Data Warehousing and Mining: Concepts, Methodologies, Tools, and Applications

John Wang
Montclair State University, USA
ABSTRACT

How to build intelligent robust applications that work with the information stored in the Web is a difficult problem for several reasons which arise from the essential nature of the Web: the information is highly distributed, it is dynamic (both in content and format), it is not usually correctly structured, and the web sources will be unreachable at some times. To build robust and adaptable web systems, it is necessary to provide a standard representation for the information (i.e., using languages such as XML and ontologies to represent the semantics of the stored knowledge). However, this is actually a research field and usually most web sources do not provide their information in a structured way.

This chapter analyzes a new approach that allows us to build robust and adaptable web systems by using a multi-agent approach. Several problems, including how to retrieve, extract, and manage the stored information from web sources, are analyzed from an agent perspective. Two difficult problems will be addressed in this chapter: designing a general architecture to deal with the problem of managing web information sources; and how these agents could work semiautomatically, adapting their behaviors to the dynamic conditions of the electronic sources.

To achieve the first goal, a generic web-based multi-agent system (MAS) will be proposed, and will be applied in a specific problem to retrieve and manage information from electronic newspapers. To partially solve the problem of retrieving and extracting web information, a semiautomatic
web parser will be designed and deployed like a reusable software component. This parser uses two sets of rules to adapt the behavior of the web agent to possible changes in the web sources. The first one is used to define the knowledge to be extracted from the HTML pages; the second one represents the final structure to store the retrieved knowledge. Using this parser, a specific web-based multi-agent system will be implemented.

INTRODUCTION

The World Wide Web (Web) is an interesting and growing environment for different research fields, e.g., Agents and multi-agent systems (see Balabanovic et al., 1995; Knoblock et al., 2000), Information Retrieval (Baeza-Yates & Ribeiro-Neto, 1999; Jones & Willett, 1997), Software Engineering (Petrie, 1996), etc. Over the past two decades, the evolution of the Web, and especially the stored information that can be obtained from the connected electronic sources, have led to an explosion of system development and research efforts.

However, the success of the Web could be its main pitfall: the enormous growth of the information stored creates so many problems that building and maintaining a web application is difficult. Actually, there is increasing interest in building systems which could reuse the information stored in the Web (Fan & Gauch, 1999). To build these systems, several problems need to be analyzed and solved, i.e. how to retrieve, extract and reuse the stored information.

Information extraction (see Freitag, 1998; Kushmerick et al., 1997) is a complex problem because many of the electronic sources connected in the Web do not provide their information in a standardized way. So, it will be necessary to use several types of specialized agents (or any other type of applications) to retrieve and extract the stored knowledge from the HTML pages. Once this knowledge is extracted, it could be used by the other agents.

Several solutions for information extraction have been proposed. Some of the most popular solutions, which have actually been implemented, are related to the Semantic Web (Berners-Lee et al., 2001). Others use XML-based specifications (Bremer & Gertz, 2002) and ontologies (Gruber, 1993) to represent, in a coherent way, the information stored in the Web. In the near future, this approach will provide the possibility of building robust distributed web applications. However, the Semantic Web is still evolving. So, if we wish to build an application that could reuse the information, we need to use other approaches that allow the system to extract the information.

The Wrapper approach (Sahuguet & Azavant, 1999) is one of the most widely used. It uses wrappers (see Sahuguet & Azavant, 2001; Serafini & Ghidini, 2000) which allow access to the Web as a relational database (see Ashish & Knoblock, 1997; Camacho et al., 2002c; Fan & Gauch, 1999). Building those wrappers may be a complex task because, when the information source changes, it is necessary to reprogram the wrappers as well. Several toolkits, including W4F (Sahuguet & Azavant, 2001) and WrapperBuilder (Ashish & Knoblock, 1997), have been deployed to help engineers build and maintain wrappers.

The main goal of this work is to search for mechanisms that allow for the design and implementation of robust and adaptable multi-agent web systems. These mechanisms should also integrate, like a particular skill of some specialized agents (web agents), the ability to automatically filter and extract the available web knowledge. Toward this end, our approach will use a semiautomatic web parser, or simply WebParser, that is deployed as a reusable software component.

