02-DE

The 6th World Multiconference on Systemics, Cybernetics and Informatics

July 14-18, 2002
Orlando, Florida, USA

PROCEEDINGS

Volume VII

Information Systems Development II

Organized by IIIS
International Institute of Informatics and Systemics

Member of
International Federation of Systems Research IFSR

EDITED BY
Nagib Callaos
John Porter
Naphtali Rishe

Data Extractor Wrapper System*

Dmitriy BERYOZA, Naphtali RISHE, Scott GRAHAM, Ian DE FELIPE
High-performance Database Research Center
School of Computer Science
Florida International University
University Park, Miami, FL 33199, USA

ABSTRACT

We describe a Data Extractor system for retrieving data from Web sites. This system represents Web sites as data tables that can be both integrated in a heterogeneous database framework and serve as data sources for standalone applications. We address issues involved in selection and analysis of Web data sources, and construction of wrappers for them. Data Extractor system design and implementation issues are discussed.

Keywords

Web Information Retrieval, Data Extraction, Web-based Interaction, Distributed / Heterogeneous Database Systems, Scripting Languages

1. INTRODUCTION

The explosive growth of the World Wide Web in recent years has provided users worldwide with unprecedented volumes of information. This wealth of information is, however, underutilized, because mechanisms for accessing it are limited. Users can browse Web pages, search for information and perform a predefined set of transactions on Web sites, but in the majority of cases, generated data is provided only for *visual consumption*. No convenient mechanisms exist for analysis and processing of the found data, because users can only read what is shown in Web pages. There is no way for users to, for example, create complex queries to a travel agent's Web site—for each such query a special program would have to be implemented inside the agent's Web server. Even if the queries that are available meet user's needs there is no way to work with the data that they return. The stock quotes that are available on the financial Web sites most of the time cannot be imported into a spreadsheet for further analysis. The sites that do provide data in an easy-to-process form, such as XML, are still rare. Although this is expected to change in the future, large volumes of data are likely to remain in HTML for quite some time.

In this work we describe a Data Extractor system that provides a mechanism for accessing data scattered across Web sites and using it in a variety of applications, such as database management systems, spreadsheets, and analytical tools.

2. DATA EXTRACTOR

Heterogeneous database integration

The majority of existing methods for accessing data on the Web specialize in extraction and purification of data, and channeling it to external applications and users. Some researchers ([10], [15]) approach Web data extraction as a part of a bigger problem of *heterogeneous database integration*. Such integration would let users access resources of multiple databases of different types via a unified interface. It will also allow them to pose queries over a unified schema of multiple data sources.

In our research we are also investigating ways of integrating data extraction into a heterogeneous database system. We developed MSemODB [21], a heterogeneous database management system, whose general architecture is shown in Figure 1.

The system is using the Semantic Binary Object-oriented Data Model (Sem-ODM) [20] for data representation and the SQL query interface for communication between its components. Sem-ODM combines the simplicity of relational and power of

