Reconfigurable Web Wrapper Agents

Chia-Hui Chang, *National Central University, Taiwan*

Jen-Jie Chiou, *Deepspot Intelligent Systems, Taiwan*

Harianto Siek, Jiann-Jyh Lu, and Chun-Nan Hsu, *Institute of Information Science, Taiwan*

Web wrapper agents exploit online Web data sources, facilitating information integration and reuse. With the DeepSpot Agent Toolbox, users can automate virtually all types of Web browsing sessions simply by browsing the target Web sites.

Web information integration differs from database information integration because of the Web's nature, where data is in interlinked, heterogeneous Web pages rather than tables or objects with a clearly defined schema. Building wrappers for relational databases is relatively easy because the defined structure lets other programs directly access the data. *Web wrappers*, however, must automate Web browsing sessions to extract data from the target Web pages so other applications can process that data. Each Web site has its own set of links, layout templates, and syntax. You could, in a brute-force solution, program a wrapper for each browsing session. However, such wrappers are sensitive to Web site changes and become difficult to scale and maintain.

We have developed a solution called the DeepSpot Agent Toolbox for rapidly generating intelligent agents that serve as Web wrappers for Web information integration. With this solution, Web wrapper agents can be easily maintained without skillful programmers. We've built various applications that demonstrate our approach's feasibility. Here, we describe a bioinformatics application and a large-scale e-government information integration task.

Development fundamentals

When developing the DeepSpot Agent Toolbox, we designed an XML-based script language called the Web Navigation Description Language. A WNDL executor interprets and executes scripts written in WNDL. The executor

- Represents complex navigation and data gathering behavior of a user session
- Expresses output in XML format that eases information interchange between applications
- Accumulates and integrate data extracted from Web pages along the navigation
- Handles dynamically generated hyperlinks and CGI query HTML forms
- Tolerates malformed HTML documents

We equipped an early prototype of our system with a wrapper induction system called Softmealy[1,2] to generate data extractors. Recently, we've developed another algorithm called IEPAD (information extraction based on pattern discovery).[3,4] Unlike other work in wrapper induction,[5,6] IEPAD applies sequential-pattern mining techniques to discover a document's data extraction patterns. The pattern discoverer applies the Patricia tree-based sequence mining technique[7] to discover possible patterns and a multilevel analyzer, which conducts several multiple string alignments for attribute extraction. IEPAD discovers a set of candidate patterns for the users to select and then generates labeled training examples for Softmealy. (We've also implemented and integrated a IEPAD user interface with the DeepSpot Agent Toolbox for users to generate extraction rules of Web pages.) This removes the need to manually label training examples and thus minimizes human intervention. Some heuristic-based products on the market claim to extract data from the Web automatically. However, these are limited to a narrow class of Web pages that match their heuristics. In contrast, IEPAD does not depend on heuristics. A complete Web wrapper agent includes a WNDL script and IEPAD data extractors.

The DeepSpot Agent Toolbox is a programming-by-example authoring tool that lets users gen-

1094-7167/03/$17.00 © 2003 IEEE
Published by the IEEE Computer Society

erate a Web wrapper agent by browsing target Web sites for their particular information-gathering task. We can reconfigure the generated Web wrapper agent with the same authoring tool to maximize a Web information integration system's maintainability and scalability. (Due to space limitations, we'll now focus on WNDL.)

Web Navigation Description Language

The WNDL is an XML application for describing a Web-browsing session. Although the terminology we use here is primarily based on the working draft *Web Characterization Terminology & Definitions Sheet*,[8] from the World Wide Web Consortium, we reuse some of these terms and endow them with slightly different meanings. We define their specific meanings used here as follows:

Definition 1. Logical Web Site: A cluster of Web pages that are related to each other, each page containing a certain amount of data. You can integrate the data distributed among these pages and have a logical meaning.

Definition 2. Web Page Class: A set of Web pages to which you can apply a given data extractor to parse and extract their contents.

Although the definition depends on the given data extractor's expressive power, a Web page class usually refers to a set of related Web pages generated by a single CGI program or Web pages with an identical layout template. For example, the output pages from PubMed's (a well-known repository of biological research papers) keyword search service comprise a Web page class (see Figure 1).

