

# Semantic Wrappers for Semi-Structured Data Extraction<sup>1</sup>

David Camacho<sup>2†</sup> and Maria D. R-Moreno<sup>‡</sup> and David F. Barrero<sup>‡</sup> and Rajendra Akerkar<sup>§</sup>

<sup>†</sup>Computer Science Department. Universidad Autónoma de Madrid  
Madrid, Spain. E-mail:david.camacho@uam.es

<sup>‡</sup> Departamento de Automática. Universidad de Alcalá  
Madrid, Spain. E-mail: {mdolores, dfbarrero}@aut.uah.es

<sup>§</sup> Department of Computer Science, Norwegian University of Science & Technology, Norway.  
Technomathematics Research Foundation, Kolhapur 416001, India.  
E-mail:raakerkar@yahoo.com

Received 8 June, 2007; accepted in revised form 4 November, 2008

## *Abstract:*

In this paper, we propose an approach to extract information from HTML pages and to add semantic (XML) tags to them. Wrapping is an essential technique used to automatically extract information from Web sources. This paper describes both, a general approach based on rules, which can be used to automatically generate wrappers, and an assistant generator wrapper called WebMantic. We also provide some experimental results to show that both the rule generation process and the preprocessing task are fast and reliable.

© 2008 European Society of Computational Methods in Sciences and Engineering

*Keywords:* Information Extraction, Web data processing/representation, Wrappers generation.

*Mathematics Subject Classification:* 68U15,68T30

*CoLe Classification Index:* Computer Science - Section A

## 1 Introduction

In the future, Web documents will be based on the Semantic Web specifications and different related technologies (XML [1], XPath [4], or Web Services [9]) will be used. However, currently the Web provides a huge amount of unstructured and non-semantic information available for both, users, and automatic crawler programs. To build systems able to access the information stored in the HTML sources, wrappers are commonly used [15]. Wrappers are specialized programs that automatically extract data from documents and convert the information stored into a structured

<sup>1</sup>Published electronically November 20, 2008

<sup>2</sup>Corresponding author.

format Three main functions need to be implemented in a Wrapper: First, they must be able to download HTML pages from a website. Second, they must search for, recognize, and extract the specified data. Third, they have to save this data in a suitably structured format to enable further manipulation. XML is very interesting to structure information, as it is, de-facto, the standard data format in recent Web specifications such as XHTML or SOAP [9] and therefore there are many tools that can use it [2, 5, 14, 17, 18].

Although the Web is evolving to build sites with structured and semantic information, these new kind of Web sites are meant to be deployed in business-to-business (B2B) scenarios. Therefore, most of users (and Web applications) will continue to access data in HTML format. Two main research fields have been involved in the process of extracting and managing the information stored on the Web [10]: Information Gathering (IG) [11, 16, 20] that integrates several different information sources with the aim of querying them as if they were a single information source, and Information Extraction (IE) [8, 12, 13, 22], IE is the process, or processes, necessary to look for a desired piece of information (inside a particular document).

This paper presents both, a general approach based on rules [7], that can be used to automatically generate wrappers, and an assistant generator wrapper (called WebMantic) that builds the wrapper. Our approach allows us to create wrappers that obtain XML documents from HTML pages. We have defined a flexible filtering and preprocessing technique that allows us to translate some pieces of information, that contain the desired information, into a more understandable and semantic representation. Only the required portions of the page will be translated. Non-selected parts of the HTML document will be ignored. Finally, we will work under a reasonable limitation: only structured information, such as data stored in lists or tables, will be considered. Our simple algorithm allows us to generate some predefined rules that can be used to represent complex and nested structures in the Web page. This provides a simple solution to the complex problems of dealing with missing or unordered components in a structure (simply these components are filtered), and then extracting the information from very nested structures. The interaction with the users provides the structures that will be extracted. This adds flexibility to the extraction process because the user focuses on the target structure (or structures) of interest, and does not have to worry with the structure of the Web page that is analyzed by WebMantic. This is an important advantage of our approach.

This paper is structured as follows. In Section 2, the rules that are used to translate HTML documents into XML documents are described. In Section 3 our approach to the problem of wrapper generation is shown. Section 4 provides a description about how our wrapper generation technique can be applied in one actual Web site. Section 5 shows how the wrappers generated using WebMantic can be integrated into an information agent. In Section 6 some experimental results about time performance in the translation (from HTML to XML) process once the Semantic Generators are used is shown. Finally, Section 7 summarizes the conclusions of the paper.