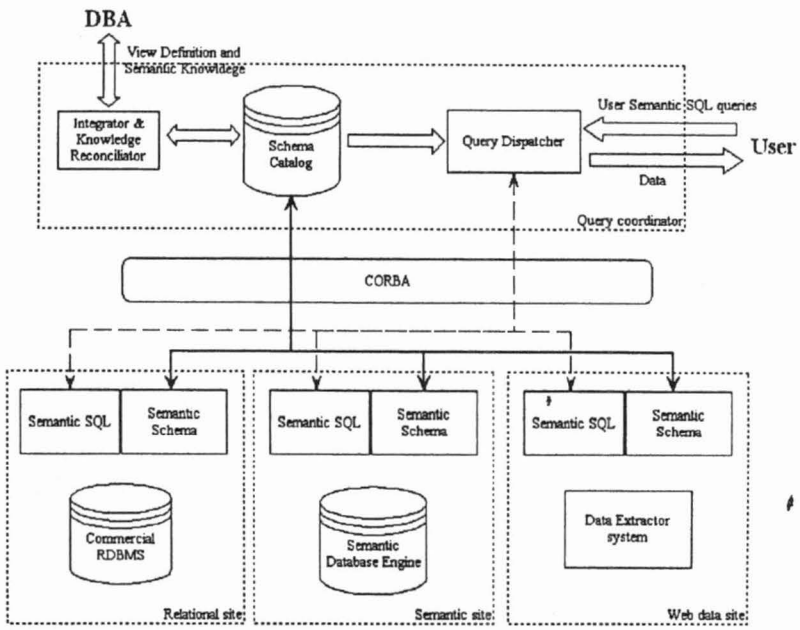

Figure 1 MSemODB architecture

* This research was supported in part by NASA (under grants NAG5-9478, NAGW-4080, NAG5-5095, NAS5-97222, and NAG5-6830), NSF (CDA-9711582, IRI-9409661, HRD-9707076, and ANI-9876409), ONR (N00014-99-1-0952), and the FSGC.

object-oriented data models. A major advantage of this model is its ability to use the standard SQL-92 query language interpreted over Sem-ODM schemas (called Semantic SQL) in a variety of relational and object-oriented databases. This feature makes MSemODB compatible with a number of existing tools developed for standard SQL. The communication between components in the system is done in CORBA, which is an efficient cross-platform and language-independent communication medium.

The main module that controls execution in the system and flow of data is *Query Coordinator*. Its function is to collect database schemas from all member databases and dispatch user queries to the appropriate database sites. It gives a common user interface to all the databases in the system. Through it, users enter queries using SemSQL and view resulting datasets in a single data model. Query Coordinator consists of *Integrator & Knowledge Reconciliator*, *Schema Catalog* and *Query Dispatcher*. *Schema Catalog* collects schemas of individual relational, semantic and Web database sites. *Integrator & Knowledge Reconciliator* coordinates schemas to resolve conflicts. Database administrator can use it to manage and modify Schema Catalog, and to introduce new relations that are not apparent from mere collection of member schemas. *Query Dispatcher* optimizes and executes queries. It decomposes queries posed by user into sub-queries based on the knowledge stored in the Schema Catalog and dispatches these sub-queries to appropriate sites for execution. When the results are available, it assembles them and presents them to the user.

The database sites are exposed in the system through their individual *Semantic SQL* and *Semantic Schema* modules. For the *Relational Site* a special knowledgebase and a reverse-engineering tool facilitate relational-to-semantic schema translation and storage. The majority of translation tasks are performed automatically. The database administrator can step in and make modifications and enhancements to the schema after the automatic conversion is completed. The Semantic SQL module of the Relational Site implements an algorithm which automates conversion from Semantic SQL queries to relational SQL queries. With this functionality, virtually any RDBMS available today can be integrated with MSemODB.

In the *Semantic Site* that wraps around Semantic Databases, integration is far more natural. The Semantic Object-oriented Database engine (Sem-ODB) already has Semantic Schema and Semantic SQL query facilities implemented. This database system is a multi-platform, distributed, fully functional client-server implementation that is suitable both for standard database applications and for large-volume data and spatial data applications.

Web Data Site is built around a Data Extractor system and provides a framework for data extraction from the Web.

Data Extractor architecture

The Semantic SQL and Semantic Schema modules access Data Extractor through a standard interface that allows schema discovery, query execution and data retrieval. Together with these modules the system works as an integral part of an MSemODB system, capable of executing SQL queries and returning result datasets. Using Data Extractor external applications pose queries to Web sites and extract data from them. Data Extractor presents extracted data in two-dimensional tables that can be further processed and returned to the user.

Data Extractor system consists of several components (see Figure 2).

