) *** (PG-13)	VC4 24
FX 25	21 Jump Street		Miami Vice		The X-Files		FX 25
HBO 26	Sinbad's Summer Jam 4: '70s Soul Music Festival						HBO 26
HBOPL 27	<<Pontiac Moon (1994) * (PG-13) (04:45)	Born on the Fourth of July (1989) *** (R)					HBOPL 27
TDC 28	Sea Wings		Gimme Shelter		Wild Discovery		TDC 28
FOOD 29	Pick of the Day	Julia Child	Cooking Live!		Ready Set Cook!	Grillin' & Chillin'	FOOD 29
ARTS 30	Northern Exposure		Law & Order		Biography		ARTS 30

Figure 6: A screen shot from TV-Guide

What we want to extract is the list of channels, and for each channel the list of program entries, where an entry is described by its name, its nature and its duration. The duration of a program is represented by the "width" of its column, i.e. the value of the attribute `COLSPAN`.

However, in order to make the TV-Listing easier to use, channels displays by rows of 10, with a navigation table header to help navigate from one time-slice to the other. Moreover, channel names are displayed as the first and last column of each row, using a bold font. The first problem can easily be solved using a range specifier such as `[1-]` that excludes the first row. For the second one, we have to exclude columns for which the text is in bold. Note that we also get rid of the optional `'<'` sign.

A screen shot of how it appears to the user is presented in Figure 6.

The extraction rules are presented below:

```
EXTRACTION_RULES
{
listing = html.body.table[1-].tr[1-](   .td[0].txt                              // channel name
                                     # .td[i:*](   .txt, match/[<]*(.*)/      // program name
                                                 # .getAttr(colspan)          // program duration
                                                 # .getAttr(bgcolor)    ) )   // program genre
    WHERE html.body.table[1-].tr[1-].td[i].font[0].numberOf(b) == 0;
}
```

The result of the extraction (for the time slice starting at 6:00pm and corresponding to the screen shot) is presented below :

```
[ VC1 21
   [  [ "Home Alone 3 (1997) ** (PG)(04:30)",          null, "B0E0E6" ]
      [ "Switchback (1997) ** (R)",                     "4", "B0E0E6" ]
      [ "Spice World (1997) ** (PG)",                 null, "B0E0E6" ]  ]
  COMEDY 22
   [  [ "Norm MacDonald",                             null, "5F9EA0" ]
      [ "Make Me Laugh",                              null, "FFFAF0" ]
      [ "The Daily Show",                             null, "F5DEB3" ]
```

```
     [ "Win Ben Stein's Money",                          null, "FFFAF0" ]
     [ "National Lampoon's Vacation (1983) *** (R)",    "2", "B0E0E6" ]  ]
  PBS 23
   [  [ "News",                                         null, "FFFAF0" ]
      [ "Images/Imagenes",                              null, "F08080" ]
      [ "State of the Arts",                            null, "FFFAF0" ]
      [ "News",                                         null, "FFFAF0" ]
      [ "This Old House",                               null, "FFE1FF" ]
      [ "About Your House With Bob Yapp",               null, "FFE1FF" ]  ]
  VC4 24
   [  [ "As Good as It Gets (1997) *** (PG-13)",        "5", "B0E0E6" ]
      [  "As Good as It Gets (1997) *** (PG-13)",     null,  "B0E0E6" ]  ]
  ...
]
```

Given this, it is up to the calling application to reconstruct the TV-Guide. Knowing the starting time of the time slide (6:00pm) and given that a column unit represent half an hour, it is straightforward to get the schedule of each program. The nature of the program consists of a simple table look-up ("B0E0E6" stands for the blue color and represents movies). For movies, the calling application just has to extract the various components (title, year, ratings, certification).
For instance, the movie *Switchback* is the second element of the list for channel *VC21* and its colspan value is 4. The first element has a null colspan value which means it represents only half an hour. The starting time of our movie is therefore 6:00pm + $\frac{1}{2}$hour and its ending time is 6:00pm + $\frac{1}{2}$hour + $4 * \frac{1}{2}$hour.