The WebParser is used by different web agents, and they can change its behavior by modifying two sets of rules. The first rules are used by the agents to define the knowledge to be extracted from the HTML pages (i.e., different agents can access different sources), and the second set of rules is used to represent the final structure for
store the knowledge that has been retrieved (so that any agent can adapt the extracted knowledge). Finally, this parser will be used as a specific skill in several agents to build a specific multi-agent web system (such as SimpleNews).

**GENERIC MULTI-AGENT WEB ARCHITECTURE**

Several authors have proposed multi-agent approaches in different domains to deal with web information (see Camacho et al., 2002b; Decker et al., 1997; Knoblock et al., 2000), some general conclusions could be summarized from those works to describe a possible generic multi-agent architecture, which could be used to implement adaptable and robust web systems. Figure 1 shows a schematic representation of this architecture. The architecture is built using a three-layer model. The functionality of those layers can be summarized in:

- **User → System Interaction.** This layer usually provides a set of agents that is able to deal with the users. These agents (UserAgents, Interface-Agents, etc.) could use different techniques, such as learning (see Howe & Drellinger, 1997; Lieberman, 1995), to facilitate the communication between the users and the whole system. In the past few years, this interaction has sparked interest in Human-Computer Interaction (Lewerenz, 2000).

- **User Agents → Task Agents.** This layer is usually built by a set of specialized agents which achieve a specific goal. We call these specialized agents Task Agents, although different architectures refer to them as Middle Agents, Execution Agents, Planning or Learning Agents, etc. Several models of MAS require that the control agents necessary for the system to work correctly be in this layer (usually named as ANS agents, AMR, Control Agents, etc.). Characteristics of this layer, i.e., coordination, organization, cooperation, negotiation, etc., are widely

---

**Figure 1. Generic Web multi-agent based architecture**

![Diagram of Generic Web Multi-Agent Architecture](image)
studied (see Nwana, 1996; Rosenschein, 1985; Sycara, 1989).

- **Middle Agents → Web Agents.** This layer involves agents, such as Information Agents, Web Agents, SoftBots, Crawlers (Selberg & Etzioni, 1997), and Spiders (Chen et al., 2001), which specialize in accessing, retrieving and filtering information from the Web. These types of agents retrieve entire pages or specific parts of those pages (usually the information belongs to <meta> tags). These agents could be characterized because they are able to access different web servers, extract some information, filter that information, and finally store the retrieved document. These agents usually use one or more wrappers (see Kushmerick, 2000; Sahuguet & Azavant, 2001; Serafini & Ghidini, 2000) to wrap the web source and retrieve the available information.

The multi-agent approaches that deal with web information have several advantages and disadvantages. They are summarized as follows:

**Advantages of a Multi-Agent Approach:**

- These systems have better adaptability when unexpected problems in web servers occur. It is easy to add new agents specialized in new web sources.
- The software maintenance is usually simpler than the traditional monolithic applications because the whole system can be split into several simple elements.
- These systems are more robust (have a better fault tolerance) because, if some of the agents are down, the whole system could still work.

**Disadvantages of a Multi-Agent Approach:**

- It is more complex to design the system than a single-agent approach because it is necessary to design the different relations between the agents; new problems, such as coordination, control, or organization, need to be performed to obtain a complete operative system.
- These systems could cause new problems, e.g., the coordination or cooperation among the agents that the engineer could need to solve. These new problems arise from the utilization of multi-agent techniques that could be avoided in a monolithic approach.
- The increasing number of elements (agents) involved in achieving the goal set by the user increases the number of communication messages between the agents. The communication process could be a serious obstacle to a good performance by the whole system.

However, if the main goal is to obtain robust, adaptable, and fault-tolerant web-based systems, we believe that the multi-agent based approach is a suitable one, which provides many important advantages in obtaining the desired systems.