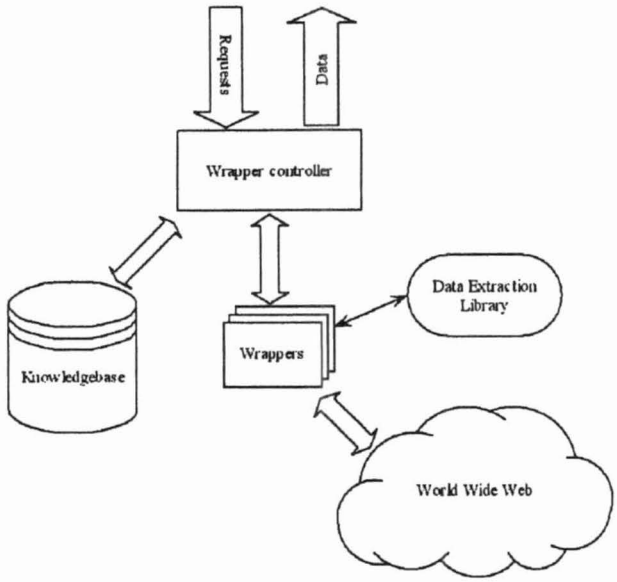

Figure 2 Data Extractor system structure

- *Wrapper Controller*. This is the main component, whose responsibility is to control and coordinate execution of other parts of the system. It is the entry point for communications with the Data Extractor from the outside. It loads, executes, and controls behavior of wrappers and redirects data that they generate to the user. It accesses the knowledgebase to become aware of the configuration and schema changes in the system.
- *Knowledgebase*. This module stores system configuration information. It contains data on what wrappers are available, where they can be loaded from, what parameters are required to execute them, and what kinds of data they generate.
- *Wrappers*. Wrappers are lightweight modules that execute in response to user requests. They extract data from Web sites and return it to the user in an easy-to-process form.
- *Data Extraction Library*. This library contains extensive network access and HTML processing functionality. This functionality simulates behavior of the Web browser, allowing wrappers to traverse Web sites and extract data from HTML pages.

Data Extractor is implemented in several versions: *standalone*, *embedded*, or *mobile*. As a *standalone* server it serves clients through a simple browser-based user interface, executes user queries, and delivers raw or processed data directly to the user. When *embedded* inside another application (as is the case with the MSemODB framework), Data Extractor acts as a data provider for that application. A lightweight *mobile* implementation of Data Extractor is shipped to the client side over the Internet and is executed there on behalf of the user.

3. WRAPPER CONSTRUCTION

Wrappers in Data Extractor execute on behalf of users and extract data from Web sites. Wrappers essentially simulate a user working with the site through the Web browser. They fill out and submit forms, "click" on links, or find data of interest inside of pages. To support this behavior, special functionality was developed that emulates browser interaction with the Web site. It allows us to create and play back a "scenario" of user navigation through the site.

The process of creating a wrapper for a Web data source consists of multiple steps. It includes *source selection, site analysis*, and *wrapper implementation.*

Source selection

Selecting a Web site as a data provider in some applications may become a complex task by itself. Cases when only a single Web site is a source of necessary information are actually quite rare. Stock quotes, airline schedules and weather information—the kinds of information that are needed in business applications—are usually provided by dozens of sites on the Web. As a result, selecting a source of information often becomes the first step in generating a wrapper.

There are many factors, that have an effect on the decision to select a particular Web site as an information provider, including site performance and availability, data completeness, data distribution (how few or how many Web pages must be retrieved to extract a data set), and others.

Web site analysis

Once a Web site is selected for data retrieval (but before a wrapper is created for it) a thorough analysis of the site must be performed. Good analysis usually makes the subsequent programming effort easier and the resulting code more efficient, less bulky, and easier to maintain. There are many properties and features of the Web site that the analyst has to identify in her research. All of them will influence how the resulting wrapper will behave and how effective it will be:

- *Starting page/deep linking.* Identifying the starting page where the data retrieval process should begin is very important, as it allows the wrapper to skip unnecessary intermediate pages and go straight to data pages, or *deep-link* to them. This will primarily work for *static* URLs that do not change because of context. *Dynamic* URLs (URLs that contain changing parameters) can also be used for deep linking if their structure is well known. The wrapper can substitute parameter values in such URLs to get different data depending on user request.

- *Page identification.* Before the data extraction can actually take place, the wrapper has to make sure that the page containing data was actually retrieved. Sometimes the page that is expected does not arrive. Network and site problems may generate HTTP error pages in response to legitimate requests. Site structure and layout changes could also cause generation of error pages. Finally, the query that the wrapper poses to the Web site might bring no results. For reliable wrapper execution it is important to check for all of these signs before attempting to perform data extraction.

- *Location markers.* One of the ways to locate data inside HTML is to search for it with reference to some visible *location marker* (or "landmark" – [3], [4], [16]) inside the document. A location marker is any HTML element or group of elements that is uniquely identifiable and that is located close to the data that we are extracting. Once the wrapper finds such a marker inside the document, it could then "move" through the page relative to marker's location in order to find data.