As with all XML applications, a WNDL script consists of a set of elements. Each element can have a set of attributes and subelements. In WNDL, we can describe a user session with a *data Web map,* which is conceptually a directed graph with *nodes* and *edges*. The DWM is the primary data container in a WNDL script. The information stored in a DWM describes how to reach targeted Web pages and how to extract the contents from those pages. The DWM subelement definitions are in an element Map. Subelements of Map include Entrance and one or more Node elements, and the element Entrance contains a subelement Edge. The edge in the element Entrance represents how to access a logical Web site outside the defined DWM's scope without further interaction with the

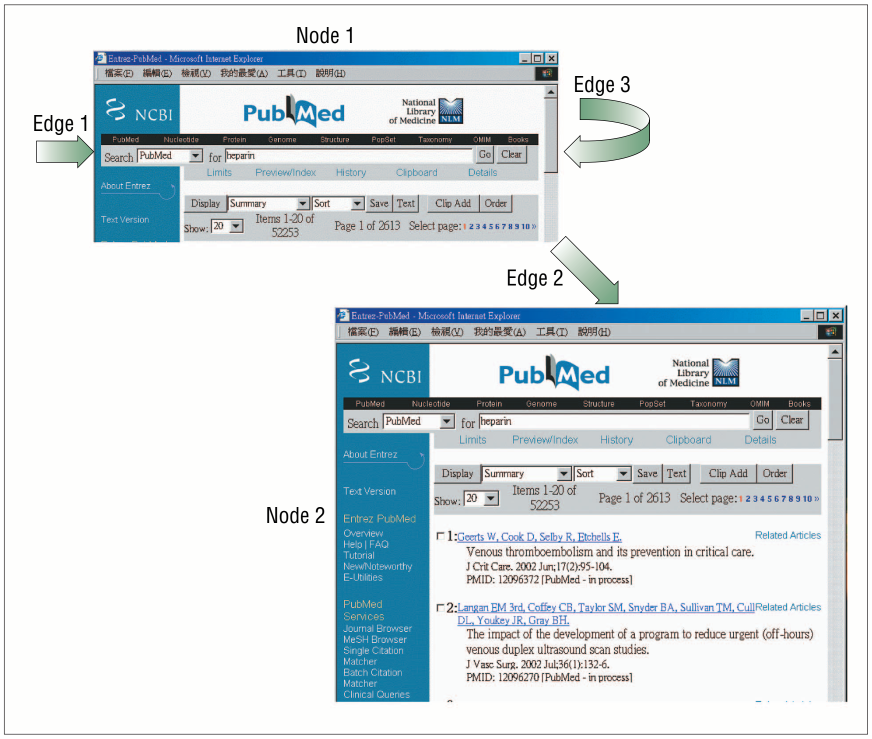


Figure 1. Conceptual illustration of the data Web map for PubMed.

Web server. Typically, this leads to a Web data source's front page. For example, the entrance to retrieve biological papers in PubMed is via its front page (www.ncbi.nlm.nih.gov/PubMed).

To demonstrate this process, we'll go through a complete example for modeling paper retrieval in a PubMed browsing session. We can view the PubMed browsing session conceptually as a graph with two nodes and three edges (see Figure 1).

DWM edge

An edge in a DWM represents a way to obtain a page that belongs to a Web page class denoted by this edge's destination node. A DWM edge serves as the container for the necessary information from the actual HTTP requests for both statically and dynamically generated HTML documents. The information for the requests consists of a set of parameters. We can either specify these parameters' values in the WNDL script or bind them during runtime.

Figure 1's example model has three edges. Edge 1 simulates submitting a new query. Edge 2 simulates browsing search results page by page, numbered from one to 10—each page contains 20 search results. Edge 3 simulates jumping to the eleventh page. For most Web sites, usually a next-page button leads to the following search results (say 21 to 40). However, PubMed's next-page button ">>" led to search results 201 to 220. Reaching the next 20 search results required following the 10 image links (numbered one to 10) one by one and then following the button ">>" for the next 200 results if any. Unlike URL hyperlinks that we can usually see on a Web page, the image links for the next pages were IMAGE INPUT {"page 1" to "page 10"} of the form named frmQueryBox.

We encoded (see Figure 2) the edges involved in these browsing steps in WNDL. Edge 1 is this map's entrance edge that sends the query to get the resulting Web page—that is, Node 1 in this case. The URL attribute can be a constant or a variable. In WNDL, HTML forms are treated as parameterized URLs. Their parameters are specified in element QueryParam, and its value can be a constant or a variable.

