Building Wrapper Agents for the Deep Web

Vicente Luque Centeno, Luis Sánchez Fernández, Carlos Delgado Kloos,
Peter T. Breuer, and Fernando Paniagua Martín

Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid,
Avda. Universidad, 30, E-28911 Leganés, Madrid, Spain

Abstract. The Web has rapidly expanded not only as the largest human knowledge repository, but also as a way to develop online applications that may be used for many different tasks. Wrapper agents may automate these tasks for the user. However, developing these wrapper agents for automating tasks on the Web is rather expensive, and they usually require a lot of maintenance effort since Web pages are not strongly structured.
This paper introduces two programming languages for automating tasks on the *legacy* Web developing low cost, robust wrapper agents that may navigate the Web emulating browsers and may process data from the Web automating some user's behaviours. These languages are based on formal methods and W3C standards and are suitable for the legacy *deep Web*, automating tasks like data mining or information integration from different sources. A platform providing execution support to agents developed in these languages has been provided.

1 Introduction

In order to automate user's navigation tasks on the Web, agents that navigate through Web pages can be used. An agent can integrate data gathered from heterogeneous Web sources and process them properly by just emulating the behaviour of a user behind a browser. With such agents, less user interactivity is required during navigation, larger amounts of Web data can be managed and complex data processing can be performed without overwhelming the user during the navigation process.

Robots like [2,1,18,3] may be used for some simple specific tasks. However, these *crawlers* are oriented to a specific task, and can not be used to navigate through the *deep Web*, i.e., the Web which is dynamically built according to user's requests by querying server side databases and other online applications. Agents being able to navigate through the *deep Web* in order to perform a specific task must properly automate, not only the behaviour provided by the user to the browser, i.e., selecting which links are to be followed, which forms are to be submitted, how form fields must be filled in and when to stop navigation, but also other kind of actions not so well directly supported by browsers, like relevant data extraction, and data processing.

However, developing these agents is rather expensive for many reasons.

J.M. Cueva Lovelle et al. (Eds.): ICWE 2003, LNCS 2722, pp. 58–67, 2003.

- Web pages usually have similar markup style for similar data within the same source. This can be used to build data extraction rules that can extract relevant data embedded within retrieved HTML pages. However, Web pages are visualization oriented, irregular semi-structured sources of information [9], so the step of extracting relevant data embedded in HTML pages is not simple.
- Many Web sites have been designed to be used only by specific browsers. These sites may not be properly navigated by other browsers, because accessibility guidelines like [19,17] were not considered by their designers. This implies that agents will have accessibility problems too.
- Agents should be able to automate tasks for the user, not only on experimental simple XML-based environments, but also on the legacy Web, i.e., the visualization-oriented Web which has been built during the last few years on Internet. This involves dealing with large differences across Web sites.

Developing an automated navigation agent for the Web involves a great programming effort. Web data are embedded in visualization oriented, badly structured, heterogeneous and difficult to be processed HTML pages. The ever changing Web may easily stop these programs from running properly, so a great maintenance effort is also needed. As a result, methods for easily developing agents that may robustly navigate through the *deep Web* are needed. These methods should allow programmers to reduce maintenance costs by improving readability and simplicity, allowing the user to encapsulate code in user defined functions and using standards-based, simple but powerful, components that can be easily combined to build easily any extraction rule or data processing behaviour without having to write too many lines of code.

2 Related Work

Most software engineering techniques oriented to the Web have been applied to the development of Web enabled applications running on the *server side*. The purpose of these projects, like [5] is to develop efficient applications able to process user's requests correctly, without providing pages with error messages and maintaining internal data structures in a consistent state.

Some projects, however, have been oriented to facilitate the development of *client side* Web applications. Data extraction from HTML pages has traditionally been implemented with custom-defined languages based on lexical analyzers and parsers, where regular expressions are applied to detect expected parts of a HTML document, treating Web pages just as plain text files. Examples of this approach can be found in [4,6,7,15,11] . In these cases, special care must be taken in order to avoid that extraction rules may be broken by simple internal representation changes at Web pages, like spacings, case sensitiveness or attribute orderings. This approach, yet powerful, since it can be applied to any text file, not only HTML pages, can be considered as a low level solution because the tree-like structure of the page is not considered and rules can't be applied only

to selected nodes. Another problem is that most of the proposed regular expression based rules, appear quite complex to many users (programmers included), since many of these projects define their own extraction language.