W4F cannot do everything. Its purpose is to do extraction. Some higher applications are in charge of further processing.

8.4 Hoover's Company Profiles

Hoover's online (http://www.hoovers.com) offers free profiles for more than 3,200 companies (US and foreign) and proposes some extra in-depth details through fee-based membership. Profiles are accessed through a form by name, ticker of keyword. Each company is given a specific id (i.e. key) that is used for direct access. Cross-references (i.e.hyperlinks) inside Hoover's online site always use this id.
Hoover's online also provides *industry snapshots*: they describe the state of the sector and cite major companies involved. It appears to be a useful entry point when one does not know who is a major player in the sector.

We have written two wrappers for the the Hoover's web site: one to retrieve major companies of a sector and one to extract basic information out of a compnay profile. Unfortunately, a lot of interesting information only available through membership.

The industry snapshot (see Figure 7) is a document that describes the sector in plain words. Pictures, diagrams, charts are mixed with text, references to companies of the sector and pointers to press articles. The HTML tagging is not really useful because it is used for the display (to align a picture with a chunk of text) and does not carry any structure. Companies we are interested in are cross-referenced in the Hoover's website. We just have to look for hyperlinks where the url

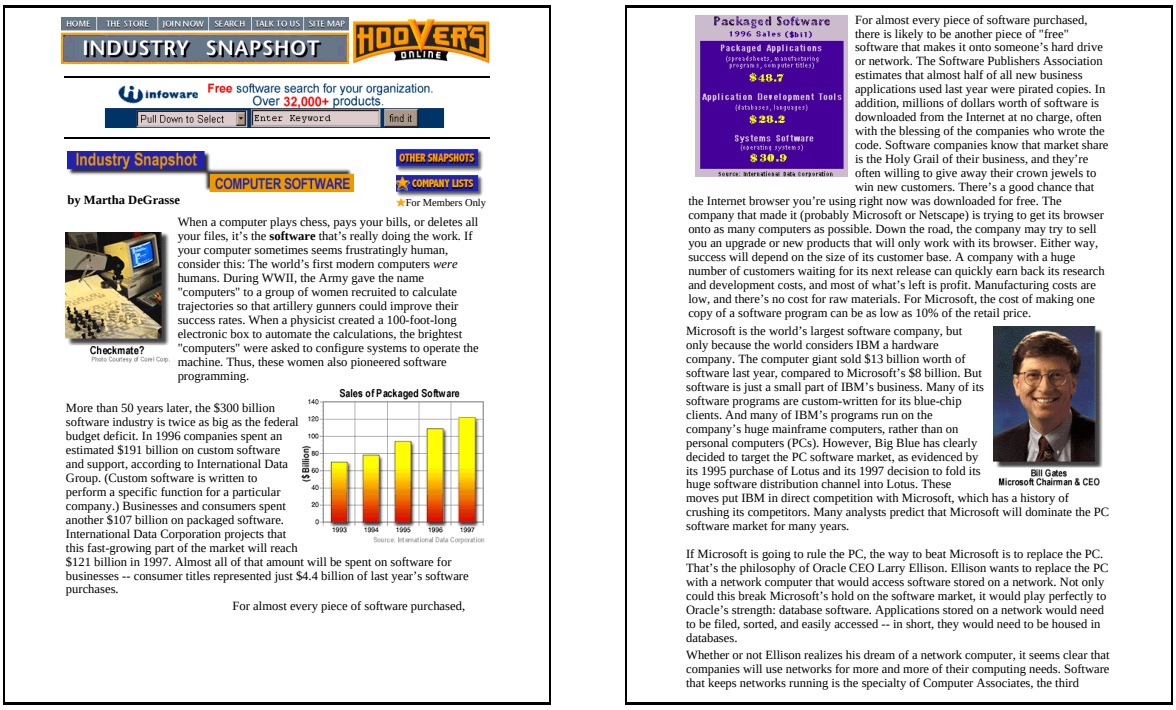


Figure 7: Hoover's Industry Snapshot: Computer Software

matches a typical Hoover's company entry (i.e. `capsule?id=`).