Once the first connection is successful, it leads us to destination Node 1. From Node 1, we can continue the next connection to Node 2 via Edge 2. As we described earlier, Edge 2 simulates browsing search results page by page. The HTTP connection information is embed-

```
<!— This is the entrance edge to Node1. —>
<edge ID='1' dest='Node1' method='post'
                    url='http://www.ncbi.nlm.nih.gov/genome/guide/gquery.cgi'>
        <QueryParam FormInput='db' value='0'/>
        <QueryParam FormInput='term' value='AIDS'/>
</edge>

<!— This is an edge to Node2 —>
<edge ID='2' src='Node1' dest='Node2' method='form'>
        <QueryForm='&form1'/>
        <QueryParam FormInput='&imglink'/>
</edge>

<!— This is an edge within Node1 —>
<edge ID='3' src='Node1' dest='Node1' method='form'
                    timeouts='20' retry='3' loops='100'>
        <QueryForm='&form2'/>
        <QueryParam FormInput='&nextTen'/>
</edge>
```

Figure 2. Edges in the Web Navigation Description Language script for PubMed.

```
<node name='Node1'>
<schema>
        <Attr Name="form1" type='edge' subtype='form' TagFilter="KeepAll"/>
        <ExtractRule File='node1/rule1/rule.txt'/>
</schema>

<schema>
        <Attr Name="form2" type='edge' subtype='form' TagFilter="KeepAll"/>
        <ExtractRule File='node1/rule2/rule.txt'/>
</schema>

<schema>
        <Attr Name="imglink" type='edge' subtype='image' TagFilter="KeepAll"/>
        <ExtractRule File='node1/rule3/rule.txt'/>
</schema>

<schema>
        <Attr Name="nextTen" type='edge' subtype='submit' TagFilter="KeepAll"/>
        <ExtractRule File='node1/rule4/rule.txt'/>
</schema>
</node>

<node name=Node2>
<schema>
        <Attr Name="Authors" type='Data' TagFilter="NoTag"/>
        <Attr Name="Title" type='Data' TagFilter="NoTag"/>
        <Attr Name="Source" type='Data' TagFilter="NoTag"/>
        <Attr Name="PMID" type='Data' TagFilter="NoTag"/>
        <ExtractRule File='node2/rule1/rule.txt'/>
</schema>
</node>
```

Figure 3. Nodes in the WNDL script for PubMed.

ded in the Web pages and can be extracted to bind the parameter values of Edge 2. In this case, because the values underlying the page numbers' images are not URL links but image submission INPUT, we must specify the form that specifies the action CGI. We can extract the image submissions and the query form and denote them with two variables, &form1 and &imglink. We can specify the connection by elements QueryForm and QueryParam. (We describe how to extract the values of these variables for Node 1 in a later section.)

Edge 3 is an edge that has an identical source and destination node, as Figure 1 depicts. Therefore, it is a self-looping edge. Like Edge 2, Edge 3's query type is an HTML form, where QueryForm is specified by variable &form2 and QueryParam refers to variable &nextTen. During runtime, Node 1 will form a self-loop. As we described earlier, we can express virtually any logical browsing session in WNDL.

Element Timeout is also a supplement of Edge. Timeout contains the control information of the event handling for time-outs. In WNDL, we can specify the number of retry attempts and the time interval between each attempt. The specified time interval equals a time-out event's time bound. If all attempts fail, the WNDL executor will throw an exception signal to its invocator.

Data Web map node

A DWM node represents one Web page class in a target logical Web site. A Web page class usually represents the pages that a CGI program generates. The CGI program can generate innumerable Web pages.

In WNDL, each node is a container of data in the pages of a Web page class. The contents extracted from a node's Web page class will be encoded by the data extractor as a database table, whose attributes we must define in a schema in advance. For example, for Node 2's Web page class in Figure 1, we want to export the retrieved papers' information into a table with these attributes: authors, title, source (where the paper was published), and PubMed ID (PMID). Figure 3 shows how we define them in WNDL (see the definition for Node 2). Because each Web page contains 20 search results, the correct output for this node should be a table of 20 records with these four attributes.