Recent projects [16,8] prefer to approach this issue by applying XML techniques, like DOM [21] programming or XSLT [20] transformation rules, to dynamically XML-ized Web pages. XSLT is then used to transform incoming XML-ized Web pages into well formed documents that can be processed further. This is a higher level approach, but something more than simple XML transformation is usually needed to properly process document's information. In fact, better than well formed XML documents, structured data directly processable by user defined computation routines are often preferred. DOM is a much more powerful solution which can solve this issue, but since it has a lower level of abstraction, more lines of code are required to be developed and to be maintained when a simple modification in the markup design of a Web site may appear. XPath has been designed as a language for addressing parts of XML documents. XPath expressions may easily fit in a single line and are well understood by many people. XPath 1.0 is a good option for selecting nodes in a document only if no complex processing need to be specified, so it becomes a good option for XSLT. More complex treatments can be specified with the new XPath 2.0 draft [22], but most of its new capabilities, have not been completely defined yet.

None of the approaches above have adopted a well known formal method as a software engineering technique for simplifying the development and reducing the maintenance costs of these agents. Software engineering techniques to develop robust agents able to navigate in a browser-oriented Web are needed nowadays to reduce navigation costs for many users who find tasks on the Web difficult.

3 Types of Automated Navigation

Manual navigation, i.e., navigation with browsers is based on interaction with the user. Users are required to build navigation paths by selecting which links are to be followed. Users need to visualize data on the screen in order to decide which action should be next performed. Browsers have no user autonomy and are unaware of the user's targets, but they can be used on any Web site. Manual navigation with browsers may be compared to sea-diver's costumes. Users may navigate through the *deep Web*, but they can visit only small depths and explore short navigation paths.

With generic non-site-adapted robots, some simple tasks can be automated. These robots use to visit all pages in a domain in order to perform the same simple predefined action over all those pages, without considering their particular semantics or structure. Robots can be compared to ships. They can be used on any Web site, and explore long navigation paths, but they can only explore its surface, without having deep access to the contents of online databases.

With generic site-adapted agents, more complex tasks can be automated. Generic agents are configured with external parameters in order to perform a specialized task on the Web. The Semantic Web [12] could be considered as

an example. With external declarative metadata, ontologies and semantic rules associated to Web pages, intelligent agents are required to find the results of a user's query within a page in a domain. These agents have no *a priori* navigation path to be followed, so they have to build it dynamically (like users in manual navigation), every time trying to choose the best promising link in a target guided basis. However, these agents can only navigate through metadata-enabled pages, so not all paths can be followed. These metadata guided navigation could be similar to built-in tunnels under the sea. They allow users to inspect the *deep Web*, but only metadata-enabled paths can be followed. Paths are implicitly defined in external metadata and there is no guarantee that the same path ending to a correct solution can be followed again in the future, unless recorded.

With task-customized agents, tasks involving a big amount of data and long navigation paths may be efficiently performed. By pre-programming a site-customized static (but flexible with alternatives) navigation path, specialized code can have programmed access to HTML pages retrieved during navigation, emulating user's behaviour completely, including choices and repetitiveness. Implicit programmed semantics allows data to be properly processed according to the user's wishes. No metadata is needed and well known paths will be followed in a deterministic way, without needing to infer which is the best link to be followed next. However, these programs, usually called *wrapper agents*, are usually difficult to be programmed and maintained, and can only be used on the Web site they were designed to navigate. Wrapper agents could be compared to small non tripulated remote-controlled submarines or *bathyscaphes*. They can navigate farther and deeper than humans. Since they don't need external metadata, they can navigate anywhere, taking into account particular programmer's preferences. However, they are very expensive to build and maintain, and a path to be followed has to be specified according to the Web site. Table 1 summarizes major differences among different kinds of Web navigation agents.