The extraction rule for the industry snapshot is presented below:

```
EXTRACTION_RULES
{
        company = html.body->a[i:*] ( .txt # .getAttr(href), match /id=([0-9]+)/ )
            WHERE html.body->a[i].getAttr(href) =~ "capsule";
}
```

The result of the extraction[9] for the sector of computer software is presented below:

```
[ [ International Data Group  40238 ],          [ Computer Associates      10383 ],
  [ Microsoft                 14120 ],          [ SAP                      91277 ],
  [ Netscape                  42472 ],          [ Check Point              51768 ],
  [ IBM                       10796 ],          [ VocalTec                 47900 ],
  [ Lotus                     13985 ],          [ Corel                    42379 ] ]
  [ Oracle                    14337 ],
```

Company profiles (see Figure 8) carry more structure. Most items are preceded by a label to identify them. The extraction rules for compnay profiles are given below:

```
EXTRACTION_RULES
{
```

[9] The second item represents the `id` used for cross-references.

```
name  = html.body.center[0].table[1].tr[0].td[0].font[0].txt;
addr  = html.body.center[0].table[1].tr[2].td[0].table[0].tr[0].td[0].font[0].pcdata[*].txt;
phone = html.body.center[0].table[1].tr[2].td[0].table[0].tr[0]
              .td[1].font[0].pcdata[0].txt, match /([0-9+\-]+)/;
fax   = html.body.center[0].table[1].tr[2].td[0].table[0].tr[0]
              .td[1].font[0].pcdata[1].txt, match /([0-9+\-]+)/;
url   = html.body.center[0].table[1].tr[2].td[0].table[0].tr[1].td[0].font[0].a[0].getAttr(href);
info  = html.body.center[0].table[1].tr[2].td[0].txt,
             match /(?:http:\/\/[^ ]+)(?:.*?[(]ABI[)])?(.*?)Top Competitors/;

ticker                = html.body.center[0].table[2].tr[0].td[0].font[1].b[0].a[0].txt;
net_income_in_million = html.body.center[0].table[2]->b[i]->pcdata[1].txt
                  WHERE html.body.center[0].table[2]->b[i].txt =~ "Net Inc.";
number_of_employees   = html.body.center[0].table[2]->b[i]->pcdata[1].txt
                  WHERE html.body.center[0].table[2]->b[i].txt =~ "Employees:";
ceo                   = html.body.center[0].table[2]->b[i]->pcdata[1].txt
                  WHERE html.body.center[0].table[2]->b[i].txt =~ "CEO";
}
```


```
name : "Intel Corporation"
phone : "408-765-8080"
fax : "408-765-6284"
address : [ "2200 Mission College Blvd.",
            "Santa Clara, CA 95052-8119" ]
web_site : "http://www.intel.com"
capsule :  "Intel is the world's #1 maker of microprocessors [...]
             [...] About 55% of its sales are outside the US."
num_of_employees : "63,700"
net_income_in_million : "$6,945.0"
ceo : "Andrew S. Grove"
ticker : "INTC"
```

Figure 8: Hoover's Company Profile: Intel Inc. and the corresponding extraction.

There are a couple of interesting points from the HEL point of view. For the address field (`addr`) we retrieve a list of strings, each string corresponding to one line in the address description. It appears that addresses can vary a lot, depending on countries. The wildcard guarantees to get all of them.

For the information converning the company (`info`), we cannot directly locate the relevant paragraph. The issue here is related to our object model. A chunk of plain text is represented by a PCDATA leaf node. However, if there is a tag inside the text (say a word is emphasized using `<EM>`), it is now a PCDATA, followed by a node `EM` followed by a PCDATA. To solve our problem, we extract the entire zone and use a regular expression pattern to extract only the relevant part: after the company web-site or the list of *subsidiary locations*[10] if any, and before the list of its

[10] Subsidiary locations are followed by `"(ABI)"` (see Figure 8).