## 2 The Semantic Generators

We define a Semantic Generator, or  $S_g$ , as a set of rules ( $HTML_2XML$ ), that can be used to translate HTML documents into XML documents. A Semantic Generator ( $S_g$ ), is built by several rules which transform a set of non-semantic HTML tags into a set of semantic XML tags. A semantic generator  $S_g$  is able to transform a set of HTML structures into the desired XML format. A  $S_g$  can be described like a non-empty set of  $HTML_2XML$  rules that verify the format shown in expression 1.

$$HTML_2XML_i = \langle header \rangle IS \langle body \rangle \#num \quad (1)$$

The *header* of the rule represents the HTML tag that will be preprocessed. For instance, if the tag `< table >` appears in the header of the rule, all the tokens between this tag and the closing token `>` will be removed (so the parameters inside this tag and their values will be ignored). Moreover, all the HTML tags inserted between the tags `< table >` and `< /table >` will be also ignored. For instance, the next HTML code that represents a table cell:

```
<TR>
  <TD>
    <A href="javascript:sg(6678)"> <B> Madrid </B> </A>
  </TD>
</TR>
```

will be represented in the header of a *HTML<sub>2</sub>XML* rule as:

```
<table.tr.td> IS <My_XML_tag>
```

and the next XML instance will be obtained:

```
<My_XML_tag> Madrid </My_XML_tag>
```

When this cell is processed, the `< tr >`, `< td >`, `< Ahref... >`, and the `< B >` tags, will be removed and only the textual information “Madrid” will be considered. The *body* of the rule corresponds to the XML (semantic) tag that will be used to translate the document. And finally, the parameter `#num` provides the number of HTML tags (in this example the number of possible cells with the same structure) that are affected by this rule (all the *HTML<sub>2</sub>XML* rules have a predefined value of 1). Figure 1 shows some examples of *HTML<sub>2</sub>XML* rules, that belong to a semantic generator (see Section 3.3) which is able to transform some table cells.

<pre>&lt;table&gt; IS &lt;table-countries&gt; #1 &lt;table.tr&gt; IS &lt;data-record&gt; #4 &lt;table.tr.td&gt; IS &lt;country&gt; #4</pre>
---

Figure 1: Some *HTML<sub>2</sub>XML* rules extracted from a Semantic Generator

The  $S_g$  can be applied in a particular Web page, or to a subset of pages with the same structure. The *HTML<sub>2</sub>XML* rules structure can be used to identify nested structures, so it is possible through the utilization of these rules to transform nested tables, or lists, into a more understandable (and easier to manage) structures.

### 3 The WebMantic approach

This section provides the description about three important issues: the architecture of a system that allows to obtain the semantic generators for a particular Web site, the algorithm used by this system to obtain them, and finally some simple examples to illustrate how these semantic generators can be generated.

#### 3.1 The WebMantic architecture

We have implemented an application, named WebMantic, that is able to obtain the Semantic Generators and to translate a HTML page into a XML document using the related  $S_g$ . Figure 2 shows the architecture of WebMantic. WebMantic modules are:

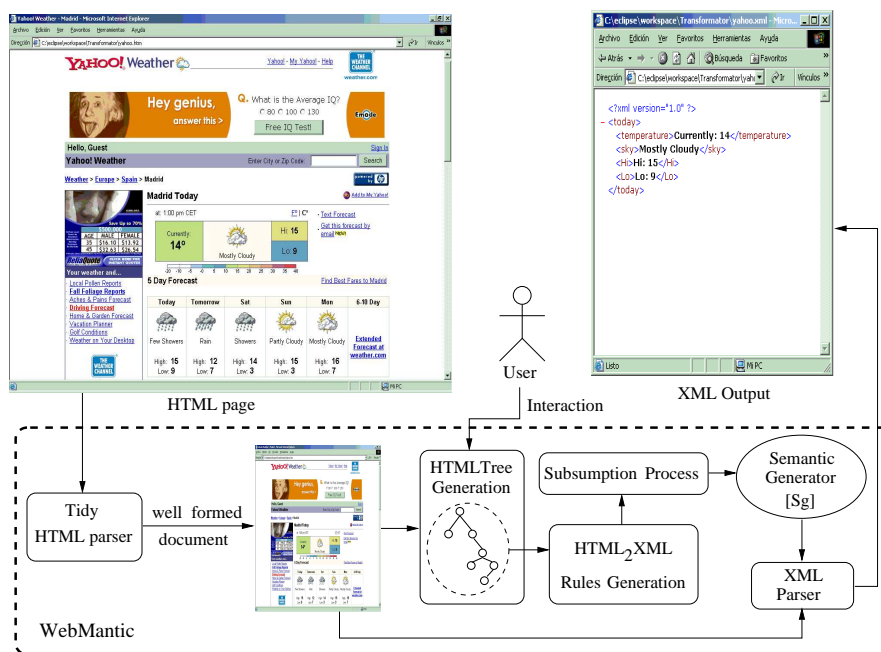


Figure 2: The WebMantic architecture.

