Designing and Implementing a Wrapper Specification Language for Web Information Sources

Konstantinos Karageorgos[1], Constantinos Valakas[1], Yannis Stavrakas[1,2], and Andreas Polyrakis[1]

[1] Knowledge and Database Laboratory, National Technical University of Athens (NTUA), GR-157 73 , Greece.
[2] Institute of Informatics & Telecommunications, National Centre for Scientific Research "Demokritos", GR-153 10, Greece.

Abstract: In this paper we describe the *Wrapper Template Language* (WTL), a high-level scripting language for defining wrapper templates. WTL has been implemented in the frame of the *Nestor* project, a web-based system that supports the management of resources such as books, proceedings, journals, papers, etc. The role of wrappers within Nestor is to automatically retrieve the "table of contents" for proceedings and journals from the appropriate web pages of the publishers. Instead of implementing a set of source-specific wrappers, we developed a flexible and powerful mechanism for dynamically defining and generating wrapper templates. Although WTL is integrated in Nestor and makes use of the Nestor database, it makes no assumptions on the scheme of the database and the nature of the hosting application. This makes WTL suitable for any kind of application that intends to extract information from web sources and store it in a relational database as a set of tuples.

1 Introduction

The wide adoption of the Web as a means of information publishing and data exchange has led the database community to consider the web as a huge database. New methods and languages have been proposed [1] for querying, re-organizing, and combining information on the Web, that are more powerful and efficient than the information retrieval methods used by search engines. The Web, however, introduces a number of new problems [2], such as the lack of schema and the loose structure of its contents, as well as the distributed and heterogeneous nature of its information sources.

To cope with those problems the database community proposed:

- The *semi-structured* data model [3], that allows irregularities in the structure of data, and considers schema information to be incorporated in data. Examples of semi-structured data are HTML documents, XML documents, RTF, etc.
- The *mediator-wrapper* architecture [4], that uses wrappers as source-specific data extractors, and mediators that combine results from many wrappers to give an integrated answer to a user query.

The mediator-wrapper architecture is a very suitable mechanism for imposing structure and integrating heterogeneous semi-structured data sources on the Web. Projects such as TSIMMIS [5,6], Garlic [7] and Ariadne [8], use wrappers to extract data from sources and create an integrated view that can serve sophisticated user queries. However, since wrappers are tied to specific sources, adding new ones require the development of new wrappers. Moreover, if the structure of a source changes, the corresponding wrapper will fail to retrieve any data. This makes wrappers the most costly and inflexible link in the chain of the mechanism.

An approach that attempts to overcome this problem is to automate the generation of wrappers as much as possible, and develop systems and languages that are able to generate wrappers from high-level specifications. Examples as such systems are TSIMMIS [5,6] and Garlic [7,9].

In this paper, we present *Nestor*, an innovative resource management system for research libraries, and we describe *Wrapper Template Language* (*WTL*), a high-level scripting language for defining wrappers. WTL is used within Nestor in order to automatically extract the contents of various journals and conference proceedings from the web, and insert them into the Nestor database. However, WTL is not restricted to be used within Nestor, as it is general enough to extract any kind of information from semi-structured sources.

2 The NESTOR Project

Nestor is a sophisticated web-based library management system that has been developed in the Knowledge and Database Systems Laboratory (KDBS Lab) at the National Technical University of Athens (NTUA). The purpose of Nestor is to assist the management and utilization of lab resources in a way that promotes the cooperation between members of the lab.

Nestor manages two types of resources: *tangible items* such as books, proceedings, journals, thesis, printed papers or articles, and *intangible items* such as URLs, software, and papers or articles available in electronic format. Tangible items have a material substance, a physical position in the library, and can be borrowed. On the contrary, intangible items have no material substance (and hence cannot be borrowed). Users can book an available item for borrowing, or get into the waiting list for an already borrowed item. Users may also subscribe to any of the journals provided by the lab. The system sends automatically emails to subscribers, to notify them about new journal issues, or borrowed items that have been returned, or even to remind to the borrowers of overdue items.

Nestor supports four categories of users, identified through an authentication process. All four user categories access the system through a conventional web browser. The typical users are *lab member users* that login with an alias and a password and have access to all the features described above. Lab members can also create and manage *virtual libraries*, which are virtual collections of items of any kind, that allow the thematic organization of the resources. Lab members can attach their comments and a number of keywords to any item. These are searchable and available to all users to view. Finally, lab members can insert new intangible items, such as URLs, into the Nestor database. The system automatically annotates such insertions with the alias of the user that performed this action.