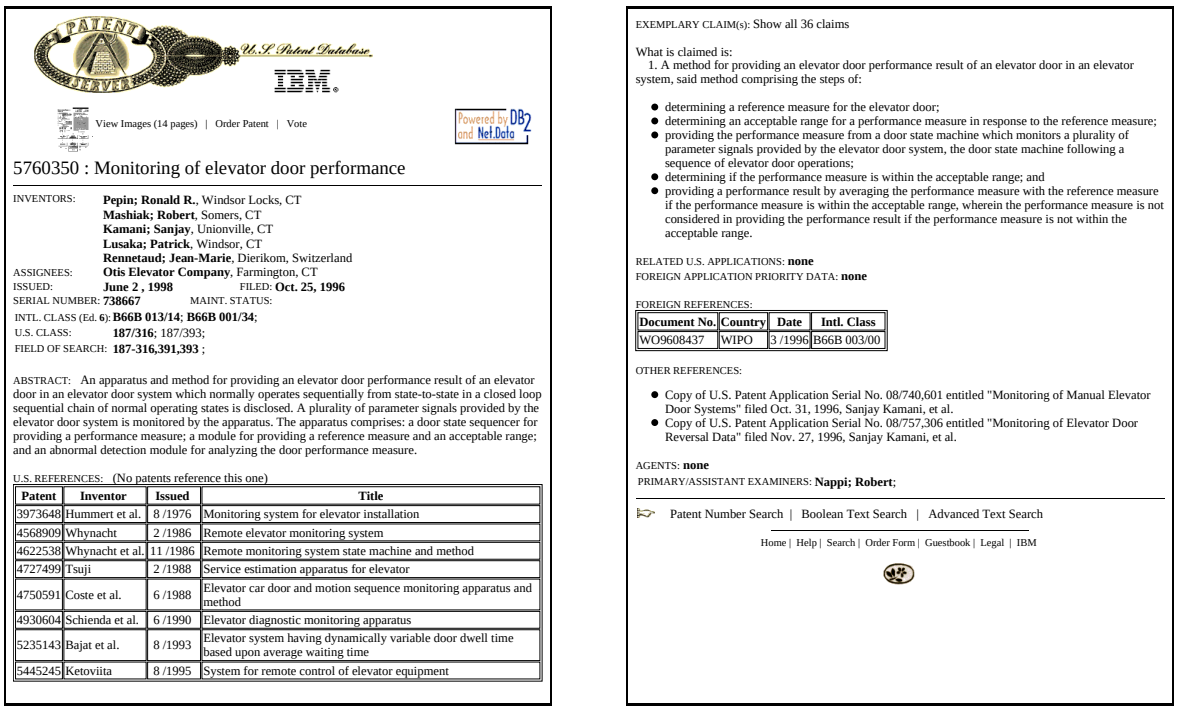
competitors).

For information located on the right part of the document (`table[2]`), we simply use conditions to match a given keyword and extract the following value.

The snapshot only references major players. However, using these entry points will lead to other entry points since company profiles reference competitors. By using both services together, it is really easy to get a representative list of the companies involved in the sector.

8.5 The IMB Patent Server

The IBM Patent Server (http://www.patents.ibm.com) grants access to over 26 years of U.S. Patent & Trademark Office (USPTO) patent descriptions.
The service offers multiple entry points by title, keyword, number, as well as some boolean queries. For a given query, a list of matches is presented, and then it is possible to go to the patent description.



U.S. Patent Database

View Images (14 pages) | Order Patent | Vote

5760350 : Monitoring of elevator door performance

INVENTORS: **Pepin; Ronald R.**, Windsor Locks, CT
Mashiak; Robert, Somers, CT
Kamani; Sanjay, Unionville, CT
Lusaka; Patrick, Windsor, CT
Rennetaud; Jean-Marie, Dierikom, Switzerland
ASSIGNEES: **Otis Elevator Company**, Farmington, CT
ISSUED: **June 2 , 1998** FILED: **Oct. 25, 1996**
SERIAL NUMBER: **738667** MAINT. STATUS:
INTL. CLASS (Ed. 6): **B66B 013/14**; **B66B 001/34**;
U.S. CLASS: **187/316**; 187/393;
FIELD OF SEARCH: **187-316,391,393** ;

ABSTRACT: An apparatus and method for providing an elevator door performance result of an elevator door in an elevator door system which normally operates sequentially from state-to-state in a closed loop sequential chain of normal operating states is disclosed. A plurality of parameter signals provided by the elevator door system is monitored by the apparatus. The apparatus comprises: a door state sequencer for providing a performance measure; a module for providing a reference measure and an acceptable range; and an abnormal detection module for analyzing the door performance measure.