For each attribute in a table, we can specify our option for HTML tag filtering (KeepAll, KeepLink, and NoTag). This determines the behavior of a built-in HTML tag filter in the WNDL executor. WNDL also lets us describe

```
<map>
        <header inputValuesPath='./inputValues.txt>'
        <edge name='edge1' dest='Node1' method='post'
                url='http://www.ncbi.nlm.nih.gov/genome/guide/gquery.cgi'>
                        <QueryParam FormInput='db value=0'/>
                        <QueryParam FormInput='term value=AIDS'/>
        </edge>
        <node name='Node1'>
                        <schema>
                                <Attr Name="form1" type='edge' subtype='form' TagFilter="KeepAll"/>
                                <ExtractRule File='node1/rule1/rule.txt'/>
                        </schema>
                        <schema>
                                <Attr Name="form2" type='edge' subtype=form TagFilter="KeepAll"/>
                                <ExtractRule File='node1/rule2/rule.txt'/>
                        </schema>
                        <schema>
                                <Attr Name="imglink" type='edge' subtype='image' TagFilter="KeepAll"/>
                                <ExtractRule File='node1/rule3/rule.txt'/>
                        </schema>
                        <schema>
                                <Attr Name="nextTen" type='edge' subtype='submit' TagFilter="KeepAll"/>
                                <ExtractRule File='node1/rule4/rule.txt'/>
                        </schema>
        </node>
        <edge name='edge2' src='Node1' dest='Node2' method='form'>
                        <QueryForm='&form1'/>
                        <QueryParam FormInput='&imglink'/>
        </edge>
        <edge name='edge3' src='Node1' dest='Node1' method='form'
                        timeouts='20' retry='3' loops='100'>
                        <QueryForm='&form2'/>
                        <QueryParam FormInput='&nextTen'/>
        </edge>
        <node name='Node2'>
                        <schema>
                                <Attr Name="Authors" type='Data' TagFilter="NoTag"/>
                                <Attr Name="Title" type='Data' TagFilter="NoTag"/>
                                <Attr Name="Source" type='Data' TagFilter="NoTag"/>
                                <Attr Name="PMID" type='Data' TagFilter="NoTag"/>
                        <schema>
                        <ExtractRule File='node2/rule1/rule.txt'/>
        </node>
</map>
```

Figure 4. The WNDL script for PubMed.

how to join two tables extracted from adjacent nodes for the output. That way, we can aggregate data extracted during the browsing session in a user-defined manner.

The data extractor for a DWM node is specified as the value of element ExtractRule. The data extractor must be declarative in the sense that its extraction rules must be allowed to replace Web page classes without changing the program codes. In our implementation, we apply Softmealy and IEPAD as the data extractors. We can also use other declarative data extractors. ExtractRule's value can be the raw text of a set of extraction rules or an external file, specified as the value of attribute File of this element.

In our PubMed example, there are two nodes in the map, as Figure 1 shows. Node 1 represents the result of the entrance connection and will be used to extract the paths to the next pages. Node 2 represents query result pages returned from the search form of PubMed.

In Node 1, the information we are interested in is the <Form> HTML tag block in this page. Some Web sites use a user session ID mechanism to recognize HTTP requests from identical users to keep track of a user session. This helps Web servers determine the Web page content to return for different users. In some Web sites, HTTP clients (or browsers) need this ID to continue navigation, whereas some Web sites use this ID optionally. Because PubMed doesn't use a session ID, we can extract the HTML query form and use it directly. If a site uses a session ID, we must start from a static link to extract the query form and the dynamically generated session ID for the following steps.

Node 1 has four sets of extraction rules. The first and second sets extract the query form that contains the CGI program to the next page of the query results. Element Schema specifies that the extracted data will be bound to variables &form1 and &form2. The extraction rules' third and the fourth sets extract the INPUTs as the query parameters, which are bound to variables &imglink and &nextTen. Node 2 represents the query results returned from PubMed's search engine. The information we are interested in is the attributes of retrieved papers, including authors, title, source, and PMID. (See Figure 4 for the complete WNDL script for PubMed.)

We specify the schema of the output data in the extraction rules. The schema dictates which attributes defined in element Schema of the nodes will eventually appear in the output data table. In this example, the output data consists of a table with the four attributes defined in Node 2.