Table 1. Classification of Web navigation agents

	Manual	**Generic non adapted**	**Generic adapted**	**Customized**
Example	Browsers	Robots	Semantic Web Agents	Wrapper agents
Algorithm	User	Simple action	Target guided	Pre-programmed
Semantics	User	None	Declarative metadata	Implicit in program
Path building	User	Cover all pages	Dynamic	Static, but flexible
Maintenance	None	None	Easy (declarative)	Difficult (program)
Deep Web	Yes (small depths)	No	If metadata enabled	Yes (any depth)
Suitable	Any Web	Any Web	If metadata enabled	A particular Web site
Metaphor	Sea-diver's costume	A ship	Tunnel under the sea	Bathyscaphe

4 Steps of Automated Navigation

Navigation through the Web involves several actions that have to be executed sequentially. Basically, these actions can be classified as implicit or explicit. Implicit actions should be transparently performed by the agent's platform execution. Explicit actions need to be explicitly declared in a task specification.

4.1 Implicit Actions

Actions can be considered implicit if the user is not supposed to be aware of them. They are typically transparently managed by browsers without requiring user intervention. Management of HTTP headers and the cookies' database, SSL support, internal reparation of badly constructed pages, query-string coding and execution support for scripting languages (like JavaScript) embedded in HTML pages are some representative examples.

4.2 Explicit Actions

Actions can be considered explicit if the user can be considered responsible of their execution. Browsers can't execute these actions by their own, so they require user intervention in manual navigation. Tasks can be described as a sequence of explicit actions.

Data extraction. Data extraction involves selecting relevant data from Web pages and extracting them for being further processed. Data extraction is the most critical action involved in task's automation on the Web, because it plays a relevant role on every navigation step. Following a link firstly involves selecting which relevant link should be followed and extracting the URL it contains. Submitting a form firstly involves selecting all form fields contained within and filling in them accordingly to their nature and user's aims. Other user defined processings, like data integration, comparisons and other behaviours, also need complex data extractions to be previously performed.

Data extraction is performed visually on the screen by the users, requiring them to visualize several windows and perform some scrolling. In automated navigation, data extraction can be automated by selecting relevant data from semi-structured documents. This is not always easy, and is clearly dependant on the page's structure. This task can be challenged by data extraction rules based on markup structure and its regularity, selecting those data that match some *expected* format. However, this markup structure is usually too much visualization oriented and may be changed with no advice, so these rules need maintenance to keep on working properly. Techniques mentioned on section 2 which can be used for developing these rules are suitable for keeping these efforts reduced. Programming mechanisms for keeping data extraction rules simple, robust and easy to maintain are highly recommended.

Data structuration. Relevant data extracted from a page might be processed several times during navigation, so they should be conveniently stored in a structured repository. In manual navigation, users are usually required to do this by their own. That's why they use to memorize data (usually in the short-term user's memory), write them in a paper or saving them wherever the user might consider. However, user's memory nor papers nor editor's windows are adequate repositories for being accessed during automated navigation. Computer's memory variables or structured files are preferred instead. Web pages might be internally saved as trees or as a sequence of bytes, depending on the platform's capabilities. Tree-like structuration, like [21] is usually preferred because it provides better capabilities for document inspection.

When saving selected parts of a document to specialized repositories, textual data obtained from semistructured documents might be transformed into suitable data types for computation, like numbers, dates, booleans, strings or perhaps, nodes in a tree-like structure. Programming effort is then reduced to the management of these simple and already known repositories and performing these simple transformations, so just a few lines of code can solve this issue. Manual navigation has no support for this, so the user is supposed to apply his own contextual semantics during navigation.

Follow links. Data are usually distributed among several documents that have to be retrieved by following links. This is an easy to do action in both manual and automated navigation. In manual navigation, it is a semiautomated action where the user just has to click in a link with her pointing device. In automated navigation, this involves a single call to a GET primitive, available in several libraries for multiple programming languages. Programming effort is reduced to set expected parameters to this primitive.

Fill in forms. Filling in form fields is a relatively easy to do action in both manual and automated navigation. In manual navigation, it is a semiautomated action where the user just has to set up proper values on every viewable field, guided by the browser. In automated navigation, this usually involves establishing a mapping between form fields and their values. Programming effort can be reduced to code a single sentence for each form field modifying it's default value, according to the values selected by the user and the form field's nature. This is usually performed within an internal structure representing the page's form, usually a list of form fields.

Submit forms. Submitting forms is an easy to do action in both manual and automated navigation. In manual navigation, it is a semiautomated action where the user just has to click on a submit button with her pointing device. In automated navigation, this involves a single call to a GET or POST primitive, available in several libraries for multiple programming languages. Programming effort is reduced to set expected parameters to this primitive.