**Characteristics to Implement Robust MAS-Web Systems**

From the previous generic MAS architecture, three important aspects for characteristics (related to the layers shown in Figure 1) need to be performed to achieve the desired goal:

1. It is necessary to provide a flexible and user-friendly user agent to adapt the behavior of the system to the needs of the user.
2. The agent and multi-agent model used to implement the final system is a critical aspect. To build the system, it is possible to use several frameworks and toolkits, such as Jade (Bellifemine et al., 1999), JATLite (Petrie, 1996), ZEUS (Collis et al., 1998), etc. These frameworks allow to the engineers to reuse libraries and agent templates to facilitate the design and implementation phases.
The selection of the agent and multi-agent architecture will be an important aspect in the implementation of the system.

3. For any system that uses web sources, the problems of accessing, retrieving, filtering, representing and, finally, reusing this information need to be overcome.

This chapter addresses the latter characteristic. The first two characteristics will not be analyzed. The process of knowledge extraction is difficult, but it is an essential characteristic for any system that needs to solve problems using web knowledge.

WebParser: SEMI-AUTOMATIC WEB KNOWLEDGE EXTRACTION

This section describes our approach to designing flexible and simple methods for information extraction from web sources. We have designed and implemented a parser, named WebParser, which, through the definition of several rules, can extract knowledge from HTML pages. If any changes are produced in the web page, it will only be necessary to redefine the rules to allow the parser to work correctly again. The utilization of rules creates flexibility and adaptability for the web agents that may use this parser.

However, it is necessary to define what kind of knowledge can be extracted and filtered from the available web pages. We will consider that all web pages can be roughly classified into two knowledge categories:

1. **Non-structured knowledge.** The stored information in the page is represented using natural language, so it will be necessary to apply NLP (Natural Language Processing) techniques to allow the information extraction.

2. **Semi-structured knowledge.** It is possible to find, inside the page, a structure (e.g., a table or list) which stores the information by using some kind of marks to delimit the data (e.g., `<table>`, `<table>`, `<ul>`, `</ul>`, `<ol>`, `</ol>`... tags in HTML).

The WebParser proposed is a simple software module, which is specialized in the extraction of knowledge stored in the second kind of pages. Therefore, the knowledge extracted by the parser will be stored in a specific structure inside the web page.

The WebParser Architecture

A parser can be defined as: *A module, library or program that is able to translate an input (usually a text file) into an output with an internal representation.* The main goal of the WebParser is to accept web pages and generate a data-output structure that contains the filtered information.
The WebParser uses as input the HTML page to be filtered along with several sets of rules that must be defined by the engineer to obtain the information.

Figure 2 shows the rule-based architecture for the semiautomatic web knowledge parser. The definition of several rules allows the engineer to modify the behavior of the parser and adapt it in a simple way. These rules will be used by the parser to represent the knowledge to extract, and the output structure to store the knowledge respectively. Two main types of rules must to be defined:

- **HTML-Rules.** These rules define the type of knowledge to extract (tables, lists, etc.), and the position where this knowledge is inside the page.

- **DataOutput-Rules.** This set of rules defines the final data (output) structure that will be generated by the parser when the extraction process ends.

We have used the term “semiautomatic” because, once the engineer defines the two sets of rules to describe the knowledge to be extracted, the rest of the processes are automatic. If the page changes, or if we want to extract other knowledge inside the same page, it will only be necessary to modify those rules. Several limitations and conditions have been considered in the process of designing the parser. These can be summarized as:

1. The WebParser uses a set of predefined rules (special characters) which are used by the parser to preprocess the HTML page. These special characters (e.g.: â, ê, ..., ñ, etc.) that have their HTML representations (as: &aacute; &eacute; .... &ntilde; etc.) are first translated into standard characters (e.g.: a, e, n, etc.) to avoid possible problems in the extraction process.
2. These sets of rules are written by the engineer, and those rules will be stored into text files to facilitate the modification.
3. Only the following types of web pages are actually parsed:
   - Web pages which contain one or more tables (<table>...</table>) can be parsed.
   - Web pages which contain one or more lists. It is possible to extract information from unordered (<ul>...</ul>), ordered (<ol>...</ol>), and definition (<dt>...</dt>) lists.
   - Web pages that contain nested structures built using tables and/or lists.
4. The code and final implementation of the WebParser will be written in the Java language to obtain a portable and reusable software. This implementation decision was made after taking into account that our main goal is to integrate this software into web agents. Actually, Java is a suitable and very popular language used by a large number of researchers and companies to implement their agent-based and web applications.