- *Data identification markers.* These markers are unique characteristics of an HTML document that point to data elements and delimit distinct data records. Such markers are usually unique only inside some part of an HTML document and they must be used together with location markers and other search techniques. They can exist in many different forms, but HTML elements that highlight parts of the document are most commonly used for this purpose. Such elements include tags that specify fonts, paragraphs, record breaks, table cells and colors. Some of the HTML elements that are not visible in the browser are also useful for data identification. For example, comments and nonprintable symbols inside text records do not manifest themselves to the user in any way, but the wrapper can split data into records using them as delimiters.

- *Tree search.* One of the most powerful ways of locating data in HTML documents is through searching. We can search for a variety of pieces of information, such as tags, tag attributes, their values, text elements and comments. For the purposes of a particular application we can search for exact strings or substrings, searches can be case-sensitive or case-insensitive.

  In Data Extractor, HTML documents are stored in the form of *trees,* similar to tree structures in [7]. In a tree structure, additional types of searches are possible. Searching can be done in the entire tree or in any of its subtrees. This means that the search can start from the root element or from any element inside the tree structure and only affect descendants of that element. This is a useful property, because it allows us to localize the search to specific logical parts of the HTML document, ignoring the rest. Additionally, we can search *linearly*. In a linear search, the document is treated like a flat stream of HTML elements. The search starts from a given position in the document and continues until its end or until the element is found. This is different from subtree search, because a subtree search finishes when the element is found or when every node in the subtree is visited by the search routine.

  The decision to select either type of search depends on application needs. For some applications, locating data inside isolated portions of the document is important (e.g. for searching inside tables). For other cases (like searching for location markers) it is easier to think of the document as a flat file and search linearly.

- *Paths.* The simplest yet the least reliable (in terms of long-term stability of the wrapper) way to locate data inside a page is by specifying a *path* to it. A path is an ordered set of nodes in the document tree that we have to traverse to reach our destination. For example, if, in order to reach a particular node in the tree, we need to start at the root element, go to the second child of that element, then go to its first child, and, finally go to the third child of that node then the resulting path will be encoded as {2,3,1}.

Unfortunately, this approach is the leading cause of wrapper failure in the event of site changes. If a single node in the path is changed the whole path becomes invalid, requiring wrapper modifications.

It might be tempting to use paths as the only method for locating data inside HTML because of their simplicity. Paths are also much easier to implement in tools that assist the analyst or developer in building wrappers. It is better, however, to concentrate on searching techniques, or to combine searching with short paths that do not originate at the root node, because this will improve wrapper robustness.

- *Multipage data.* The absolute majority of sites that provide large volumes of data dispense it in portions, showing it to the user through sets of linked pages. Data extraction of multi-page results is done continuously. When the first page of results is processed, the wrapper follows the link to the next page. When that page is loaded, data extraction is done and the process repeats. The data that is extracted is returned as a single data table or data stream to the process that executes the wrapper.

- *Parallel page retrieval.* Taking advantage of a *parallel page retrieval* technique can significantly increase wrapper performance, as page downloads are done by multiple concurrent threads of execution. This technique works well when data is distributed across multiple pages and links to several such pages are accessible from a single location, such as an index or summary page.

- *Hidden data extraction.* Data extraction is usually done on textual portions of the HTML page. To be effectively communicated to the user, data has to be highly visible and occupy a prominent position inside the page. There are times, however, when data can be extracted from other, *hidden* parts of HTML. Good candidates for this are comments, tag attributes and their values, URLs, and even scripts. Useful data might include IDs, prices, addresses, phone numbers, and other pieces of information. Extracting this information can save time on unnecessary page retrievals.

- *Scripting simulation.* Web sites are often *script-intensive*—in other words, they make wide use of JavaScript and VBScript for a variety of tasks. Some of these tasks, such as various visual effects, are not of particular interest to data extraction. Others, such as navigation, or form validation and submission, are important. Script execution is beyond the current capabilities of the Data Extractor system, but a wrapper can be programmed to *simulate* some of the actions of the script without having to execute it.