U.S. REFERENCES: (No patents reference this one)

Patent	Inventor	Issued	Title
3973648	Hummert et al.	8 /1976	Monitoring system for elevator installation
4568909	Whynacht	2 /1986	Remote elevator monitoring system
4622538	Whynacht et al.	11 /1986	Remote monitoring system state machine and method
4727499	Tsuji	2 /1988	Service estimation apparatus for elevator
4750591	Coste et al.	6 /1988	Elevator car door and motion sequence monitoring apparatus and method
4930604	Schienda et al.	6 /1990	Elevator diagnostic monitoring apparatus
5235143	Bajat et al.	8 /1993	Elevator system having dynamically variable door dwell time based upon average waiting time
5445245	Ketoviita	8 /1995	System for remote control of elevator equipment

EXEMPLARY CLAIM(s): Show all 36 claims

What is claimed is:
1. A method for providing an elevator door performance result of an elevator door in an elevator system, said method comprising the steps of:

- determining a reference measure for the elevator door;
- determining an acceptable range for a performance measure in response to the reference measure;
- providing the performance measure from a door state machine which monitors a plurality of parameter signals provided by the elevator door system, the door state machine following a sequence of elevator door operations;
- determining if the performance measure is within the acceptable range; and
- providing a performance result by averaging the performance measure with the reference measure if the performance measure is within the acceptable range, wherein the performance measure is not considered in providing the performance result if the performance measure is not within the acceptable range.

RELATED U.S. APPLICATIONS: **none**
FOREIGN APPLICATION PRIORITY DATA: **none**

FOREIGN REFERENCES:

Document No.	Country	Date	Intl. Class
WO9608437	WIPO	3 /1996	B66B 003/00

OTHER REFERENCES:

- Copy of U.S. Patent Application Serial No. 08/740,601 entitled "Monitoring of Manual Elevator Door Systems" filed Oct. 31, 1996, Sanjay Kamani, et al.
- Copy of U.S. Patent Application Serial No. 08/757,306 entitled "Monitoring of Elevator Door Reversal Data" filed Nov. 27, 1996, Sanjay Kamani, et al.

AGENTS: **none**
PRIMARY/ASSISTANT EXAMINERS: **Nappi; Robert**;

Patent Number Search | Boolean Text Search | Advanced Text Search

Home | Help | Search | Order Form | Guestbook | Legal | IBM

Figure 9: The IBM Patent Server

Patents can be directly accessed through their unique *patent number*. It corresponds to a GET method with a parameterized url such as: http://www.patents.ibm.com/details?patent_number=5760350. A patent description (for the "Monitoring of elevator door performance") is presented in Figure 9. The structure of a patent is rather tricky because many fields are optional: some have no references, some have no related U.S./Foreign applications, etc. Moreover a lot of structure is carried by text

values. The field *inventors* contains a list of people and for each of them his last name, first name and country. The description also makes references to other patents and it is important to be able to capture them (name of the patent and url shortcut).