### Definition of the Sets Rules in the WebParser

From the architecture designed for the WebParser (shown in Figure 2), it is necessary to provide two different rules to extract the information from a given page.

**HTML-Rules.** Although it is possible to define different rules, the WebParser uses a specific HTML-Rule for filtering each page. This rule is used to select what structures will be filtered from the page. These filtering rules have two attributes:

- **Type.** This attribute tells the parser what type of structure will be filtered. Only list and table attributes are allowed.
- **Position.** If the web page stores several structures (tables, lists, etc.), this attribute is used to locate which of those structures are the target of the extraction process. If there are nested structures, we can use the dot (“.”) to locate the exact position of the structure, i.e., struc1.struc2.strucj represents that information stored...
in the $j$-th structure, that is nested with two level depth, will be extracted.

**DataOutput-Rules.** These rules define the output data structure and what knowledge will be extracted from the page. Only one of those rules (as in the HTML-Rules) is used for every page. These rules are built using the following attributes:

- **Data Level.** This attribute shows where the data is located within the structure.
- **Begin-mark/End-mark.** Once the cells that store the data are fixed (using the previous attribute), it is necessary to set the begin and end patterns which are used to enclose the data. For instance, when the data is stored in a table (it will be stored between the tags `<td>` and `</td>`), it is possible to use as begin-mark the symbol `<td>...data...</td>` to show the string that represents the data begins from this symbol (it is possible to use any string to indicate the beginning and ending of the pattern).
- **Attribute-name.** Once the information is selected, it is necessary to provide the name of the attributes that will be associated with the retrieved information.
- **Attribute distribution.** This attribute shows the attributes-name when the structure to be filtered is a table. In this situation, we have a horizontal (Table 2 inside the third structure in Figure 3) or a vertical (Table 3 inside the third structure in the example) distribution in the tables. This attribute could have a null value if the structure does not have any attribute name (e.g., a table with only numerical information). If the structure to filter is a list, the value will be null because no distribution is necessary for the parser (the different items retrieved will be stored in a Java vector).
- **Data types.** The predefined value of any attribute or data extracted is String. However, the parser can extract other types of data such as: integer (int), float (flo), doubles (doub), etc. The WebParser will cast the extracted string into the desired type of data.
- **Data structure.** Finally, it is necessary to provide the final data output structure that the parser will generate. It can be either a vector or a table. It is possible to select a horizontal table (`tableh: the attributes will be put in the first row and the data in the next rows`) or a vertical table (`tablev: the attributes will be put in the first column and the data in the next columns`). If the extracted information is a list, it will be stored in a vector.

Figure 3 shows an example of a simple web page (and its related HTML code) that stores three different structures: a simple unordered list, a table, and a nested structure which combines lists and tables recursively. For instance, if we wish to extract only the second simple table and the ordered list stored in the third structure (that is nested inside into a table, and inside into a list) from the web page, it will only be necessary to define the rules (HTML and DataOutput) shown in Figure 4. The attributes shown in these rules are used by the parser to:

- **HTML-Rule (a) describes that the structure to extract is a table, and that it is the second (position = 2) structure stored in the page. DataOutput-Rule (a) shows that the data is in the second cell of the table, and that the `<td>` tag is used as begin-end pattern. The names of the attributes are provided in that order to the parser. So, if the distribution of the attributes in the table is horizontal/vertical (`distrib=hv`), we will first indicate the name of the attributes in the rows `{att1,1+att1,2+att1,3}` and then name the attributes in the columns `{att2,1+att3,1}`). The `data type` to retrieve will be `String` values and, finally, the WebParser will generate a table (`data struc= tablehv`) to store the retrieved data.
• **HTML-Rule (b)** describes that the structure to extract is a *list* which is stored inside the third structure (*position*=3). **DataOutput-Rule (b)** shows that the data is stored in the *second* cell of the table, which is stored in third position in the list (*data Level*=3.3.2), and that it possibly uses the *<li>* tag as the begin-end pattern. There are no names associated with the data to retrieve (*attrib*=null), and no distribution of them is necessary. The *data type* to retrieve will be *integer* values and, finally, the WebParser will generate a vector (*data struct=sortlist*) to store the retrieved data.

