

Reconfigurable Web Wrapper Agents for Web Information Integration

Chun-Nan Hsu[†], Chia-Hui Chang[‡], Harianto Siek[†], Jiann-Jyh Lu[†], Jen-Jie Chiou[‡]

[†]Institute of Information Science, Academia Sinica, Nankang, Taipei, Taiwan 105

[‡]Dept. of Comp. Sci. and Info. Eng., National Central University, Chung-Li, Taiwan 320

[‡]DeepSpot Intelligent Systems, Inc., Taipei, Taiwan, 104

Abstract

This paper provides a solution for rapidly building software agents that can serve as Web wrappers for biological information integration. We define an XML-based language called **WNDL**, which provides a representation of a Web browsing session. A **WNDL** script describes how to locate the data, extract the data and combine the data. By executing different **WNDL** scripts, user can automate virtually all types of Web browsing sessions. We also describe **IEPAD**, a data extractor based on pattern discovery techniques. **IEPAD** allows our software agents to automatically discover the extraction rules to extract the contents of a structurally formatted Web page without the need to label a Web page to train a wrapper. With a programming-by-example authoring tool, a user can generate a complete Web wrapper agent by browsing the target Web sites. We have built a variety of applications to demonstrate the feasibility of our approach.

1 Introduction

The information integration system provides its users a single cohesive view with seamless integrated information that cannot be easily accomplished by a single information provider. Information integration systems allow their users to formulate their queries in domain relation terms defined in advance to describe a target application domain. Then a *mediator* will construct a query plan to decompose the user query into subqueries to external data sources and determine execution and data flow orders of the subqueries. Mediators rely on *wrappers* to allow for transparent access to the data sources. Wrappers serve as the translators between the mediator and the data sources. Therefore, a wrapper is required for each type of data sources. Since the largest and the most up-to-date biological data sources are on the Web, building wrappers for Web data sources is important.

Web information integration is different from database information integration due to the nature of the Web, where data are contained in interlinked Web pages rather than tables or objects with clearly defined schema as in database systems. Building wrappers for relational databases is relatively easy because they are ready for access by another program. Web

wrappers, however, must automate Web browsing sessions to extract data from the contents of the target Web pages. But each Web site has its particular page linkages, layout templates, and syntax. A brute-force solution is to program a wrapper for each particular browsing session. That solution, however, may lead to wrappers that are sensitive to Web site changes and thus may become difficult to scale up and maintain. Our solution emphasizes the *reconfigurability* of the Web wrappers so that they can be rapidly developed and easily maintained without skillful programmers.

This paper provides a solution using an XML-based language, called **WNDL** (Web Navigation Description Language). Scripts written in **WNDL** are interpreted and executed by a *WNDL executor*, which offers the following features:

1. declaratively represent complex navigation and data gathering behavior of a user session,
2. XML format that eases information interchange between applications,
3. accumulate and integrate data extracted from Web pages along the traversal,
4. handle dynamically generated hyperlinks and CGI query HTML forms,
5. tolerate mal-formed HTML documents.

An early prototype of our system is equipped with a wrapper induction system called *Softmealy* [Hsu and Dung, 1998] to generate data extractors. Recently, we have developed another algorithm called **IEPAD** (an acronym for information extraction based on pattern discovery) [Chang *et al.*, 2003; chia Hui Chang and Lui, 2001]. Unlike the work in wrapper induction [Kushmerick *et al.*, 1997; Muslea *et al.*, 1999], **IEPAD** applies sequential pattern mining techniques to discover data extraction patterns from a document. This removes the need of labeling training examples and thus minimizes human intervention. There are some heuristic-based work on the market that claim to be able to extract data from the Web automatically. However, these work are limited to a very narrow class of Web pages that matches their heuristics. In contrast, **IEPAD** does not depend on heuristics.

A complete Web wrapper agent includes a **WNDL** script as well as **IEPAD** data extractors. We also developed a programming-by-example authoring tool which allows users

to generate a Web wrapper agents by browsing the target Web sites for their particular information gathering task. The generated Web wrapper agent can be reconfigured through the same authoring tool to maximize the maintainability and scalability for a Web information integration system. This paper describes more on WNDL and less on IEPAD due to the page limit. For more details on IEPAD please see [Chang *et al.*, 2003; chia Hui Chang and Lui, 2001].

2 Web Navigation Description Language (WNDL)

The Web Navigation Description Language (WNDL) is an application of eXtensible Markup Language (XML) for describing a Web browsing session. This section presents the definition of WNDL, the executor that executes a script in WNDL, and the authoring tool that generates WNDL scripts.

Although the terminologies used in this section are primarily based on a working draft, *Web Characterization Terminology & Definitions Sheet* [Consortium, 1999], from the World Wide Web Consortium (W3C), we reuse some of these terms and endow them with slightly different meanings. Their specific meanings in this context are defined as follows.

Definition 2.1 (Logical Web Site) *A cluster of Web pages that are related to each other, each page contains certain amount of data. The data distributed among these pages can be integrated together and have a logical meaning.*

Definition 2.2 (Web Page Class) *A set of Web pages that a given data extractor can be applied to parse and extract their contents.*

Though the definition depends on the expressive power of the given data extractor, a Web page class usually refers to a set of related Web pages generated by one single CGI program or Web pages with an identical layout template. For example, the output pages of PubMed's keyword search service comprise a Web page class.