- *Weak binding.* By *binding,* we mean wrapper reliance on particular features of the Web site. In order to identify and extract data from an HTML page a wrapper has to look for certain features and markers inside the page. Such binding has to be *weak* so a wrapper could withstand minor Web site changes without having to be rewritten or corrected.

  This issue is closely related to all of the site analysis and description techniques discussed so far and must be taken into account when applying them. Trying to find a balance between reliable data identification and weak binding has proved to be rather challenging. It is hard to come up with universal recipes on how to do this optimally, as these two tasks are inherently contradicting. The weaker the binding, the less reliable data identification within the site is. The stronger the binding, the better data identification is, and the greater the chance that the wrapper will not withstand the next site change.

  In the absence of a clear-cut solution to this problem, we suggest that the site analyst try to identify the smallest set of site features that will help pinpoint data location inside the site. When selecting these features, the analyst also has to make sure that they are *content-dependent* (could be searched for) rather than *structure-dependent* (are located at specific positions inside markup trees), as the latter is more likely to change. When such set of features is identified, it can be used to create a wrapper that is more tolerant to site changes.

Wrapper implementation

When the Web site analysis has been completed, the next step is to implement the wrapper. In the Data Extractor project, wrapper code is implemented in Java using the functionality of the Data Extraction Library. The results of site analysis that describe steps to traverse the site and acquire data are implemented in Java and debugged using the *wrapper executor.* Wrapper executor is a Java application that, through a simple interface, allows a programmer to specify wrapper parameters, execute them and step through the wrapper code in any Java development environment. Debug information about network communications and returned data is given as feedback to the programmer during wrapper execution. When the wrapper is debugged and tested, it can be integrated into the Data Extractor system through the knowledgebase.

Data output: The data that is extracted from Web sites is either returned directly to the user or fed into the calling application that analyzes and processes it. For a wrapper to be integrated into heterogeneous and other database systems its interface has to act as a mini-database system that produces data in response to queries. Therefore, one of the major tasks of wrapper analysis and implementation is the definition of the structure of the wrapper's output, or *schema.* Schema description and registration is done through the knowledgebase.

We define a schema by specifying names and types for pieces of data, or *fields*, that the Web site provides. The wrapper is then programmed to output data using the field information defined in the schema.

In the Data Extractor system, wrappers can return data both row-by-row and in complete tables. In the row-by-row approach, a row can be output as soon as the data for it is extracted from the Web site. This simplifies concurrent execution of wrapper and applications that process and consume data. As soon as the wrapper has extracted and returned the first record to the calling application, data can immediately be filtered, modified or otherwise processed by that application. Building and returning a table might be the only option in cases when complete data for each row is not available until the end of the extraction process.

There are problems associated with schema definition. Data available on a Web site is usually taken from a data source or database internal to that Web site. Only a small portion of that database is displayed to the user. Knowledge about the database schema is not exposed: no information is given on how data is decomposed into tables internally or what relations exist between tables. Some fields (e.g. internal codes or product IDs)

are rarely shown to the user. Finally, the size of the data set displayed is often less than the one that is stored in the database. All of these factors make schema definition complicated.

Wrappers in the Data Extractor project return data in simple two-dimensional tables similar to the ones used in relational databases. There is a single table defined for each wrapper. In the future we plan to use more complex data structures and generate multiple tables from a single wrapper.

Wrapper parameters: Some of the advantages of wrappers lie in their ability to shield the user from Web site complexity, and to generate data in response to requests made through a simple interface. In order to be truly useful, a wrapper has to be able to execute a class of queries, not just a single query (i.e. a wrapper that extracts weather information for major cities is more valuable than wrapper that can only give weather for New York City).

In the Data Extractor system, wrappers can have *parameters* that let users modify their behavior. In the example above, a wrapper would receive the name of the city as a parameter and then pass that information to the Web site to get weather for that city.

4. IMPLEMENTATION

Data Extraction Library

The purpose of the Data Extraction Library is to provide Web document retrieval, parsing and data extraction functionality for wrappers, ensuring full interoperability with any Web site and full browser simulation.