The remaining user categories are *guests*, *librarians*, and *administrators*. Guests have a restricted access to Nestor and they can only search for items in a number of different ways and view the information card of an item. The librarian is in charge of inserting and updating tangible item records, as well as handling the borrowings and returns of items. Finally, the administrator is in charge of the operation of the system in general. Administrator’s tasks include managing user accounts, maintaining the database, and creating and updating wrapper templates.

Nestor has the ambition to become something more than a simple library management system. One of the most innovative features of Nestor is the capability to define wrapper templates. In a research facility, such as the KDBS lab, information such as the title of a conference proceedings is usually not enough. People need searchable information about the papers published in the laboratory's journals and proceedings, and even the table of contents of books. Information like that, however, requires enormous data entry effort from the part of the librarian. Fortunately, in many cases, such data exist in the web in the form of HTML pages. A way to automate the data entry process is to create a wrapper for each HTML data source. The wrapper parses the semi-structured source, and extracts information that can then be inserted into the Nestor database.

This approach, however, is not flexible enough. For example, a new journal would require a new wrapper to be developed, which would lead to occasional tampering with the Nestor source code. The same would happen if, for some reason, an existing HTML source changed its structure. The solution implemented in the frame of Nestor was *Wrapper Template Language* (*WTL*), a high-level scripting language especially designed for wrapper specification, that enables the Nestor administrator to easily define or modify wrapper templates.

3 The Wrapper Template Language (WTL)

The main idea of WTL is to implement a language that would allow dynamic generation of wrappers from high-level template code. Furthermore, the use of WTL would relieve programmers from the need to delve into the details of wrapper implementation. WTL programmers do not need to concern themselves with the actual wrapper source code, as this is automatically generated. In this way, repeating similar work is avoided and the burden of developing and maintaining wrappers is greatly reduced.

3.1 The Design of WTL

The basic principle of this language is to form templates that will produce wrappers. The purpose of such a wrapper is to connect to a certain website, retrieve the required data and store it into a database. However, certain difficulties arise that make these goals hard to achieve. The need to specify the address of the website is obvious. Furthermore, knowledge of how the required data are structured in the website -so that the wrapper will know what to look for and retrieve- is also an essential feature of the WTL. Last but not least, the way the produced tuples will be stored in the database needs to be defined as well.

In order to deal with these problems, WTL introduces a number of *reserved variables* and *commands*. Probably, the most important reserved variable of the WTL is the ***$PageUrl*** variable. This variable holds the URL path of the web page that is going to be processed and must be declared at the very beginning of the template code. In order to dynamically define the web page URL, the $PageUrl variable can be assigned a value in a template-like manner, by using a number of special input variables. The ***$FormVar1,...,$FormVarN*** variables are used as arguments to pass values to the wrapper template. The specific variables make the authoring of the template much easier, as the author doesn't have to create multiple wrappers for web pages that have similar data structure. Instead, the same wrapper can be called, choosing each time a different set of values as input variables.

```
$PageUrl=http://www.informatik.uni-trier.de/
            ~ley/db/conf/$FormVar1/$FormVar1$FormVar2.html
```

As an example, for `$FormVar1=vldb` and `$FormVar2=88`, the `$PageUrl` variable becomes as defined below:

```
$PageUrl=http://www.informatik.uni-trier.de/
                               ~ley/db/conf/vldb/vldb88.html
```

The retrieval of the desired data from a website is based on the very tags of the HTML source code. In order to obtain a sequence of characters from the HTML source code, a *pattern*, which contains all the characters that define the position of the data in the document, needs to be defined. In WTL, this pattern consists of the starting and ending characters of the sequence, separated by the wildcard ***par***, that means any character. The requested data is enclosed in parentheses. If part of the starting and ending characters need to be extracted as well, they must also be enclosed in the parentheses.

```
match(@MainDescription,<h2>(par)</h2>)
match(@MainUrl,(<apar</a>))
```

The retrieved data must be stored in variables that are in consistence with the structure of the corresponding table, before being inserted into the database. Using the prefix "`@Main`" followed be any other name, we create a certain type of variables called ***@Main*** variables, that serve this purpose. The `@Main` variables are dynamic arrays and can store more than one value. Once values are stored in a `@Main` variable they can be inserted into the database in one step. If we need to manipulate the retrieved data before storing it to the database, then we can use the ***$Temp*** variables. These are temporary variables, similar to common variables of programming languages, and can be declared using the "`$Temp`" prefix followed by any other name.

```
match($Temp_temp1,>(par)</a>)
```