Actually, the output of the WebParser is a Java object (vector or tables), so this output will be modified by the agent as needed.

### DEPLOYING A WEB-MAS USING THE WebParser

The WebParser has been implemented as a Java re-usable software component. This allows us to:

- Modify, in a flexible way, the behavior of the parser by changing only the rules.
- Integrate this component, like a new skill, in a specialized web agent.

We have deployed a Java application from this WebParser to test different rules retrieved from the selected web pages. This allows the engineer to test the behavior of the parser before it will be integrated as a new skill in the web agent.

We used a simple model to design our web agents. This model allows to us to migrate the designed agent to any predefined architecture that
will ultimately be used to deploy the multi-agent web system. Figure 5 shows the model that defines a basic web agent using the next modules:

- Communication module. This module defines the protocols and languages used by the agents to communicate with other agents in the system [i.e., KQML (Finin et al., 1994) or FIPA-ACL (FIPA.org, 1997)].

- Skill: wrapper. This basic skill must be implemented by every agent that wishes to retrieve information from the Web. This module needs to be able to do the following tasks: access the web source automatically; retrieve the HTML answer; filter the information; and, finally, extract the knowledge.

- Control. This module coordinates and manages all the tasks in the agents.

To correctly integrate the WebParser in the web agent, or to change the actual wrapper skill if the agent is deployed, it will be necessary to adapt this agent’s functionality to the behavior of the software component. To achieve successful migration to the WebParser, it will be necessary to change or modify the following:

- The processes which are used by the agent to access to the information source and to extract the knowledge (Automatic Web Access and Parser modules respectively).

- The answers retrieved by the agent (HTML pages) will be provided as input to the parser (and the rules defined by the engineer).

Once the web agent is correctly designed, the integration of the WebParser only needs to define the two set of rules analyzed in the previous section, and then use the API provided with the WebParser to correctly execute this software module.

**SimpleNews: A MetaSearch System for Electronic News**

SimpleNews (Camacho et al., 2002a) is a meta-search engine that, by means of several specialized and cooperative agents, searches for news in a set of electronic newspapers. SimpleNews uses a very simple topology (as Figure 6 shows), where all of the web agents solve the queries sent by the UserAgent. The motivation for designing and implementing SimpleNews was to obtain a...
Rule-Based Parsing for Web Data Extraction

Figure 5. Architecture for a Web agent

![Architecture for a Web agent](image)

A web system that could be used to evaluate and compare empirically different multi-agent frameworks in the same domain. Actually, SimpleNews has been implemented using the Jade, JATLite, SkeletonAgent (Camacho et al., 2002b), and ZEUS frameworks.

The SimpleNews engine uses a set of specialized agents to retrieve information from a particular electronic newspaper. SimpleNews can retrieve information from the selected electronic sources, filter the different answers from the specialized agents, and show them to the user. As Figure 6 shows, the architecture of SimpleNews can be structured in several interconnected layers:

- **UserAgent Interface.** This agent only provides a simple Graphical User Interface to allow users to make requests for news from the selected electronic papers. SimpleNews uses a UserAgent that provides a simple graphical user interface for making queries, the number of solutions requested, and the agents that will be consulted. The interface used by this agent allows to the user to know: the actual state of the agents (active, suspended, searching or finished) and the messages and contents sent between the agents. Finally, all requests retrieved by the agents are analyzed (only different requests are taken into account) and the UserAgent builds an HTML file, which is subsequently displayed to the user.

- **Control Access Layer.** Jade, JATLite, or any other multi-agent architecture needs to use specific agents to manage, run or control the whole system (AMS, ACC, DF in Jade, or AMR in JATLite). This level represents the set of necessary agents (for the architecture analyzed) that will be used by SimpleNews to work correctly. This layer solves the differences from the two versions of SimpleNews that are implemented (from Jade framework and from JATLite).
Rule-Based Parsing for Web Data Extraction

Figure 6. SimpleNews architecture

- **Web Access Layer**: Finally, this layer represents the specialized web agents that retrieve information from the specific electronic sources in the Web.