- *Tidy HTML parser* (<http://tidy.sourceforge.net>). It fixes bad-formed HTML documents (unbalanced tags, etc.). The HTML Tidy program (HTML parser and pretty printer) has been integrated as the first preprocessing module in WebMantic. It corrects common markup errors and outputs well formed HTML documents (using the traditional XML perspective).
- *Tree generator module*. Once the HTML page is preprocessed by the Tidy parser, a tree representation of the structures stored in the page is built. In this representation any table or list tags generates a node, and the leafs of the tree are: cells for tables (th,td,tr) or items for lists (li,lo). For instance, if we consider the rules from Figure 1 the tree generated is shown in Figure 3. This tree represents a table with a header (TABLE.TR.TH), and the four rows that are to be extracted (TABLE.TR.TD).

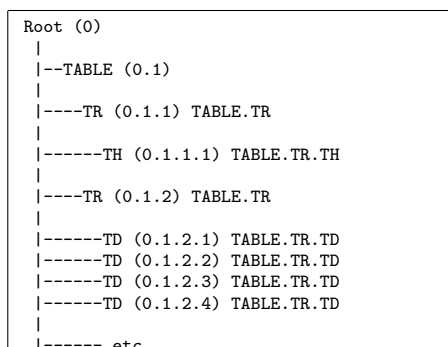


Figure 3: Tree generated from a HTML page

- *HTML<sub>2</sub>XML: Rule generator module.* The tree representation obtained is used by this module to generate a set of rules ( $S_g$ ) that represent the information to be translated and what structures inside the page will be ignored, and the XML tags that have been defined by the user (or loaded directly from a file that stores the XML tags). Figure 4 shows the relation between the HTML tree generated by the previous module and the XML tags.

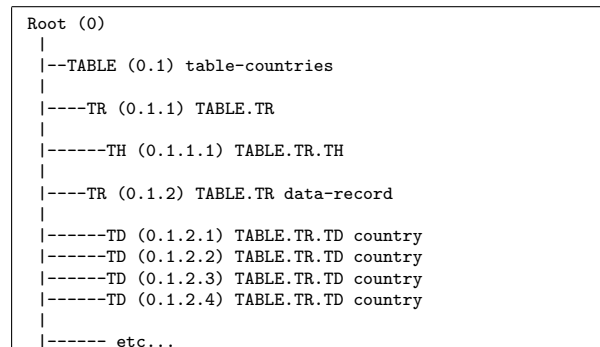


Figure 4: Relation between the HTML tree and the XML-tags provided by the user

Using both, the HTML tree and the XML tags, this module generates the following *HTML<sub>2</sub>XML* rules:

```

<table> IS <table-countries>
  <table.tr> IS <>
    <table.tr.th> IS <>
  <table.tr> IS <data-record>
    <table.tr.td> IS <country>
  <table.tr> IS <data-record>
    <table.tr.td> IS <country>
  <table.tr> IS <data-record>
    <table.tr.td> IS <country>
  <table.tr> IS <data-record>
    <table.tr.td> IS <country>

```

- *Subsumption module.* The previous module generates a rule for each structure to be transformed. However, some of those rules can be generalized if the XML-tag represents the same concept. (i.e. the rules in previous example that represent the concepts of  $\langle data - record \rangle$  and  $\langle country \rangle$ ). The minimum set of rules necessary to transform the document is obtained:

```

<table> IS <table-countries>
  <table.tr> IS void
    <table.tr.th> IS void
  <table.tr> IS <data-record> #4
  <table.tr.td> IS <country> #4

```

- *XML Parser module.* This module receives both, the Semantic Generator obtained in previous module, and the (well formed) HTML document. This module generates the XML document as follows:

- The header of the XML document is automatically written in a file.
- The rules of the Semantic Generator are sequentially executed to preprocess each structure inside the HTML document.

### 3.2 The Semantic Generator Algorithm

The WebMantic application follows the next steps to obtain the  $S_g$ , and finally generates the XML document from a particular HTML page:

1. An HTML page to be translated is provided (by the user) to WebMantic. A preliminary preprocessing of the HTML document is done by the *Tidy* parser, so a well formed HTML page is obtained.
2. Tables and lists are sequentially located inside the page. The HTML document is sequentially processed and a HTML tree is built where every node represents a HTML tag. For each tag inside the structure the related XML-tag is requested from the user (the user must **interact** with the application). If no XML-tag is provided a predefined value is used. If the user does not need a particular list or table, it will be marked as "void". This value is used to remove this structure in the transformation process. Once the HTML document is completely processed a tree structure is generated that represents all the possible structures that can be gathered from this HTML document (or other documents following the same structure).
3. From the HTML tree, a set of *HTML<sub>2</sub>XML* rules are generated using the previous interaction with the user. Therefore, if a particular piece of information is not desired by the user, this rule will be marked as "void".
4. For these rules a subsumption (generalization) process is carried out. This process allows to minimize the number of rules needed to translate the target structures. The new semantic generator  $S'_g$  verify that  $S'_g \subset S_g$ . The algorithm used to minimize the number of semantic generators can be summarized as:
  - 4.1. For each rule that represents a target structure (table or list) in the  $S_g$ . it looks for all the rules with the same header until a different header is found.
  - 4.2. Those rules that have the same header and body are removed.
  - 4.3. This process is repeated for all rules until no more rules can be removed.

After all this processing, a minimum set of *HTML<sub>2</sub>XML* rules is obtained. In other words, the semantic generator  $S_g$  for this page is built. Using this  $S_g$ , WebMantic filters the HTML page (that previously has been preprocessed by Tidy) parsing the marked (desired) information by the user into an XML document. The other structures in the page that have been marked as void, will be ignored in the parsing process. This  $S_g$  can be used for other pages with the same structure (typically, many Web servers provide information in different pages, but identically structured).

### 3.3 Obtaining Semantic Generators for Simple HTML Documents

The following simple example is provided to show how a Semantic Generator is able to provide semantics generating a new XML document from an HTML document. Figure 5 shows both, the HTML output and the code, for a simple Web page which contains two (unordered and ordered) lists of items. Let us suppose that the information to be extracted is stored in the first list and the information stored in the second ones is not desired.

Let us suppose that the user wants to transform the previous HTML document into the next XML structure:

```
<country-list>
  <country>Germany</country>
  <country>Ireland</country>
```

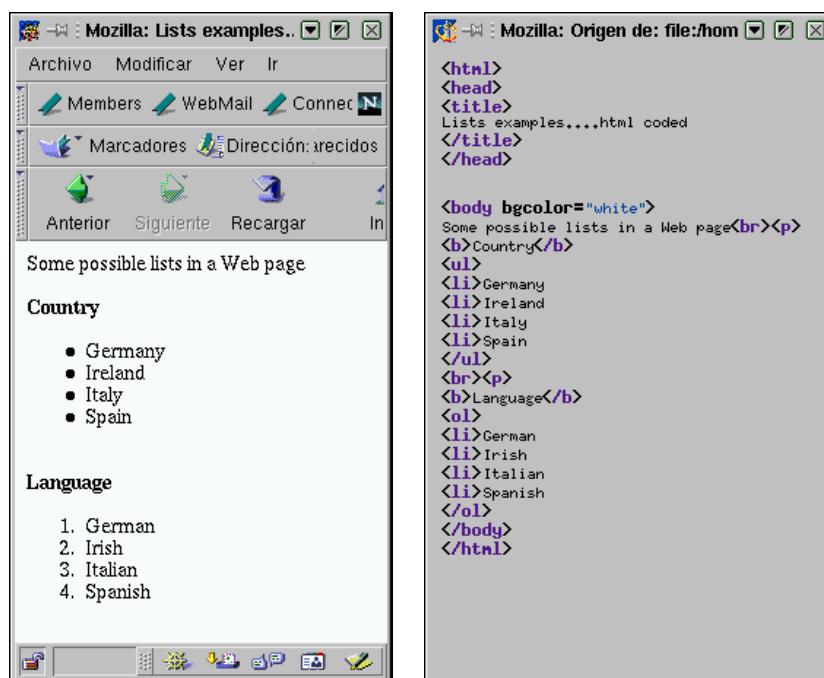


Figure 5: Lists examples to be transformed.

```

<country>Italy</country>
<country>Spain</country>
</country-list>

```

Initially, using the previous HTML document (Figures 5), WebMantic scans sequentially the HTML document and, for every structure (list), it asks the user for their XML tags. The next set of rules will be generated:

```

LISTS: Sg1
<ul> IS <country-list>
  <ul.li> IS <country> #1
  <ul.li> IS <country> #1
  <ul.li> IS <country> #1
  <ul.li> IS <country> #1
<ul> is void #1

```

The  $S_{g1}$  generated is responsible to transform the information. In the example, only the first list will be translated, and the second ones will be removed. The  $S_{g1}$  is built using six  $HTML_2XML$  rules, the rule with the value “void” is used to mark the second list as removable. Once the subsumption algorithm is carried out by WebMantic, the previous set of rules are simplified to:

```

LISTS: Sg1
<ul> IS <list> #1
  <ul.li> IS <country> #4
<ul> is void #1

```

This simple example shows how the subsumption process allows to obtain the minimum number of rules necessary to build a new XML document which only stores the target information.