The data extraction is, as shown above, accomplished through the ***match*** command. This command takes as arguments the variable to which extracted data will be stored, and the pattern which defines the position of that data in the document. The variable can either be a `$Temp` or a `@Main` variable. The manipulation of data stored in `$Temp` variables can be performed using the ***perl*** command, which allows Perl commands, such as assignments, concatenations, etc. to be incorporated in the template code. Data in temporary variables can be stored into main variables using the ***bind*** command.

```
perl $Temp_temp2.=$Temp_temp1.',
bind(@MainAuthors,$Temp_temp2)
```

It is very common to extract data that include HTML links. These links can be in either *absolute* or *relative* format [10]. However if such a link is stored in its relative format it can no longer be used as a link. In a situation like this, the ***convert*** command can be used to convert the relative address of a link stored in the variable, to the absolute one. In case the data to be extracted has a repeated structure through the HTML document, WTL offers a loop feature that allows a group of commands to be executed repeatedly. The ***loop*** command receives as arguments the boundaries of a section of an HTML code, that contains the information that must be processed, and applies the group of commands consisting the loop body through that section. The ***endloop*** indicates the end of the loop body in the template code.

Data extracted and stored in `@Main` variables must be organized in tuples so that they can be inserted into tables. After all data that correspond to a tuple have been retrieved, the ***`increase`*** command is used to increase the index of all the `@Main` variables. Finally, in order to insert the data into the database, the ***`insert`*** command is used. This command inserts the data assigned to the `@Main` variables into a specified table ("`proc_record`" in the case of the example below). The values of each main variable are inserted into the corresponding columns of the table, in the order they are passed to the command.

```
insert(proc_record,@MainTitle,@MainAuthors,@MainPages,
     @MainUrl,@MainDescription)
```

WTL is formally defined in Extended Backus-Naur Form (EBNF). From the description of the language it is easy to realize that WTL commands make no assumptions of the format of the data source. Therefore, WTL can be used to extract information not only from HTML documents, but from any type of semi-structured data source, such as XML documents, or even from unstructured text.

3.2 A Comprehensive Example

In order to give an example, we will refer to the well-known site "DBLP Bibliography" of the University of Trier (`http://www.informatik.uni-trier.de/~ley/db/index.html`). Our purpose is to retrieve certain information from the proceedings of the VLDB conference of the year 1988. The URL for the proceedings is:

```
http://www.informatik.uni-trier.de/~ley/db/conf/vldb/vldb88.html
```

At first, we are interested in extracting the ***Title*** of each paper (e.g. "Enforcing Inclusion Dependencies and Referential Integrity" in the example bellow), as well as the ***Authors*** (e.g. "Marco A. Casanova") and the ***Pages*** of the paper. Also, we want to retrieve the general ***Description*** (e.g. "Database Theory") of the group of papers and of course the corresponding ***URL*** (e.g. "Electronic Edition"). Those elements are going to be stored in the respective table of the Nestor database.

A typical piece of that web page (but with only one paper per *general Description*) is:

Database Theory

- Marco A. Casanova, Luiz Tucherman, Antonio L. Furtado:
  Enforcing Inclusion Dependencies and Referential Integrity. 38-49
  Electronic Edition

Nested Relations and Complex Objects

- Anand Deshpande, Dirk Van Gucht:
  An Implementation for Nested Relational Databases. 76-87
  Electronic Edition

The corresponding HTML source code of the example above is presented below:

```
<h2>Database Theory</h2>
<ul>
```

```
<li><a name="CasanovaTF88" href="../../indices/a-
tree/c/Casanova:Marco_A=.html">Marco A. Casanova</a>, <a
href="../../indices/a-tree/t/Tucherman:Luiz.html">Luiz
Tucherman</a>, <a href="../../indices/a-
tree/f/Furtado:Antonio_L=.html">Antonio L. Furtado</a>:
<br><b>Enforcing Inclusion Dependencies and Referential Integrity.
</b>38-49<br>
<a href="CasanovaTF88.html"><i>Electronic Edition</i></a>
</ul>
<h2>Nested Relations and Complex Objects</h2>
<ul>
<li><a name="DeshpandeG88" href="../../indices/a-
tree/d/Deshpande:Anand.html">Anand Deshpande</a>, <a
href="../../indices/a-tree/g/Gucht:Dirk_Van.html">Dirk Van
Gucht</a>:
<br><b>An Implementation for Nested Relational Databases.
</b>76-87<br>
<a href="DeshpandeG88.html"><i>Electronic Edition</i></a>
</ul>
```