WNDL executor architecture and implementation

The WNDL executor consists of an executor kernel, page fetcher, and data extractor. Figure 5 shows the relationship between them and the order of execution steps. We can consider a WNDL script the configuration file of a Web wrapper agent that wraps a logical Web site. The configuration file defines the Web wrapper agent's behavior. During the execution, the executor kernel invokes the page fetcher and the data extrac-

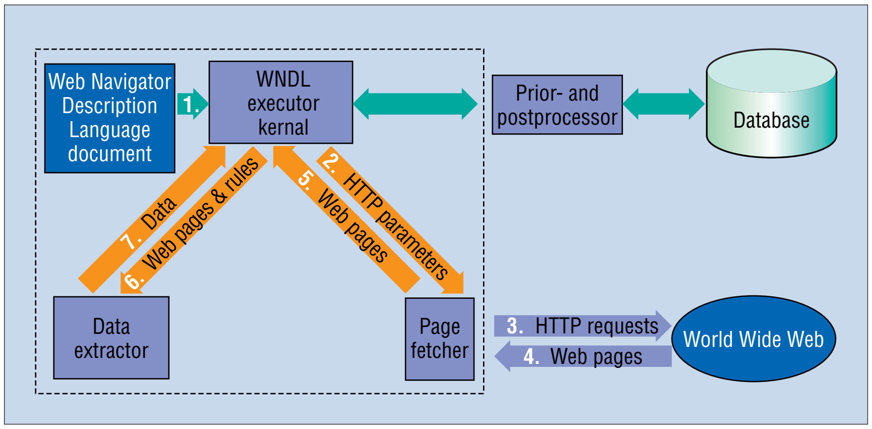
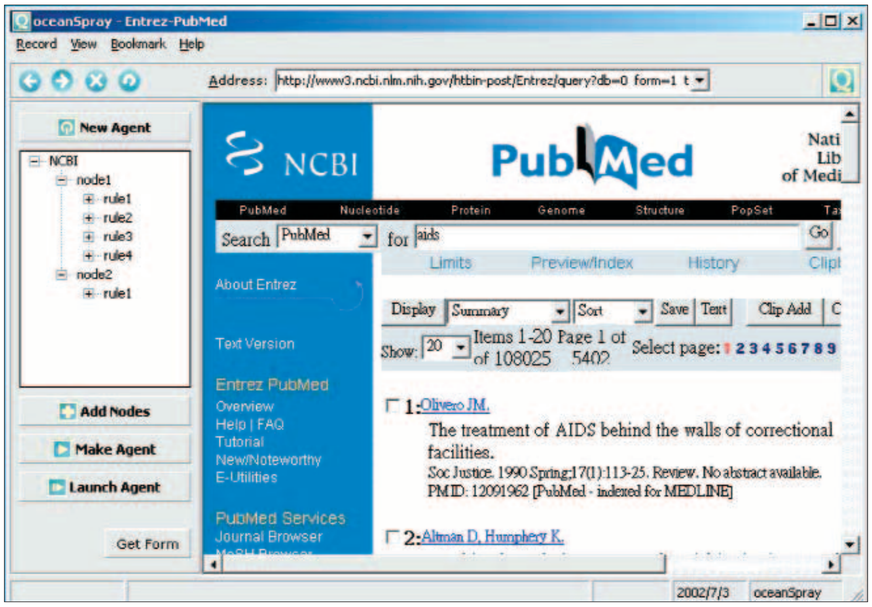


Figure 5. Architecture of the WNDL wrapper executor.


oceanSpray - Entrez-PubMed
Record View Bookmark Help
Address: http://www3.ncbi.nlm.nih.gov/htbin-post/Entrez/query?db=0 form=1 t
New Agent
NCBI
node1
rule1
rule2
rule3
rule4
node2
rule1
Add Nodes
Make Agent
Launch Agent
Get Form
NCBI
PubMed
Nati Lib of Medi
PubMed Nucleotide Protein Genome Structure PopSet Ta
Search PubMed for aids Go
Limits Preview/Index History Clip
About Entrez
Text Version
Entrez PubMed
Overview
Help | FAQ
Tutorial
New/Noteworthy
E-Utilities
PubMed Services
Journal Browser
Display Summary Sort Save Text Clip Add
Show: 20 Items 1-20 Page 1 of 5402 of 108025
Select page: 1 2 3 4 5 6 7 8 9
2002/7/3 oceanSpray

1: Olivero JM.
The treatment of AIDS behind the walls of correctional facilities.
Soc Justice. 1990 Spring;17(1):113-25. Review. No abstract available.
PMID: 12091962 [PubMed - indexed for MEDLINE]

2: Altman D, Humphery K.