Data processing. Finally, structured data can be processed according to the task's aims, without being dependant on the Web sites' singularities. Data processing involves data comparisons, aggregation, text processing, sorting data and producing some computed results. Data processing might be rather heterogeneous according to the task which is being performed, so it is usually performed by the user directly with her mental capabilities on manual navigation, but programmable in user defined code in automated navigation. For example, finding the best hotel for the user taking into account several criteria (price, place, distance to relevant places, breakfast included, sauna, spa, ...) can only be performed if the user specifies relevance for those criteria. These user defined computing can be programmed, not only in user defined routines, but also on external programs.

As can be seen in table 2, only three from six actions are supported by browsers during manual navigation. Users need to participate in all of them during manual navigation, but semiautomated support from browsers can only be found at simplest actions. For automated navigation, a single line of code can easily implement also the simplest actions. However, more complex actions need to be programmed completely by the user, so they usually require more lines of code.

Table 2. Explicit actions for Web tasks

Explicit action	Manual Navigation	Automated Navigation
Data extraction	User	Data extraction rules based on markup
Data structuration	User	Structured repositories
Follow links	User + Browser	Call to GET primitive
Fill in forms	User + Browser	Metadata, programmer's code
Submit forms	User + Browser	Call to POST/GET primitives
Data processing	User	User defined routines, external programs, ...

5 XTendedPath: A XPath 2.0 Extension for Extraction and Manipulation of XML Documents

XTendedPath has been designed as a XPath 2.0 [22] extension for extracting data from XML documents. XPath-like data extraction rules may be easily specified in a single line, taking also into account the internal tree-like structure of XML documents. However, XPath only allows addressing whole XML nodes within that tree, their whole text or their attributes, so it results difficult to express other portions of XML documents. XTendedPath provides a set of simple yet powerful primitives which implement most of the main XPath features, like axis, predicates, variables, sequences, quantification, conditional and iteration in

a functional notation. However, some features not considered in XPath 2.0 have been added to provide new functionalities like addressing ranges (parts of a document not fully attached to nodes), manipulating document or user supporting user definable routines.

- XPointer's ranges and points [23] have been included as a XTendedPath data type. These data types are not considered in XPath, but they provide great flexibility to address small pieces of text.
- With document manipulation, XTendedPath expressions may change document's internal structure by adding or removing nodes within the document or changing attribute values within nodes. This is useful to eliminate non desired parts of documents or to fill in forms.
- With user definable routines support, XTendedPath programmers may encapsulate their own reusable lambda expressions [14]. By allowing lambda expressions to be considered as a first order data type, a higher order set of functions may be applied to solve some common tasks by manipulating sequences of data extracted from Web sites.

6 XPlore: A MSC-Based Language for Navigation and Data Processing on the Web

XPlore has been designed as a programming languages suitable for processing data obtained from the Web by XTendedPath expressions and defining explicit navigation paths for automating tasks on the Web. XPlore is based on the well known MSC [10] formal method from the ITU (Telecommunication Union). MSC graphs provide a high level specification of behaviours expected by distributed components exchanging messages. Since their birth in 1992, MSC specification have received important enhancements for improving its ability to describe complex communication systems. XPlore is a Web adapted language based on MSC. With XPlore, the programmer can easily define his own specific navigation paths for traversing links in a pre-programmed way in order to automate well known tasks on the Web. XPlore also provides suitable mechanisms for processing all those data obtained from those visited pages.

6.1 Navigation Paths

Building well known navigation paths is not difficult with XPlore. XPlore provides easy-to-call and detail-enabled primitives for sending HTTP commands like GET or POST to Web servers and receiving their responses, a desirable feature according to table 2. However, GET and POST primitives are not enough for building longer-than-one-step session based navigation paths. Complex navigation paths can be considered as the skeleton of a task on the Web. Navigation paths must be properly specified prior to data processing. For that reason, it is required that these GET and POST primitives may be properly parameterized with URLs which may not be a priori known and which must be obtained during the navigation process. These links can be obtained with XTendedPath rules,

except for the first URL to be visited. Without such rules, the program would not be able to select the proper link to be followed. It is also needed to adjust low level parameters like some HTTP headers for maintaining a session with the server. A fully detailed navigation path is a guarantee that all needed documents can be retrieved from the Web.