The emphasized HTML tags indicate patterns enclosing the requested information. The following template code assumes one or more papers per *general Description*:

```
1      $PageUrl=http://www.informatik.uni-trier.de/
                                 ~ley/db/conf/vldb/vldb88.html
2      loop(<h2>,</ul>)
3         match(@MainDescription,<h2>(par)</h2>)
4         loop(<li>,</i></a>)
5            perl $Temp_temp2=''
6            loop(<a,</a>[,:])
7               match($Temp_temp1,>(par)</a>)
8               perl $Temp_temp2.=$Temp_temp1.', '
9            endloop()
10           bind(@MainAuthors,$Temp_temp2)
11           match(@MainTitle,<b>(par)</b>)
12           match(@MainPages,^(par)<br>)
13           match(@MainUrl,(<apar</a>))
14           convert(@MainUrl)
15           increase()
16        endloop()
17     endloop()
18     insert(proc_record,@MainTitle,@MainAuthors,@MainPages,
              @MainUrl,@MainDescription)
```

In line 1, we define the URL of the site by assigning a proper value to the `$PageUrl` variable. We then proceed to the main body of the template, which extends from line 2 through 17. In line 2 we use the `loop` command in order to define the limits of the first repeated piece of data, which is from the first **`<h2>`** tag to the **`</ul>`** tag in the HTML code. In the next line, we extract the first type of data we want to retrieve, namely **Description**, using the `match` command.

With the second `loop` command in line 4, we define the second inner loop that contains the rest of the information we need, which actually corresponds to each bullet in the example. The `perl` command in line 4 assigns to the temporary variable `$Temp_temp2` the NULL character. This variable will be used in the next step, where the extraction of the **Authors** is taking place. Since each paper often has more than one authors, we use for another time the `loop` command to retrieve each name and assign it to the temporary variable `$Temp_temp1`. Before proceeding to the next name, we use the `perl` command again to concatenate the new value of the `$Temp_temp1` to the `$Temp_Temp2` variable. So, when the loop has ended, all the names of the authors have been stored as a string in the `$Temp_temp1` variable. Note that extracted author names are not links anymore, as they were in the original HTML page.

In line 9, the `endloop` keyword marks the end of the present loop and in the next line, we assign the value of the `$Temp_temp2` variable to the `@Main` variable for Authors. Furthermore, in the lines 11, 12 and 13 we retrieve the data that is related to the rest of the elements of the proceeding. In line 11 we extract the **Title**, in line 12 the corresponding **Pages** and in line 13 the **URL** of the paper. Because all the URL paths in the source are relative we use the `convert` command in line 14 to convert those URL paths to absolute ones. In line 15, we mark that all the information of the present tuple has been retrieved and that the wrapper can proceed to the next tuple. Note that, if a `@Main` variable has not been assigned a new value since the last time the `increase` command was called, a new call to `increase` will result in using the most recent value of that variable for the new tuple. Such is the case of `@MainDescription` in line 3, if many papers exist per *general Description*. Finally, in line 18 we store in the table "`proc_record`" all the data that have been extracted, by using the `insert` command. The tuples inserted in "`proc_record`" are shown in the table below, and are the output of the wrapper described above for the specific example.

Title	Authors	Pages	URL	Description
Enforcing Inclusion Dependencies and Referential Integrity.	Marco A. Casanova, Luiz Tucherman, Antonio L. Furtado	38-49	*Electronic Edition*	Database Theory
An Implementation for Nested Relational Databases.	Anand Deshpande, Dirk Van Gucht	76-87	*Electronic Edition*	Nested Relations and Complex Objects

The text inserted in the "URL" column of the corresponding database table is actually an HTML extract that corresponds to a link, and is displayed as a link in the table above. For example, the text inserted in the "URL" column of the database table for the first row of the example is:

```
<a href="http://www.informatik.uni-trier.de/~ley/db/vldb/
             CasanovaTF88.html"><i>Electronic Edition</i></a>
```

4 System Architecture

As it was described earlier, Nestor has four categories of users: administrators, librarians, lab members and guests. The system administrator and the librarian are actually the only users that make direct use of wrappers. The administrator has the responsibility for wrapper template authoring and wrapper maintenance. When the librarian realizes that a new type of proceeding or journal is going to be inserted into the Nestor database, notifies the administrator to create a new wrapper suitable for the new items. If a suitable wrapper already exists, the librarian can select it from a list, and launch it. The wrapper performs the data extraction and the librarian can check the results and confirm the data storage. Other user categories can use the data stored by the wrapper mechanism when searching the Nestor database.