The extraction rules are presented below:

```
EXTRACTION_RULES
{
// in the following lines, for readability reasons
// html.body.table[0].tr[1].td[0] has been replaced by html...td[0]
title                = html...td[0].font[0].pcdata[0].txt, match /[:] (.*)/ ;
number               = html...td[0].txt, match /([0-9]+)/ ;
inventors            = html...td[0].table[0].tr[0].td[1].b[*].txt, split /;/ ;
assignees            = html...td[0].table[0].tr[1].td[1].b[0].txt ;
issued_date          = html...td[0].table[0].tr[i].td[1].b[0].txt
                 WHERE html...td[0].table[0].tr[i].td[0].txt =~ "ISSUED" ;
filed_date           = html...td[0].table[0].tr[i].td[4].b[0].txt
                 WHERE html...td[0].table[0].tr[i].td[3].txt =~ "FILED" ;
serial_number        = html...td[0].table[0].tr[i].td[1].txt
                 WHERE html...td[0].table[0].tr[i].td[0].txt =~ "SERIAL NUMBER" ;
main_status          = html...td[0].table[0].tr[i].td[4].txt
                 WHERE html...td[0].table[0].tr[i].td[3].txt =~ "STATUS" ;
intl_class           = html...td[0].table[1].tr[0].td[1].txt, split /;/ ;
us_class             = html...td[0].table[1].tr[1].td[1].txt, split /;/, split /// ;
field_of_search      = html...td[0].table[1].tr[2].td[1].txt, split /;/ ;
abstrac              = html...td[0].pcdata[0].txt ;
references           = html...td[0].table[2].tr[1-]( .td[2].txt, split /// #
                                                    .td[3].txt # .td[0].txt # .td[1].txt ) ;
references_to_this   = html...td[0].a[i]( .txt, match /([0-9]+)/ # .getAttr(href) )
                 WHERE html...td[0].a[i].txt =~ "that reference this one" ;
claims               = html...td[0]->a[i]( .txt, match /([0-9]+)/ # .getAttr(href) )
                 WHERE html...td[0]->a[i].txt =~ "Show.*?claim" ;
us_applications      = html...td[0].table[i].tr[1-].td[*].txt
                 WHERE html...td[0].table[i].tr[0].th[0].txt =~ "Patent No" ;
foreign_applications = html...td[0].table[i].tr[1-].td[*].txt
                 WHERE html...td[0].table[i].tr[0].th[0].txt =~ "Application No" ;
foreign_references   = html...td[0].table[i].tr[1-].td[*].txt
                 WHERE html...td[0].table[i].tr[0].th[0].txt =~ "Document No" ;
attorney             = html...td[0].table[i].tr[0].td[1].txt,  split /;/
                 WHERE html...td[0].table[i].tr[0].td[0].txt =~ "ATTORNEY" ;
examiners            = html...td[0].table[i].tr[1].td[1].txt, split /;/
                 WHERE html...td[0].table[i].tr[1].td[0].txt =~ "EXAMINERS" ;
}
```

As one can see, the description can become quite complex, event though most of the rule look alike. The interesting points are the frequent use of match and split operators.

8.6 Experience with HEL and W4F

Our exprerience with W4F and HEL and still limited. So far we have written wrappers for the following sources:

- Internet Movie Database

- CIA World Factbook
- GIST TV-Listing
- IBM Patent Server
- France Telecom White Pages
- CD-Now Music Store
- Hoover's online (company profile and industry snapshot)

Given a Web document, it is easy to find a path-expression to extract the desired information using the wysiwyg interface. The problem is really for a given source to find a sample of Web documents that display most irregularities to be encountered when dealing with the source. Most of the time it means refining the path-expression to make it more fault-tolerant. The use of the arrow operator with a condition is often useful.

Another issue is the retrieval layer. Some sources use frames which obliges to have several retrieval layers.

9 Related Work

There is a lot of published work in the literature, regarding the issue of *wrappers*. Not surprisingly, most of them come from the areas such as database, information extraction and machine learning. However, the emergence of the Web as *the* new medium of communication has increased this interest.

9.1 What structure to use

As far as the issue of wrapping-up of Web documents is concerned, approaches can be split in two categories.

On the one hand, some people [MAM+98, KMA+, HGMC+, HKL+, Bre] view a Web document only as a flow of tokens and completely ignore tag-based hierarchy. They process the tokens either through an expressive grammar (see [MAM+98]) or through simpler regular expressions (see [KMA+, HGMC+, HKL+, Bre]). In [MMK], the grammar defines a document hierarchy, for each type of document.