## 4 The WebMantic Assistant Tool

This section provides a description of the visual interface provided by WebMantic to parse and transform HTML documents, and one application example of the assistant tool in actual Web site.

### 4.1 WebMantic Graphical User Interface

WebMantic provides a simple Graphical User Interface (GUI) that allows the user to analyze any HTML document. The WebMantic is made of three frames: the frame for user interaction; the frame to display the rules; and the third frame implemented using the Clue Web browser<sup>3</sup> (see Figure 6).

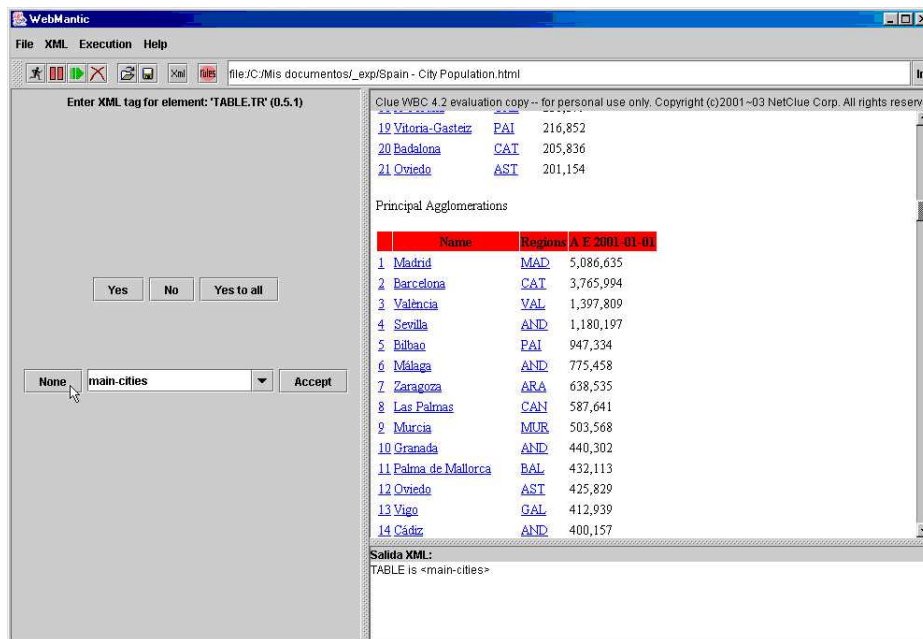


Figure 6: Graphical User Interface provided by WebMantic.

The assistant tool allows to test and execute the rules obtained before the semantic generator is deployed. Finally, the button *XML* in the menu allows to generate the XML file from the HTML example page using the rules generated, and the button *Rules* allows to store the rules in a file.

### 4.2 Applying WebMantic in Actual Web Sites

Here, as an example we will apply our approach to actual Web page. We have studied three different kind of representative Web sites, namely population Web site, transport Web site and weather Web

<sup>3</sup>Actually we are using the Clue WBC 4.2 version, <http://www.netcluesoft.com/>

site, that provide useful information in a semi-structured way. The detailed experimental results are described as a separate section. From all different transport Web sites, Iberia Airline (<http://www.iberia.com>) was selected as a representative company that provides a flight Web service:

#### 4.2.1 Iberia Airflight Web Site

This Web site corresponds to an actual transport company. These kind of sites typically answer user queries for a particular transport service. The server answers with a Web page, that contains the requested information. Figures 7 (a) and (b), show both a user's request for a Madrid-London flight (a), and the XML desired target (b)<sup>4</sup>. In this example the table that stores the data about the possible flights will be the target information.