4.1 The Wrapper Mechanism

The wrapper template code, created by the administrator, is stored in the hard disk, and informative annotations associated to a template are stored in the database table "`templates`". This table (Fig. 1) contains useful information about wrapper templates, such as the name of the template and a short description, the corresponding URL of the data source, and the type of the template. Currently, two types of wrapper templates are used in Nestor, corresponding to proceedings or journals.

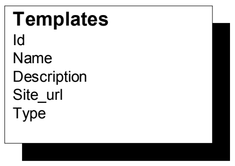

Fig. 1. Storing Wrapper Information

Depending on the type, data is stored in two database tables: "`proc_record`" and "`journal_record`". As depicted in Fig. 2, those tables refer to two other tables, "`proceedings`" and "`journals`" respectively, which contain the general data of the corresponding proceeding or journal. Furthermore, these two tables refer to a more general table, the "`resources`" table, that describes all the items in Nestor.

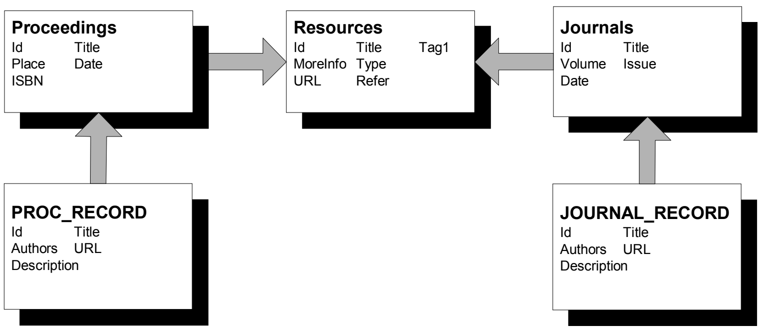

Fig. 2. Incorporating Extracted Data in Nestor

The wrapper mechanism, shown in Fig. 3, is divided into two subsystems, one that handles the definition of templates and wrapper generation, performed by the system administrator, and another that uses the wrappers, performed by the librarian. As far as the first subsystem is concerned, the first step is to edit a wrapper template in the **Template Editor**. The new template goes through the **Wrapper Generator**, where it is checked for syntax errors. If no errors are found, the generator creates the executable for the specific wrapper, which the system administrator can test. The testing procedure is identical to the normal operation of the wrapper, with the difference that the extracted data is not stored, but presented to the administrator on the screen instead. After testing, the wrapper is stored in the hard disk, and its description in the corresponding table of the database.

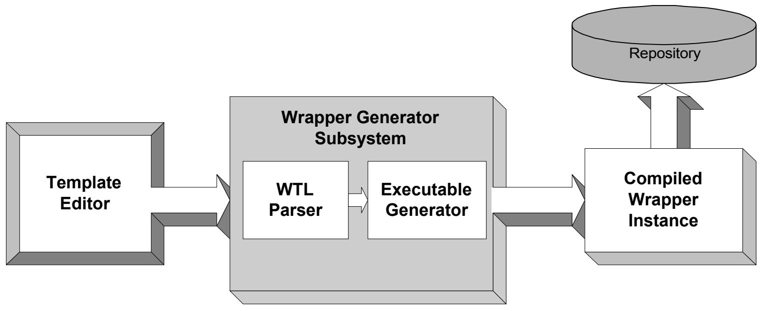

Fig. 3. Wrapper Generator Subsystem

The other subsystem of the wrapper mechanism, displayed in Fig. 4, is used by the librarian. The librarian uses the proper wrapper in order to retrieve the content of a data source (either proceeding or journal). If the corresponding web page is present, it is downloaded to the local server, stored in a temporary location, and processed. The results are presented to the librarian's screen and after his/hers confirmation are stored permanently in the database.