The problem is that HTML has somehow to be reinvented for each wrapper. In [FFF], they have to build a custom parser specifically for the CIA World Factbook. In [Nav97], they redefine all the HTML tags as regular expressions. [MMK] introduces the notion of *landmarks* that represent information tokens useful for document navigation, like HTML `<H1>` headings.

The advantage of using low-level tools is performance. [HGMC+] reports a speed of 10k/sec for a native HTML parser vs. 2MB/sec for their own extraction tools based on Python `find` command. For extraction based only on regular expressions, languages like Perl and Python offer the ideal procedural framework.

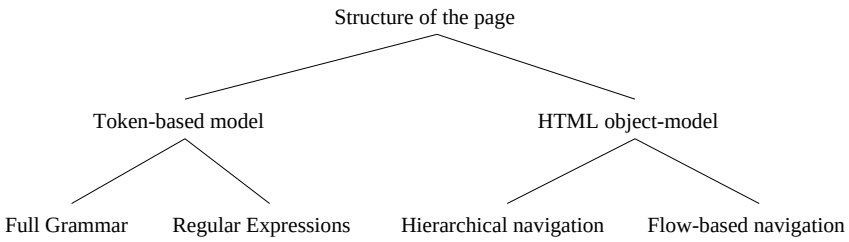


Figure 10: What structure to use

On the other hand, some people want to take advantage of the structure implied by the HTML tags. [AM98, All97] really use the hierarchical structure of HTML documents. The benefit of this approach comes from the fact that indeed many Web documents are highly structured (specially, when they are generated from databases which are highly structured sources). However, navigation along this explicit structure is sometimes restricted to the hierarchy (see [All97]) and does not respect the flow of the document. Pure hierarchical navigation is not capable of capturing structure with a thinner granularity (inside tags).

9.2 How to build a wrapper

The construction of a wrapper can be done manually, automatically or in a semi-automatic way.

The manual generation of a wrapper often involves the writing of adhoc code. The creator has to spend quite sometime understanding the structure of the document (if any) and translate it into an abstract representation. In ([HGMC$^+$], [MAM$^+$98], [HKL$^+$], [Bre]), wrappers are built in an ad-hoc way using human expertise by looking at the raw HTML source. Web-OQL (see [AM98]) takes advantage of a generic mapping between the HTML structure and the OQL object-model but the extraction is performed via highly complicated `select-from-where` queries. Here again, the HTML source has to be perfectly understood.

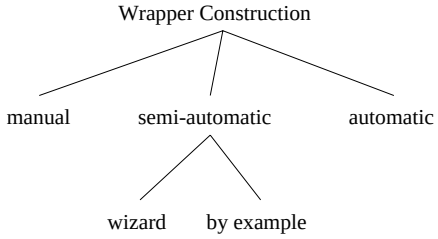


Figure 11: Wrapper generation

Semi-automatic generation can involve the use of a wizard to help designing the wrapper. The toolkit provided by Web-Methods (see [All97]) displays to the user the HTML structure, as understood by the system: the user then just has to pick whatever information he is interested in.

[KMA+] offers a *demonstration-oriented* interface where the user shows the system what information to extract.

"Automatic" wrapper generation uses machine-learning (ML) techniques [Mit97]. First the problem of extraction must be stated in ML way. [Kus] for instance identifies classes of wrappers like left-right where the token to be extracted is located by what comes before (left) and what comes after (right). Then the system has to be fed with some training examples. In many cases, the learning has to be supervised. Even "automatic" generation requries the intervention of human experts and the statement of the problem in terms of ML is often tricky.

In W4F, we rely on human expertise. However, we offer the user a *wysiwyg* interface where he can select information by just picking it as he sees it. Even when there is a need to go deeper into the HTML source, the user is always guided by the default path-expression offered by the system. ML techniques could be used to improve the quality of the default path-expression returned by the system. For instance, W4F cannot return path-expressions for arrays. ML could be used to infer the correct range specifier given a set of items.