The meta-search engine includes a UserAgent and six specialized web agents. The specialized web agents can be classified into the following categories:

1. **Financial Information**: Two web agents have been implemented, and they will specialize in financial newspapers: Expansion (http://www.expansion.es), and CincoDias (http://www.cincodias.es).
2. **Sports information**: Two other web agents specialize in sportive newspapers: Marca (http://www.marca.es) and Futvol.com (http://www.futvol.com).
3. **General information**: Finally, two more web agents have been implemented to retrieve information from generic newspapers: El Pais (http://www.elpais.es) and El Mundo (http://www.elmundo.es).

The selected electronic sources are Spanish to allow a better evaluation of the retrieval process.

It is difficult to evaluate the performance of a particular web agent when using a query in a different language. Another reason is that most of those sources are widely used in Spain, so the information stored in them should be enough to test an MAS built with those web agents.

From all the possible available versions of SimpleNews, we selected the Jade 2.4 version for several reasons:

- This framework provides an excellent API, and it is easy to change or modify some Java modules of the agents.
- The performance evaluation shows us that the system has a good fault tolerance.
- It is a multi-agent framework widely used in this research field. So, more researchers can analyze the possible advantages of integrating this module into their agent-based or web-based applications.

**WebParser Integration into SimpleNews**

This section provides a practical example which shows how the WebParser could be integrated into several web agents that belong to a deployed
Figure 7. HTML and DataOutput-Rules to extract the headlines from the Web page request

- HTML rule:
  type= table
  position= 1

- DataOutput rule:
  data Level= 1.2
  begin-mark= <td><b>
  end-mark= </b></td>
  attrib= {null}
  distrib= (null)
  data type= (str)
  data struc= sortlist

MAS (SimpleNews). The following steps must be taken by the engineer to replace the actual skill (wrapper) in the selected agent:

1. Analyze the actual wrapper used by the web agent. Then, identify which modules (or classes) are responsible for the extraction of the information.
2. Analyze the web source. It will be necessary to generate the set of rules to extract the information and generate the information in the appropriate format. The WebParser provides a simple Java object for storing the extracted information. If a more sophisticated structure is necessary, the engineer may need to program a method that translates these objects into the internal representation of the agent.
3. Change the actual wrapper module used by the WebParser and use the tested rules as input to the parser.
4. Test the web agent. If the integration is successful, the behavior does not change in any of the possible situations (information not found, server down, etc.) managed by the agent.

For instance, assume that we want to integrate the WebParser into the specialized web agent www-ElPais that belongs to SimpleNews. The method of achieving the previous steps is outlined below:

1. to the one shown in Figure 5. The wrapping process is achieved by several Java classes that belong to a specialized package (agent wrapper).
2. The web source is shown in Figure 8. The figure on the left shows the answer (using the query: “bush”) to the search engine used by this electronic newspaper. The figure on the right shows the HTML request. It is interesting to see how the information is stored in a nested table (our system only retrieves news headlines). Figure 7 shows the HTML and DataOutput rules necessary to extract the information.
3. Once the previous rules have been tested (using several pages retrieved from the information sources) and the different situations have been considered, the classes or package identified in the first step are changed by the WebParser and the related rules.
4. Finally, the web agent is tested with some test that has been used previously, and the results are compared.

These two rules will be stored in two different files, which will be used by the WebParser when the wrapping skill of the agent is used to extract the knowledge in the source request. The final integration of the WebParser will be achieved by the engineer through exchanging
the actual Java classes in the agent for a simple method invocation with several parameters (like the name of the rules and the page to be filtered).

**Verification and Test**

This section shows how complex it is to integrate the WebParser into a deployed system. So, previous processes were repeated for every agent in the selected version of SimpleNews. This version uses six web agents, specialized in three type of news. Two groups of three different agents were made and were modified by different programmers. We have evaluated seven phases:

1. **Architecture Analyses.** This stage is used by the engineer to study and analyze the architecture of the implemented MAS.