6.2 Data Processing

Once relevant data are extracted from visited pages with XTendedPath data extraction rules, extracted data must be properly structured and stored in a convenient repository. XPlore provides typical programming data types for storing atomic values like numbers, booleans, strings or complex structures like lists and objects. XPlore also manages references to nodes in a document. User defined routines written in Java can take those data as arguments and process them according to the programmer's aims. However, simple arithmetical or string data processing may be specified with some pre-defined in-line XPlore primitives, so simple processing need not always be specified in user defined routines. External processes may also be called for invoking legacy programs.

A fully commented example of a XPlore program accessing a Yahoo! mail account can be found at [13].

7 Conclusions

Information retrieval is only the first step towards Web task automation. Once all relevant pages are retrieved, further computing needs to be applied to the data embedded in those pages. Integrating data from Web pages into a program requires giving structure to those data, according to extraction rules which are usually based on markup regularities. These extraction rules, which enclose some semantics for the task of the user, can be easily broken when expressed in general not XML suitable formalisms, like regular expressions. XML related standards like XPath have been defined to solve many issues, but they can poorly be applied to define complex computing. In this paper, a well known formal method (Message Sequence Charts) has been adapted for the construction and maintenance of Web clients that automate the Web and a extraction language has been based on several W3C Recommendations.

Acknowledgements. The work reported in this paper has been partially funded by the project *TEL1999-0207* of the Spanish Ministry of Science and Research.

References

1. Altavista. www.altavista.com.
2. Google. www.google.com.
3. Wget tool. sunsite.auc.dk/pub/infosystems/wget/.

4. A. S. F. Azavant. Building light-weight wrappers for legacy web data-sources using w4f. *International Conference on Very Large Databases (VLDB)*, 1999.
5. J. Baeten, H. van Beek, and S. Mauw. An MSC based representation of DiCons. In *Proceedings of the 10th SDL Forum*, pages 328–347, Copenhagen, Denmark, June 2001.
6. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. D. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *16th Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, Japan, 1994.
7. C. P. David Buttler, Ling Liu. A fully automated extraction system for the world wide web. *IEEE ICDCS-21*, April 16-19 2001.
8. D. Florescu, A. Grunhagen, and D. Kossmann. Xl: An xml programming language for web service specification and composition. In *WWW 11th conference*, 2002.
9. R. Goldman, J. McHugh, and J. Widom. From semistructured data to XML: Migrating the lore data model and query language. In *Workshop on the Web and Databases (WebDB '99)*, pages 25–30, 1999.
10. ITU-T. Recommendation z.120: Message sequence chart (msc). In *Formal description techniques (FDT)*, Geneva, Switzerland, 1997.
11. L. Liu, C. Pu, and W. Han. XWRAP: An XML-enabled wrapper construction system for web information sources. In *ICDE*, pages 611–621, 2000.
12. S. Lu, M. Dong, and F. Fotouhi. The semantic web: Opportunities and challenges for next-generation web applications. *Information Research*, 7(4), 2002. Special Issue on the Semantic Web.
13. V. Luque-Centeno, L. Sanchez-Fernandez, C. Delgado-Kloos, P. T. Breuer, and M. E. Gonzalo-Cabellos. Standards-based languages for programming web navigation assistants. In *5th IEEE International Workshop on Networked Appliances*, pages 70–75, Liverpool, U.K., October 2002.
14. G. Michaelson. An introduction to functional programming through lambda calculus. In *Addison-Wesley, XV, 320 S. - ISBN 0-201-17812-5*, 1988.
15. I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001.
16. J. Myllymaki. Effective web data extraction with standard XML technologies. In *World Wide Web 10th Conference, Hong Kong*, pages 689–696, 2001.
17. W3C. Policies relating to web accessibility. http://www.w3.org/WAI/Policy/.
18. W3C. W3c link checker. validator.w3.org/checklink.
19. W3C. Web content accessibility guidelines 1.0. *W3C Recommendation 5-May-1999*, 1999.
20. W3C. Xsl transformations (xslt) version 1.0. *W3C Recommendation 16 November 1999*, 1999.
21. W3C. Document object model (dom) level 2. *W3C Recommendation 13 November, 2000*, 2000.
22. W3C. Xml path language (xpath) 2.0. *W3C Working Draft 15 November 2002*, 2002.
23. W3C. Xml pointer language (xpointer). *W3C Working Draft 16 August 2002*, 2002.