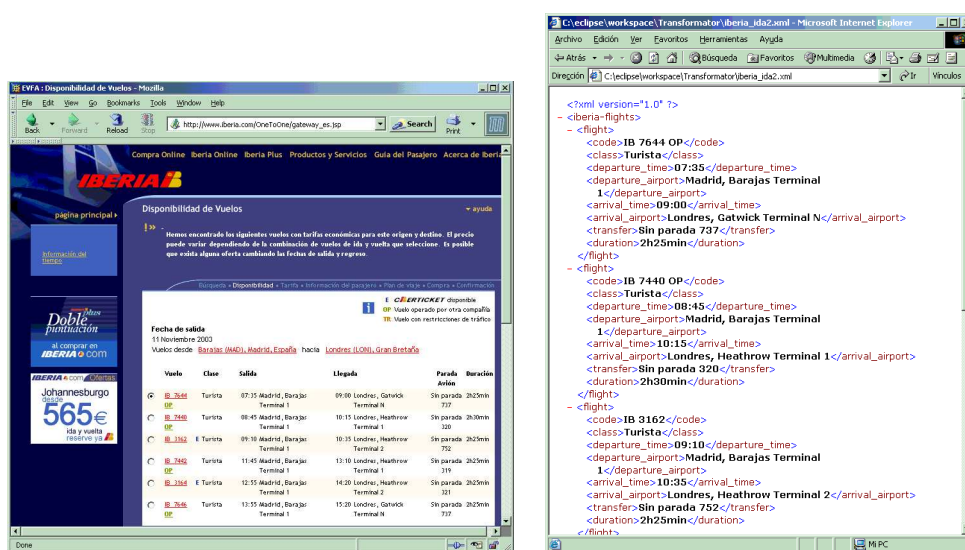


Figure 7: HTML example from the Iberia Airflight Web site (a), and the XML target transformation (b).

From the previous example, we can see some of the  $HTML_2XML$  generated rules to obtain the information:

```

...
<TABLE . TR . TD . TABLE . TR . TD . TABLE . TR . TD .
TABLE . TR . TD . TABLE . TR . TD . TABLE . TR . TD . TABLE>
IS <iberia-flights> #1
<TABLE . TR . TD . TABLE . TR . TD . TABLE . TR . TD .
TABLE . TR . TD . TABLE . TR . TD . TABLE . TR . TD . TABLE . TR>
IS void #1
<TABLE . TR . TD . TABLE . TR . TD . TABLE . TR . TD .
TABLE . TR . TD . TABLE . TR . TD . TABLE . TR . TD . TABLE . TR . TD>
IS void #1
<TABLE . TR . TD . TABLE . TR . TD . TABLE . TR . TD .
TABLE . TR . TD . TABLE . TR . TD . TABLE . TR . TD . TABLE . TR>
IS <flight> #6
    
```

<sup>4</sup>This XML output corresponds to the HTML processed document by WebMantic

```

<TABLE . TR . TD . TABLE . TR . TD . TABLE . TR . TD .
TABLE . TR . TD . TABLE . TR . TD . TABLE . TR . TD . TABLE . TR . TD >
IS void #6
<TABLE . TR . TD . TABLE . TR . TD . TABLE . TR . TD .
TABLE . TR . TD . TABLE . TR . TD . TABLE . TR . TD . TABLE . TR . TD >
IS <code> #6
etc...

```

The rules that have a void value, represent the "radio button" that appears in the HTML code and that do not provide information (these items are used by the user to select a particular flight). This Web example is specially complex. The difficulty to retrieve the table information appears because the Iberia server uses seven nested tables to show the results. The utilization of several nested tables are used in many Web sites to format displayed data.

## 5 Integration of Semantic Generators into an Information Agent

The work shown in this paper stems from the authors experiences in Multi-Agent Systems (MAS) applied to Web domains [6]. Our goal is to solve some problems that usually appear in wrapper agents that extract information from Web sites. These agents are hard to maintain due to the volatility of the Web sources (data and/or format of the HTML documents change frequently). The underlying idea, is to isolate into two independent modules the wrapper ability of those agents:

- *Wrapper Module.* This module allows to translate into an standard format (i.e. XML) the (desired) portions of the HTML document given by a particular Web Site.
- *Extraction Module.* The extraction module will be used to extract the target information from a XML document. This module will use XML technologies, like DOM (Document Object Model: <http://www.w3.org/DOM/>), or XPath (XML Path Language: <http://www.w3.org/TR/xpath>) to extract the information.

Figure 8 (a) shows a representation for a traditional Wrapper (information) agent, and how this type of agent receives a question (from other agents or users) and taking the information stored in the request, is used to access, retrieve, and extract automatically the information from one or several known Web sites. Figure 8 (b) represents the same information agent when its wrapper ability is divided into the two previous modules.

In order to illustrate how the ideas in this paper could be used, we have integrated the Semantic Generators into several information agents that belongs to a Multi-Agent framework named MAPWEB. MAPWEB [6] (**M**ulti-**A**gent **P**lanning in the **W**eb) is a general MAS information gathering architecture that integrates planning and Web information gathering agents.

## 6 Experimental Results

We have carried out an experimental evaluation of WebMantic in order to know how it performs in practice. Several information agents have been implemented and the Semantic Generators necessary to wrap three actual Web sites have been generated. One of such Web sites was described in Section 4.2. We have selected three different types of Web sites:

- **Population Web site.** Twenty pages about population in different countries were selected from the City Population site.
- **Airflight Web site.** Twenty different consults about flights were obtained using Iberia Airlines Web site.