2. **WebParser Analyses.** It is necessary to study the API (http://scalab.uc3m.es/~agente/Projects/WebParser/API) provided by the WebParser to correctly integrate the new software module.

3. **WebSource Analyses.** The possible answers and requests from the web source are analyzed.

4. **Generate/Test Rules.** The HTML and DataOutput Rules are generated by the engineer. Using the WebParser application, the engineer tests the rules, using as examples some of the possible HTML requests from the web source.

5. **Change Skill.** When the rules work correctly, the WebParser is changed for the actual Java classes.

6. **Test Agent.** The agent with the integrated parser is tested in several possible situations.

7. **Test Multi-Agent System.** Finally, all the web agents are tested. If the integration process is successful, then the new system should not present any differences.

Table 1 shows the average measures obtained by two students who used the deployed system to change the skill of all the web agents. Each programmer modified three agents (one of each type), and the table shows the average effort for
Rule-Based Parsing for Web Data Extraction

Table 1. Average time (hours) for each integration phase

<table>
<thead>
<tr>
<th>Phase</th>
<th>General Information</th>
<th>Financial Information</th>
<th>Sportive Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyze Simple News</td>
<td>34.2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Analyze WebParser</td>
<td>13.4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Analyze Web Source</td>
<td>1.6</td>
<td>0.4</td>
<td>0.7</td>
</tr>
<tr>
<td>Generate/Test Rules</td>
<td>1</td>
<td>0.2</td>
<td>0.15</td>
</tr>
<tr>
<td>Change agent Skill</td>
<td>4.3</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>Test Web agent</td>
<td>0.9</td>
<td>0.5</td>
<td>0.3</td>
</tr>
<tr>
<td>Test MAS</td>
<td>2</td>
<td>0.6</td>
<td>0.3</td>
</tr>
</tbody>
</table>

each integration phase. It is important to note that the order of modifying the different agents is shown in the table; the General Information agents (www-Elpais, www-ElMundo) were modified first. From this table, it is possible to show when the multi-agent system is analyzed (and when the functionality and software modules of the agents are properly understood by the engineers). Changing and modifying the actual wrapping skill in the agents only required a few hours to adapt it in the first agent. When this process is successfully implemented, the next agents only need about one hour to build the rules, to change the skill and to test the new agents. This average time is measured over deployed agents (so it is necessary to change the implemented modules). If we are building new agents from scratch or using an MAS framework, it is not necessary to implement the Change Skill phase, so the implementation of a wrapper agent could take about 30 minutes (the wrapper skill). This means that the time and effort to program (and to reprogram these classes when the sources change) the agent is highly reduced.

- Our approach tries to obtain a reusable and portable software that could be used by different web agents to extract knowledge from specific web sources.
- The utilization of rules provides two advantages: the flexible modification of the parser behavior when the source changes; and easy reutilization of the well-tested rules for similar web sources.
- Once the API of the WebParser is analyzed by the programmer, it is easy to use it as a new skill module inside the web agent. This could improve the implementation of web-based multi-agent systems and gathering systems.

Currently, we have implemented an initial version of the WebParser, and have integrated it into several web agents that belong to a simple multi-agent web system.

However, there are several important points that will be addressed to obtain a fully portable and reusable software for extracting web knowledge. These points can be summarized as:

- To study the flexibility of the rules that can be defined in the WebParser. Is it possible to extract other types of stored knowledge with this simple representation?
- To study other agent-based and multi-agent technologies and frameworks that actually have been used by different researchers

CONCLUSIONS AND FUTURE WORK

The main contributions of this chapter can be summarized as follows:
Rule-Based Parsing for Web Data Extraction

and companies, including ZEUS (Collis et al., 1998) and JATLite (Petrie, 1996), and to see if it is possible to implement web or wrapper agents which integrate the parser inside the agents implemented with those technologies.

REFERENCES


This work was previously published in Intelligent Agents for Data Mining and Information Retrieval, edited by M. Mohammadian, pp. 65-87, copyright 2004 by IGI Publishing, formerly known as Idea Group Publishing (an imprint of IGI Global).