Figure 6. Snapshot of OceanSpray.

tor according to the order the DWM map specifies, and the kernel handles static information and variable binding information to complete a Web-browsing session. The executor kernel maintains a pointer of the current state to traverse the DWM map. Basically, when the pointer points to an edge, the kernel invokes the page fetcher to obtain the next page and moves the pointer to the next node; when the pointer points to a node, the kernel invokes the data extractor and moves the pointer to the next edge.

The page fetcher abstracts HTTP connections to higher-level interfaces for the executor. HTML and HTTP features that the page fetcher can handle include form element parsing, GET and POST HTTP methods, cookies, time-out, user authentication, and malformed URL handling. The page fetcher transforms the parameters received from the executor into low-level, executable HTTP requests. After actually obtaining a Web page from a Web server, the page fetcher sends this page back to the executor kernel directly. The executor kernel then feeds this page and the corresponding extraction rules to the data extractor. Then, the data extractor will return the extracted data to the executor for further processing.A page might go through this process multiple times if it requires more than one set of the extraction rules.

OceanSpray

The DeepSpot Agent Toolbox, alos known as OceanSpray, is an authoring tool that can generate the script automatically. (For a trial version of OceanSpray, visit www.deepspot.com or contact the authors.) OceanSpray lets a user generate a WNDL script in a programming-by-example manner—that is, the user browses the Web to show the authoring tool an example user session, and the authoring tool generalizes the example into a WNDL script that describes this user session. Figure 6 shows a snapshot of OceanSpray interface after generating the complete WNDL script for PubMed. OceanSpray is equipped with IEPAD and Softmealy to generate extraction rules for the data extractors. With OceanSpray, it takes four steps to generate a WNDL script:

1. Open the target Web site's front page by specifying its URL in a Web browser.
2. Create nodes by clicking the "Add Nodes" button when browsing a new Web page class.
3. Invoke IEPAD to generate extraction rules for each node.
4. If you need more than one node, go back to step 2.

As Figure 6's left frame shows, the example script contains two nodes with five sets of extraction rules.

Creating the edges involves several steps. The first edge is created to reach Node 1. The user can accomplish this by clicking the submit button with parameters term and db set to value "aids" and "PubMed." The system monitors this query and compares it to all forms contained in the PubMed front page. OceanSpray will record the submitted parameters in QueryParam, which can be a constant value specified in the script or a variable bound to other user-specified query terms during execution. For Node 1, the user also need to generate four extraction rules and its schema. Each attribute in the schema is either of type Edge or Data. A Data attribute will appear in the final output, while an Edge attribute can be either a static link, form, submit button, or image button. The user can create an edge through a static link or a form with a submit image INPUT type. For each Edge attribute, the user must specify the destination node. Node 2 is created similarly.

Once specifying all nodes and edges, the user can obtain the complete WNDL script by clicking the "Make Agent" button. OceanSpray also provides a "Launch Agent" button for invoking the executor to test the generated WNDL script.

Applications

We have successfully applied our tool in several real-world projects, including a bioinformatics application, which demonstrates how an agent can automate a complex browsing session, and a large-scale e-government information integration task that involves thousands of Web wrapper agents.

Searching SNPs in ESTs

Many genomic and proteomic sequence databases in overwhelming volumes are available for public access on the Web. How to query and integrate these public domain databases has become an indispensable biological experimental technique and is one of the most critical issues in bioinformatics, where Web wrapper agents can play an important role. Our example application aims to search *single nucleotide polymorphisms* in *expressed sequence tags* by browsing the Web automatically. ESTs are short nucleotide sequences considered a shortcut to the alternative spliced and expressed forms of the genes. ESTs provide invaluable hints for interpreting genome sequences. dbEST, one of many Genebank databases hosted by NCBI, contained 17 million EST entries as of 11 July 2003 and is one of the largest and fastest growing sequence databases. An SNP marker, which received intense research attention lately, is the variant in DNA sequences that causes or contributes to the risk of developing a particular genetic disorder. Because identifying ESTs that contain a given SNP might shed new light on possible treatments of many genetic disorders, enormous efforts have been devoted to associating SNPs with ESTs.