The Data Extraction Library provides four main groups of functionality:

- *Page retrieval functionality.* Fast and reliable page retrieval from the Internet is crucial to wrapper operation. In Data Extraction Library page requesting and retrieval is done through *sessions*. A session is a conversation with a Web server, the result of which is a stream of data. This data can be read as a stream or (when data is actually an HTML page) can be converted into a *document*. Sessions are more than simple HTTP request/response pairs—they provide rich functionality for building requests and modification of parameters (similar to filling out forms in browsers).
- *HTML processing functionality.* The contents of the retrieved HTML pages are stored internally in the form of a tree that consists of markup, or tags, and text elements. Each HTML page that is retrieved from the Web is automatically converted into an HTML tree. The parsing and document storage functionality was built to parse HTML and any other markup language that is based on SGML. It is, for example, capable of storing and processing XML and other XML-based formats. HTML processing functions give wrappers access to every detail of the document and feature powerful tree search and manipulation capabilities. Special functionality allows the creation of page downloading sessions from HTML forms and links, and automates data extraction from hierarchies of tags and HTML tables.

  The HTML parser that we implemented is not a *general-purpose parser* because it is lacking the *validation* mechanism. The validation functionality is not necessary in Data Extraction Library, because we only need to store documents in memory, traverse them and search them for information. Tags are treated equally regardless of their type. This makes the parser fast, more robust (because it does not refuse to process documents that are syntactically incorrect), and able to build a tree for almost any document.
- *Data representation functionality.* When data is extracted from the Web, it is shipped to the consumer. There are two ways to store data in the Data Extraction Library—in rows and in tables, with tables being collections of rows. This model allows us to create wrappers that generate data in either block or stream fashion. For the majority of wrappers, stream is the preferred way to output data because it allows portions of data to be processed by a higher-level application while the rest of it is still being extracted.
- *Wrapper interface functionality.* Wrapper interface functionality provides a simple communication and control mechanism that simplifies the implementation of wrappers and transmission of data. It allows the calling process to fully manipulate wrapper execution, pass input parameters into the wrapper and control the flow of data out of it. Through this interface, a wrapper reports errors to the Data Extractor system, which tells systems administrator that the wrapper needs to be updated.

Challenges

Some difficulties were encountered while implementing the functionality for HTML parsing and page retrieval.

- *Syntax errors.* In the course of building wrappers for a variety of sites we found a large portion of HTML pages to be syntactically incorrect. Shockingly, an estimated 90% of all Web pages on commercial Web sites we have analyzed so far have contained syntax errors. The high quality of modern Web browsers is partially to blame for this. In order to accommodate the widest possible variety of Web sites and make an effort to display any Web page, no matter how badly structured, the browsers were made extremely forgiving. HTML page authors can miss or mismatch closing tags, put end tags in the wrong order, not close comments or make other mistakes—and browsers will not alert her to the problem.

  Syntax errors, however, had little impact on the work of our parsing functionality. A wrapper could not be built due to hopelessly incorrect HTML pages for only one site out of over a hundred we analyzed. For all other sites the parser did a satisfactory job of building markup trees out of pages. Such trees weren't always correct semantic representations of documents, but they provided a data structure that is adequate for traversing and information extraction.
- *Slowness of core network functionality.* Java is an interpreted language and this takes its toll on the performance of time-critical routines in the standard Java libraries. Slowness of the network functionality in Java contributed the most to the overall slowdown of the wrapper operations. In our tests between 40% and 70% of the overall wrapper execution time was spent connecting to Web servers, sending requests for pages and receiving pages. In comparison, only about 5-10% of the time was spent on parsing and processing, and the rest—on

operations associated with data extraction and control of wrapper execution.

Network operations were somewhat sped up when the standard Java HTTP protocol implementation was re-written using sockets and when other optimizations were applied as suggested in [11]. This way a lot of unnecessary operations were eliminated, making the implementation leaner, faster, and more flexible, accommodating redirects, cookies, and other protocols.

In the future we expect the slowness of the network functionality to remain one of the stumbling blocks for successful wrapper implementation. Slowness of such operations is not inherent to Java—the same operations implemented in C++ for comparison purposes performed only insignificantly faster. This decreased performance significantly reduces the usefulness of the applications written using wrapper technology because the speed of data set generation is slow and sometimes insufficient for a satisfactory user experience.