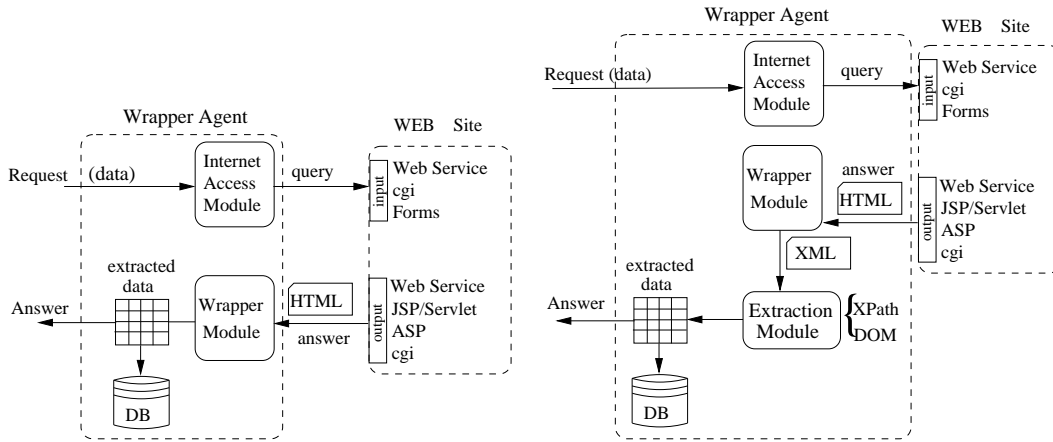


Figure 8: (a) Standard wrapper process. (b) Wrapper process divided in two related subprocesses.

Table 1: Experimental evaluation of WebMantic using several Web sites.

Web Site	N. pages	N. Structs	Max. nested	N. of rules	Time $S_g$	Transf.time
Population	20	5.33	1	10.58	0.0755	2.49
Airflight	20	2	8	205.2	0.3107	1.08
Weather	20	9	5.43	310.0	0.3522	0.39

- **Weather Web site.** Twenty consults about the weather in different cities were stored and analysed from Yahoo! Weather site.

The following quantities have been measured:

1. Number of pages tested from each Web site.
2. Number of accessible structures. This parameter is used to show the number of structures stored in the pages that can be translated (and therefore retrieved) by WebMantic.
3. Maximum nested structure. It measures how complex is the most nested structure inside the Web document
4. Average number of  $HTML_2XML$  rules for each Semantic Generator ( $S_g$ ), once the subsumption process has finished. This quantity is related to the two previous ones.
5. Average time (seconds) to generate the  $S_g$  (Time  $S_g$ ), after the user has provided the semantics (i.e. the XML tags).
6. Average time (seconds) to translate from HTML to XML for the set of training pages (transformation time).

From the experimental results shown in Table 1, several conclusions can be drawn:

- WebMantic is able to look automatically for information inside real, highly nested, visualization oriented HTML documents.

- The time necessary to build the  $HTML_2XML$  rules is small (about 0.35 seconds in the worst case). The number of rules is proportional to the nested level of the tables. There is one case where this seems to be wrong. The Airflight and Weather sites have depth 8 and 205.2 rules vs. depth 5.43 and 310.0 rules. This is due to the actual information visualization for each Web site.
- The time to translate the HTML documents is not very high. The worst case (2.49 seconds) is for the population pages, even though only ten rules are involved.

Finally, both the hand-made wrappers and the wrappers implemented through the utilization of the Semantic Generators have been compared using an information agent. Table 2 shows the experimental results of this comparison.

Table 2: Comparison between hand-made and WebMantic wrappers.

WebAgent	hand-made (sec.)	WebMantic- $S_g$ (sec.)
Population	2.12	2.71
Iberia	1.08	1.32
Yahoo-Weather	0.34	0.56

The experimental results from Table 2 show that there is some overload in processing when Semantic Generators are used, but the differences are not significative, at least compared to other delays due to collapsed networks, overloaded servers, etc. In any case, Semantic Generators' processing time can be improved if some parsing processes like the Tidy process are not necessary.

## 7 Conclusions

This paper has presented an approach to wrapping semi-structured Web pages, the interaction with the user allows to build up XML-based wrappers that finally can be integrated into information agents. The main contribution of this paper is to define a technique which is able to provide a semantic representation (using XML-tags) to semi-structured (tables and lists) Web pages through a set of rules (encapsulated in a Semantic Generator). Our approach has two important advantages. On one hand, it is possible to use other XML related technologies like XSLT, SAX, XPath, or XQuery, to retrieve, gather, and manage the information (once it has been translated into XML). On the other hand, our wrappers provide robustness and adaptability to those agents that are involved in extraction or gathering processes. Currently, our technique uses the semantics provided by the user (or from a file). However we think that it will be simple to extend our approach to provide semantics using standardized ontologies and also other related techniques like Machine Learning (e.g. Genetic Algorithms [3], or inductive supervised algorithms [19, 21]) could be used to improve the  $S_g$  generation process.