Our example application regards an agent that automates the search of a set of known ESTs that contain a given SNP in the databases at NCBI. More precisely, given the reference cluster ID (or reference SNP number) of an SNP (such as rs1614984), the agent should return all ESTs in dbEST that contain this SNP. A user can browse the Web to obtain the search results. The browsing session requires the following steps:

1. From the dbSNP homepage (www.ncbi.nlm.nih.gov/SNP), enter the RS number to search the SNP data.
2. From the output Web page, extract the gene names and hyperlinks in the LocusLink section.
3. Follow each hyperlink and get the UniGene name and hyperlink in the Additional Links section.
4. Follow the hyperlink and get the Genebank access number and hyperlink of ESTs in the EST Sequences section.
5. Get the hyperlink to the dbEST entry in Sequence Information section.
6. Extract the sequence in the dbEST entry page.

Except for the first step, which starts from a static URL, all other steps involve dynamic URLs that a user must obtain from each previous step's search results. So, each search result requires a data extractor to extract specified data for use in the next step. Additionally, a user must repeat steps 3 to 6 for every gene name obtained in step 2. Therefore, it might take hundreds of interactions for a user to collect all known ESTs that contain a given SNP. For SNP rs1614984, the output includes 289 EST entries, which could take several working days to complete by browsing the Web by hand.

How to query and integrate these public domain databases has become an indispensable biological experimental technique and is one of the most critical issues in bioinformatics.

To automate this process, we can generate a WNDL agent for this task with our authoring tool by showing the tool how to obtain one of the EST entries given a SNP. One browsing path suffices for our tool to generate a WNDL script that generalizes to collect all intended EST entries and necessary data extractors. The WNDL executor will formulate the output into structured XML data records ready for postprocessing.

A common approach to such a task is to mirror dbEST and dbSNP (only flat files available) via FTP in advance and perform the search locally. That approach requires substantial programming skills and computing resources—a large disk space (about 100 Gbytes to store 17 million EST entries), parser to parse flat files, data normalizer to decompose parsed data into data tables that conform to the relational data model's requirement, and algorithms to identify ESTs that contain an SNP. In contrast, applying agent technologies requires much less programming skills and computing resources. Also, we can leverage available data links established by NCBI between databases. Moreover, each agent execution collects the most recent results in dbEST submitted from all over the world. This is critical for genetics databases that grow at a breakneck pace. With a timer and a redundant data filter, we can extend the agent to alert its users of interesting updates and new data entries in dbEST.

SARS information portal

Our second application is one of our largest, in which we built an integrated repository of seven categories of government public information available on the Web sites of 1,642 government agencies in Taiwan. The seven categories include, for example, government announcements and press releases, calendars of events, policy implementation plans, and administration guidance. The goal is to provide a single-query, complete-search service to the public and promote information sharing between government agencies.

Integrating 1,642 Web data sources is an extremely challenging task because each Web site has a particular structure, layout format, and backend architecture. Originally, the Taiwanese government established a repository of government announcements and press releases that government employees had to manually update. Because it was not mandatory for the government agencies to update the repository, the repository soon became out of date and useless.

Our team took over the project and applied our Web wrapper agents to resolve the problem. We had a team of 20 part-time students use our tool to build agents during their off hours. In four months, the team successfully built approximately 3,000 agents to update the repository. These agents automatically browse and extract seven categories of public information from the 1,642 Web sites and convert the extracted data into XML data with a uniform schema. We also developed a Web-based agent manager to manage these agents. We equipped the agent manager with a timer to launch agents periodically and a monitor to inspect the agents' execution status and error log. With the agent manager, the system requires only two part-time personnel to maintain it—a huge cost reduction compared to other options.

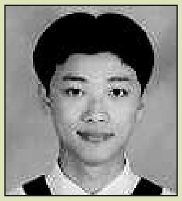
Our integrated government repository became particularly useful during the Severe Acute Respiratory Syndrome outbreak in Taiwan, May to June 2003, when rumors and unverified information caused unnecessary panic. To provide instant official information from related government agencies about the disease for integrated, easy online access, our team developed an instant official SARS information portal (http://gina.nat.gov.tw) based on the repository, with the agents updating the information hourly. The development project took only a couple of days to organize the SARS-related information the agents collected. This application demonstrates that applying Web wrapper agents is an efficient and cost-effective solution for integrating numerous heterogeneous data sources on the Web. In the future, we could rapidly develop similar instant information portals for other emergency situations such as a drought.

Acknowledgments

We thank the AIIA Lab alumni at the Institute of Information Science, Academia Sinica (Hung-Hsuan Huang, Chien-Chi Chang, and Elan Hung), for implementing the early version of the system. We also acknowledge the contribution of members in the R&D department of DeepSpot—Frank Lin, Melody Chen, Gibbs Wu, Vica Lin, Monte Liao, and Jerome Yeh—for their effort in building the e-government information integration application.