Data Extraction Scripting Language

As the Data Extractor project was progressing, two things soon became apparent. First, the majority of the Web sites which were analyzed and for which wrappers were implemented had a simple structure and did not need the full power of Java for data extraction. Second, maintenance of Java wrappers became cumbersome in some cases, where Web sites would change their structure once in three months or even more often, which in turn required changing the Java source of the appropriate wrapper. These observations initiated the work on a *Data Extractor Scripting Language (DESL),* a simple scripting language that will make fast definition of wrappers possible for the majority of Web sites.

We followed several requirements when we designed DESL. First, it had to be simple, expressive, and cover only the functionality necessary to extract data from HTML. It was not necessary to create another programming language similar in power to Java. Simplicity and expressiveness of the language improve understandability and reduce code size, thus reducing overall maintenance time. Java can still be used in cases when a site is too complex for a scripting language.

Second, we had to be able to generate scripts in this language using a user-friendly GUI. One of the plans for future development of Data Extractor is to build a GUI environment, where a designer could create and update wrappers quickly, using a WYSIWYG ("What You See Is What You Get") interface. A wrapper would be a result of a "macro recording" of the steps the designer takes through the Web site and the data extraction instructions would be generated in response to the data fields that the designer highlights inside the site. Because of the demands of the GUI, DESL must support "round trip engineering." This means that we should not only be able to generate the script based on designer actions, but also import it into the GUI afterwards and modify it if the need arises.

DESL syntax is currently being finalized. We will report on it as soon as the development and testing of the working prototype is completed.

Related work

Over the past several years, many researchers have studied ways for collecting and processing data available on Web sites. Although a fully automatic extraction and labeling of data on arbitrary sites is currently beyond the capabilities of computer science, assisted, learning-based data extraction has been quite successful in some systems ([1], [9], [13]).

An alternative approach to data extraction is based on custom wrappers built around Web sites. Wrappers are coded manually or generated through special wrapper-generating browsers, such as the ones described in [12], and [22]. In our research we used a wrapper-based approach because it gives the highest accuracy of results and can be used to cover virtually any problem domain. As many researchers have noted, these advantages are sometimes offset by the costs of continuous wrapper maintenance. To help reduce those costs, we intend to further develop a two-fold approach to generating wrappers, where wrappers are written in a simple scripting language and wrappers for complex sites are written in Java.

Specialized languages dominate wrapper development, with implementations ranging from Prolog-like predicate logic ([5], [14]) to SQL flavors ([2], [3], [17]). We have not seen wrapper implementations based on a general purpose programming language like Java.

Some of Data Extractor's features can be found in existing systems. These features include, in particular, regular expressions for searching, data extraction and navigation through HTML documents ([2], [17], [18]), tree representation of HTML documents ([2], [4], [8]) and form processing ([3], [6]). XML query languages (e.g. XSLT [23]) have also influenced our research.

5. CONCLUSIONS AND FUTURE WORK

In this work we have described a Data Extractor system that facilitates data retrieval from Web sites. Data Extractor plays an important role as a data provider for the MSemODB heterogeneous database system. The ability to access data not only from relational and semantic databases, but also from the unstructured Web data sources significantly increases the power of MSemODB and extends the range of applications it could be applied to. The solution is portable and can be used both as a standalone data provider and embedded into applications.

We defined custom functionality for implementing wrappers using a high-level programming language and a library of specialized data extractions. A description of a wrapper construction process and a number of techniques and guidelines for site analysis and wrapper design were presented. Data Extractor fully simulates user interaction with a browser and can fill out forms and extract data from complex sets of linked Web pages.

In our future work we will focus on simplification of wrapper design process by building wrapper generation tools and further developing DESL. The problem of purity of extracted data also needs serious attention, because data frequently needs to be cleaned to filter out erroneous records. Translation that helps reconciliation of data from different sources is also necessary. We are also experimenting with data output formats similar to OEM [19] and XML.

6. ACKNOWLEDGEMENTS

We would like to thank the entire Data Extractor team at HPDRC for the invaluable help and stimulating discussions during the writing of this paper.

7. REFERENCES

[1] Adelberg, B. and Denny, M. Nodose version 2.0. Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, 1999, Pages 559 - 561

[2] Arocena, G. and Mendelzon, A. WebOQL: Restructuring Documents, Databases, and Webs. Proceedings of ICDE'98, Orlando, February 1998.