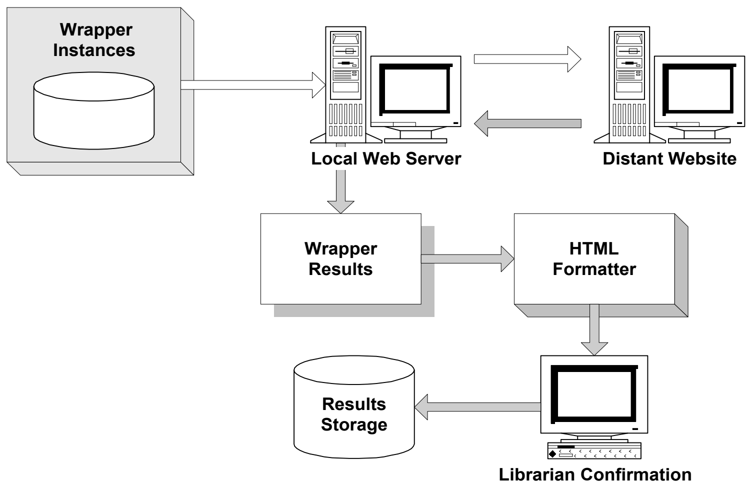

Fig. 4. Wrapper Operation

4.2 Implementation

WTL, as well as most of the Nestor code, was written in Perl 5.0 [11,12]. The DBMS used to store Nestor data is a PostgresSQL. Nestor currently runs on Linux 7.2, and an Apache web server is used to provide the web interface.

Nestor can be reached at `http://www.dblab.ntua.gr/~nestor2`. Guests may search through the entire database, including the proceedings and journal contents that have been stored through wrappers. However, only the Nestor administrator and the librarian of the laboratory can use directly the wrapper mechanism.

5 Conclusions

In this paper we presented Nestor, an innovative management system for research libraries, and WTL, a high-level scripting language for defining wrapper templates. We described the main functionality offered by Nestor, and we explained in detail the simple commands of WTL. We gave a comprehensive example of a wrapper in WTL, and went through the design and implementation of the language. Finally, we described how the wrapper mechanism fits in the Nestor infrastructure, and how it is used from within Nestor.

Being in version 2, Nestor is an evolving system that will undoubtedly reach version 3 in the near future. WTL is a most recent addition to Nestor, and one of its most interesting features. One of our immediate plans is to use WTL with data sources other than HTML, starting with XML documents. We would also like to have the opportunity to use WTL in a frame different than Nestor, and test its generality and expressiveness through another application.

References

1. D. Florescu, A. Levy, A. Mendelzon. *"Database Techniques for the World-Wide Web: a Survey"*. SIGMOD Record 27(3), September 1998.
2. Phil Bernstein et al. *"The Asilomar Report on Database Research"*. SIGMOD Record 27(4), December 1998.
3. D Suciu. *"An Overview of Semistructured Data"*. SIGART News 29(4):28-38, December 1998.
4. Gio Wiederhold. *Mediators in the Architecture of Future Information Systems*. IEEE Computer, March 1992.
5. Yannis Papakonstantinou, Hector Garcia-Molina, Jeffrey Ullman. *MedMaker: A Meditation System Based on Declarative Specifications.* International Conference on Data Engineering, New Orleans, 1996.
6. Joachim Hammer, Hector Garcia-Molina, Svetlozar Nestorov, Ramana Yenerdi, Marcus Breunig, Vassilis Vassalos. *Template-Based Wrappers in the TSIMMIS System*. Proceedings of the 26th SIGMOD International Conference on Management of Data, Tucson Arizona, May 1997.
7. M. J. Carey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. W. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J. H. Williams, E. L. Wimmers. *Towards Heterogeneous Multimedia Information Systems: The Garlic Approach*. Proceedings of the Fifth International Workshop on Research Issues in Data Engineering (RIDE): Distributed Object Management.
8. Ion Muslea, Steve Minton, Graig Knoblock. *STALKER: Learning Extraction Rules for Semistructured, Web-based Information Sources*. Proceedings of AAAI-98 Workshop on AI and Information Integration, AAAI Press, Menlo Park, CA,1998.
9. Naveen Ashish and Graig A. Knoblock. *Wrapper Generation for Semi-structured Internet Sources*. SIGMOD Record, Vol. 26, No 4, December 1997.
10. *HTML Documentation*. http://www.w3.org, 2001.
11. Michael Schilli. *A Jumpstart Guide to Programming with Perl 5*. Perl Power, 1999.
12. *Perl Documentation*. http://www.perldoc.com, 2001.