The Authors

Chia-Hui Chang is an assistant professor in the Department of Computer Science and Information Engineering at the National Central University in Taiwan. Her research interests include information retrieval, knowledge discovery from databases, machine learning, and Web-related research. She received her PhD in computer science and information engineering from the National Taiwan University. Contact her at the Dept. of Computer Science and Information Engineering, National Central Univ., ChungLi, 320, Taiwan; chia@csie.ncu.edu.tw.



Harianto Siek is a senior research engineer at the Adaptive Intelligent Internet Agent research lab of the Institute of Information Science, Academia Sinica, Taiwan. His research interests include AI and Web-related technologies. He received his BS in computer science from National Cheng-Chi University, Taipei, Taiwan. Contact him at the Institute of Information Science, Academia Sinica, Taipei, 115, Taiwan; chihai@iis.sinica.edu.tw.

Jiann-Jyh Lu is a senior research engineer at the Adaptive Intelligent Internet Agent research lab of the Institute of Information Science, Academia Sinica, Taiwan. His research interests include pattern recognition, data mining, and Web-related technologies. He received his MS in computer science and information engineering from the National Chiao Tung University, Taiwan. Contact him at the Institute of Information Science, Academia Sinica, Taipei, 115, Taiwan; jjlu@iis.sinica.edu.tw.

Jen-Jie Chiou is a bioinformatics specialist at Deepspot Intelligent Systems, Taiwan. His research interests include bioinformatics and Web-related technologies. He received his MS in occupational medicine and industrial hygiene, College of Public Health, National Taiwan University, Taiwan. Contact him at the Institute of Information Science, Academia Sinica, Taipei, 115, Taiwan; jjchiou@iis.sinica.edu.tw.

Chun-Nan Hsu is an associate research fellow at the Institute of Information Science, Academia Sinica, Taiwan. His current research interests include machine learning, knowledge discovery and data mining, databases, intelligent Internet agents, and their applications in bioinformatics. He received his PhD in computer science from the University of Southern California. He has two US patents pending. He is a member of the IEEE, ACM, AAAI, and Taiwanese Association for Artificial Intelligence (TAAI). Contact him at the Institute of Information Science, Academia Sinica, Taipei, 115, Taiwan; chunnan@iis.sinica.edu.tw.

References

1. C.-N. Hsu and M.-T. Dung, "Generating Finite-State Transducers for Semi-Structured Data Extraction from the Web," *Information Systems*, vol. 23, no. 8, Dec. 1998, pp. 521–538.
2. C.-N. Hsu and C.-C. Chang, "Finite-State Transducers for Semi-Structured Text Mining," *Proc. Int'l Joint Conf. Artificial Intelligence Workshop Text Mining: Foundations, Techniques, and Applications* (IJCAI 99), IJCAI Inc., 1999, pp. 38–49.
3. C.-H. Chang, C.-N. Hsu, and S.-C. Lui, "Automatic Information Extraction from Semi-Structured Web Pages by Pattern Discovery," *Decision Support Systems*, vol. 35, no. 1, Jan. 2003, pp. 129–147.
4. C.-H. Chang and S.-C. Lui, "IEPAD: Information Extraction Based on Pattern Discovery," *Proc. 10th Int'l Conf. World Wide Web*, WWW10 Ltd., 2001, pp. 681–688.
5. N. Kushmerick, D. Weld, and R. Doorenbos. "Wrapper Induction for Information Extraction," *Proc. 15th Int'l Joint Conf. Artificial Intelligence* (IJCAI 97), Morgan Kaufmann, 1997, pp. 729–737.
6. I. Muslea, S. Minton, and C.A. Knoblock, "A Hierarchical Approach to Wrapper Induction," *Proc. 3rd Int'l Conf. Autonomous Agents*, ACM Press, 1999, pp. 190–197.
7. D.R. Morrison. "Patricia—Practical Algorithm to Retrieve Information Coded in Alphanumeric," *J. ACM*, vol. 15, no. 4, Jan. 1968, pp. 514–534.
8. World Wide Web Consortium, *Web Characterization Terminology and Definitions Sheet*, working draft, May 1999, http://www.w3.org/1999/05/WCA-terms/.