[3] Bauer, M. and Dengler, D. InfoBeams--configuration of personalized information assistants. Proceedings of the 1999 International Conference on Intelligent User Interfaces, 1999, Pages 153 - 156

[4] Bauer, M., Dengler, D. and Paul, G. Instructible information agents for Web mining. Proceedings of the 2000 International Conference on Intelligent User Interfaces, 2000, Pages 21 - 28

[5] Cohen, W. A Web-based information system that reasons with structured collections of text. Proceedings of the 2nd International Conference on Autonomous Agents, 1998, Pages 400 - 407

[6] Doorenbos, R., Etzioni, O., Weld, D. A Scalable Comparison-Shopping Agent for the World-Wide Web. Autonomous Agents '97

[7] World Wide Web Consortium. Document Object Model (DOM). http://www.w3.org/DOM

[8] Embley, D. W., Jiang, Y. and Ng, Y.-K. Record-boundary discovery in Web documents. Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, 1999, Pages 467 - 478

[9] Grumbach, S., Mecca, G. In Search of the Lost Schema. In Proceedings of International Conference on Database Theory (ICDT'99), 1999

[10] Hammer, J., Garcia-Molina, H., Ireland, K., Papakonstantinou, Y., Ullman, J., and Widom, J. Information Translation, Mediation, and Mosaic-Based Browsing in the TSIMMIS System. In Exhibits Program of the Proceedings of the ACM SIGMOD International Conference on Management of Data, page 483, San Jose, California, June 1995.

[11] Heydon, A., and Najork, M. Performance Limitations of the Java Core Libraries. In Proceedings of the 1999 ACM Java Grande Conference, pages 35-41, June, 1999.

[12] Liu, L., Han, W., Buttler, D., Pu, C. and Tang, W. An XJML-based wrapper generator for Web information extraction. Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, 1999, Pages 540 - 543

[13] Lim, S.-J. and Ng, Y.-K. An automated approach for retrieving hierarchical data from HTML tables. Proceedings of the 8th International Conference on Information Knowledge Management, 1999, Pages 466 - 474

[14] Lakshmanan, L., Sadri, F., and Subramanian, I. A Declarative Language for Querying and Restructuring the World-Wide-Web. Post-ICDE IEEE Workshop on Research Issues in Data Engineering (RIDE-NDS'96), New Orleans, February 1996.

[15] Mecca, G., Atzeni, P., Masci, A., Sindoni, G. and Merialdo, P. The Araneus Web-based management system. Proceedings of ACM SIGMOD International Conference on Management of Data, 1998, Pages 544 - 546

[16] Muslea, M., Minton, S. and Knoblock, C. A hierarchical approach to wrapper induction. Proceedings of the 3rd Annual Conference on Autonomous Agents, 1999, Pages 190 - 197

[17] Mendelzon, A., Mihaila, G., Milo, T. Querying the World Wide Web. PDIS 1996, December 1996, pages 80-91

[18] Neven, F., and Schwentick, T. Expressive and efficient pattern languages for tree-structured data (extended abstract). Proceedings of the 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, 2000, Pages 145 - 156

[19] Papakonstantinou, Y., Garcia-Molina, H., Widom, J. Object exchange across heterogeneous information sources. In Proceedings of the Data Engineering Conference. Computer Society of IEEE, Taipei, Taiwan, March 1995.

[20] Rishe, N. Database Design: the semantic modeling approach. McGraw-Hill, 1992, 528 pp.

[21] Rishe, N., Yuan, J., Athauda, R., Lu, X., Ma, X., Vaschillo, A., Shaposhnikov, A., Vasilevsky, D. and Chen, S.C. SemanticAccess: Semantic Interface for Querying Databases. The International Conference on Very Large Data Bases (VLDB 2000), September 10-14, 2000.

[22] Sugiura, A. and Koseki, Y. Internet scrapbook: automating Web browsing tasks by demonstration. Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology, 1998, Pages 9 - 18

[23] World Wide Web Consortium. XSL Transformations (XSLT). W3C Recommendation. http://www.w3.org/TR/xslt