9.3 How to describe the extraction

The extraction processing can be described in a declarative or procedural way (often as part of a program). The advantage of the procedural way is that it is directly embedded into the application. However, it is harder to maintain. [MAM+98] used procedural descriptions with their *EDITOR* language.

The declarative approach makes wrappers easier to read, to maintain and to re-use. [PGH+95, AM98, Bre, HKL+, All97] use a declarative specification for extraction. [MAM+98] and [KMA+] use a mix with a declarative grammar-based approach.

[AM98] and [HKL+] use declarative specifications that include browsing: they can follow hyperlinks and move from page to page. They are also query languages per-se: Florid is based on F-Logic while Web-OQL is based on OQL.

9.4 Extraction data-model

Wrappers are also in charge of providing a structured access to the extracted information. [HKL+] uses F-Logic to map extracted information into an object-oriented schema. For [AM98], everything is an QOL instance from the beginning. [HGMC+] converts the extracted information into the OEM format (semi-structured data). Most of the others look at extracted information as plain strings.

While some wrappers can directly build structured objects (complex objects, relations) out of the information, we prefer to use the NSL intermediate representation in order to favor re-use and tunability. For instance, OEM objects in Tsimmis carry attribute names. The advantage of our *anonymous* NSL structure is that it can be used (consumed) by anyone. The extraction layer can be resued as is.

W4F separates the wrapper into 3 independent layers to maximize re-use. Extraction information is stored as NSL that can be mapped/consumed into any data-structure. Extraction takes advantage of the HTML structure of the document but provides at the same time regular expression capabilities to capture finer-grain structure. This multi-granularity approach allows an effective

and robust description of the source. The price to pay is to have each HTML document and its tree built. Wrappers are generated using W4F wysiwyg interface by a human expert, since in any case a human expert is required.

10 Conclusion

11 Appendix A: HEL BNF specification

References

[All97] Charles Allen. WIDL: Application Integration with XML. *World Wide Web Journal*, 2(4), November 1997.

[AM98] Gustavo Arocena and Alberto Mendelzon. WebOQL: Restructuring Documents, Databases, and Webs. In *Proc. ICDE'98*, Orlando, February 1998.

[Bre] Stéphane Bressan. *Grenouille Version 1.1.* COntext INterchange project, `http://context.mit.edu/~coin/demos/wrapper`.

[FFF] Gleb Franck, Adam Farquhar, and Richard Fikes. Building a large Knowledge Base from a Structured Source: The CIA World Factbook.

[Fla98] David Flanagan. *JavaScript: The Definitive Guide.* O'Reilly & Associates, 3rd edition, 1998.

[HGMC+] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting Semistructured Information from the Web.

[HKL+] Rainer Himmeroder, Paul-Th. Kandzia, Bertram Ludasher, Wolfgang May, and Georg Lausen. Search, Analysis, and Integration of Web Documents: A Case Study with FLORID.

[KMA+] Craig A. Knoblock, Steven Minton, Jose Luis Ambite, Naveen Ashish, Praguesh Jay Modi, Ion Muslea, Andrew G. Philipot, and Sheila Tasheda. Modeling Web sources for Information Integration.

[Kus] Nicholas Kushmerick. Wrapper induction: Efficiency and Expressiveness.

[MAM+98] 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.

[Mit97] Tom Mitchell. *Machine Learning.* McGraw Hill, 1997.

[MK98] Chuck Musciano and Bill Kennedy. *HTML: The Definitive Guide.* O'Reilly & Associates, 2nd edition, 1998.

[MMK] Ion Muslea, Steven Minton, and Craig A. Knoblock. Wrapper Induction for Semistructured, Web-base Information Sources.

[Nav97] 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.

[PGH+95] Yannis Papakonstantinou, Ashish Gupta, Hector, Garcia-Molina, and Jeffrey Ullman. A Query Translation Scheme for Rapid Implementation of Wrappers. In *Proc. DOOD'95*, 1995.

[WCS96] Larry Wall, Tom Christiansen, and Randal L. Schwartz. *Programming Perl.* O'Reilly & Associates, 1996.