## Acknowledgements

This work has been funded by following CICYT projects: TSI2006-12085, TIN2007-65989, TIN2007-64718 and the Universidad de Alcalá (research project UAH PI2005/084). The authors wish to thank the anonymous referees for their careful reading of the manuscript and their fruitful comments and suggestions.

## References

- [1] The World Wide Consortium (W3C). Extensible markup language (XML). <http://www.w3.org>, 1997.
- [2] Serge Abiteboul, Dan Suciu, and Peter Buneman. *Data on the Web : From Relations to Semistructured Data and XML*. Morgan Kaufmann, Los Altos, CA, 1999.
- [3] David F. Barrero, David Camacho, and María D. R-Moreno. *Data Mining and Multiagent Integration*, chapter Automatic Web Data Extraction based on Genetic Algorithms and Regular Expressions, page To appear. Springer-Verlag, 2009.
- [4] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernandez, Michael Kay, Jonathan Robie, and Jerome Simeon. *XML Path Language (XPath) 2.0*. W3C Recommendation, January 2007.
- [5] Robin D. Burke, Kristian J. Hammond, and E. Cooper. Knowledge-based information retrieval from semi-structured text. In *In AAAI Workshop on Internet-based Information Systems*, pages 9–15. AAAI, 1996.
- [6] David Camacho, Ricardo Aler, Daniel Borrajo, and José M. Molina. A multi-agent architecture for intelligent gathering systems. *AI Communications. The European Journal on Artificial Intelligence. Ed. by IOS Press.*, 18(1):15–32, 2005.
- [7] David Camacho, Ricardo Aler, and Juan Cuadrado. *Data Warehousing and Mining: Concepts, Methodologies, Tools and Applications*, chapter 2.3: Rule-Based Parsing for Web Data Extraction, pages 469–484. Information Science Reference-IGI Global, 2008.
- [8] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 109–118, 2001.
- [9] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *Internet Computing, IEEE*, 6(2):86–93, 2002.
- [10] Line Eikvil. Information extraction from world wide web - a survey. Technical Report 945, Norwegian Computing Center, 1999.
- [11] Yizhong Fan and Susan Gauch. Adaptive agents for information gathering from multiple, distributed information sources. In *Proceedings of 1999 AAAI Symposium on Intelligent Agents in Cyberspace*. Stanford University, March 1999.
- [12] Dayne Freitag. Using grammatical inference to improve precision in information extraction. In *Working Notes of the ICML-97 Workshop on Automata Induction, Grammatical Inference, and Language Acquisition*, 1997.
- [13] Dayne Freitag. Information extraction from html: Application of a general learning approach. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*. AAAI, 1998.
- [14] Jean-Robert Gruser, Louiqa Raschid, M. E. Vidal, and Laura Bright. Wrapper generation for web accessible data sources. In *Conference on Cooperative Information Systems*, pages 14–23, 1998.

- [15] J.Y.-J. Hsu and W.-T. Yih. Template-based information mining from html documents. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 256–262. AAAI Press, Menlo Park, CA, 1997.
- [16] Craig A. Knoblock and Joé Luis Ambite. *Agents for Information Gathering*, chapter In Software Agents. J. Bradshaw editor, AAAI/MIT Press, Menlo Park, CA, 1997.
- [17] Stefan Kuhllins and Ross Tredwell. Toolkits for generating wrappers. In Mehmet Aksit, Mira Mezini, and Rainer Unland, editors, *International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World*, pages 184–198. Lecture Notes In Computer Science, Springer-Verlag, 2002.
- [18] Kristina Lerman, Craig A. Knoblock, and Steven Minton. Automatic data extraction from lists and tables in web sources. In *Automatic Text Extraction and Mining workshop (ATEM-01), IJCAI-01*, August 2001.
- [19] Kristina Lerman, Steven Minton, and Craig Knoblock. Wrapper maintenance: A machine learning approach. *Journal of Artificial Intelligence Research*, (18):149–181, 2003.
- [20] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the Twenty-second International Conference on Very Large Databases*, pages 251–262, Bombay, India, 1996. VLDB Endowment, Saratoga, California.
- [21] Ion Muslea, Steve Minton, and Craig Knoblock. A hierarchical approach to wrapper induction. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 190–197, Seattle, WA, USA, 1999. ACM Press.
- [22] Stephen Soderland. Learning to extract text-based information from the world wide web. In *Proceedings of the 3th International Conference on Knowledge Discovery and Data Mining*, 1997.