2.1 WNDL Definitions

As all applications of XML, a WNDL script consists of a set of elements. Each element may have a set of attributes and subelements. In WNDL, a user session can be described by a data Web map (DWM), which is conceptually a directed graph with nodes and edges. The DWM map is the primary data container in a WNDL script. The information stored in DWM describes how to reach destined Web pages and how to extract the contents from those pages. The definitions of the subelements of DWM are enclosed in element Map. Subelements of Map include Entrance and one or more Node elements, and element Entrance contains a subelement Edge. The edge in element Entrance represents the way to access a logical Web site outside the scope of the defined DWM without further interaction with the Web server. Typically, this leads to the front page of a Web data source. For example, the entrance to retrieve papers in PubMed is via the URL <http://www.ncbi.nlm.nih.gov/PubMed>, its front page.

In the following subsections, we will go through a complete example for modeling the browsing session of retrieving

```
<!-- 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 WNDL script for PubMed

papers in the well-known online biomedical paper collection PubMed. The browsing session for PubMed can be perceived conceptually as a graph with two nodes and three edges as Figure 1 depicts.

Data Web Map Edge

An Edge in a DWM represents a possible mean to obtain a page that belongs to a Web page class denoted by the destination node of this edge. A DWM edge serves as the container of the necessary information of actual HTTP requests for both statically and dynamically generated HTML documents. The information for the requests consists of a set of parameters. Values of these parameters can be either specified in the WNDL script or bound during the run-time.

There are three edges in the example model. Edge 1 simulates the action of submitting a new query. Edge 2 simulates the action of browsing search results page by page numbered from 1 to 10 (each page contains 20 search results). Edge 3 simulates the action of 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 provides a next button “}” which leads to search results 201 to 220. To get the next twenty search results, we need to follow the ten image links (numbered 1 to 10) one by one and then follow the button “}” for the next 200 results if they exist. Note that unlike URL hyperlinks that can be usually seen in a Web page, the image links for next pages are `IMAGE INPUT {“page 1” to “page 10”}` of the form named “frmQueryBox.”

The edges involved in the above browsing steps are encoded in WNDL as shown in Figure 2. Edge 1 is the entrance edge of this map that sends the query to get the resulting Web page, i.e., Node1 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. Again, the value of QueryParam can be a constant or a variable. Note that some Web sites use a user session ID mechanism to recognize HTTP requests from an identical user to keep track of a user session. This mechanism helps Web servers to determine the contents of Web pages they should respond for each user. If such a mechanism is used, we have to start from a static link to extract the dynamically generated session ID in-

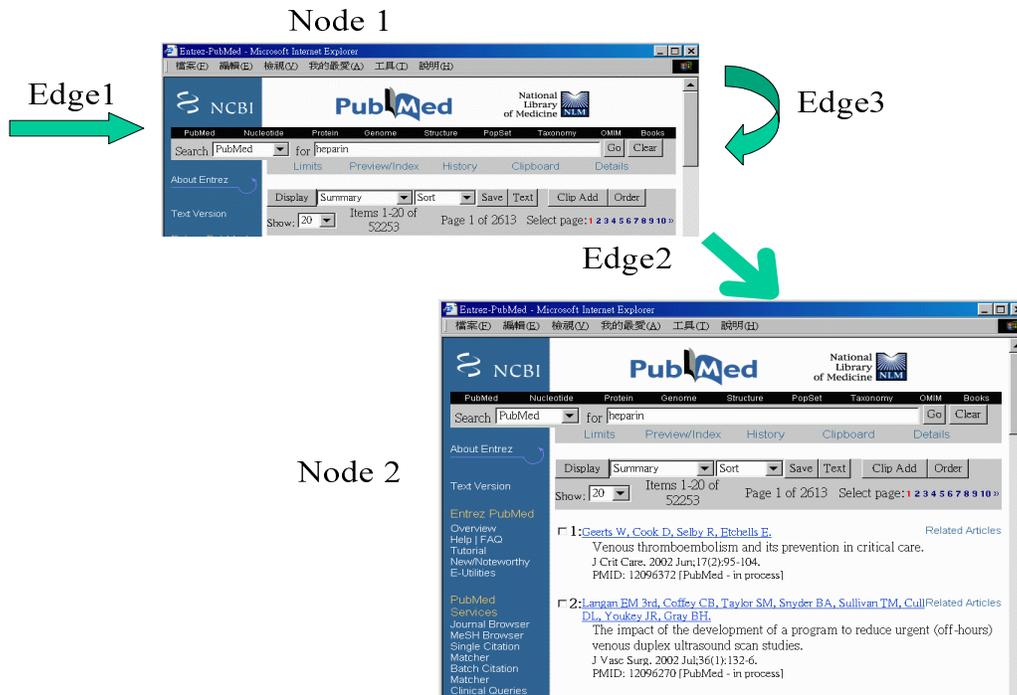


Figure 1: Conceptual illustration of the data Web map for PubMed

stead of sending an HTTP request directly to obtain a destined page.

Once the first connection is successful, it leads us to the destination Node1. From Node1, we can continue the next connection to Node2 via Edge 2. As described above, Edge 2 simulates the action of browsing search results page by page. The HTTP connection information is embedded in the Web pages and can be extracted to bind the parameter values of Edge 2. In this case, since the values underlying the images of the page numbers are not URL links but image submission INPUT, the form that specifies the action CGI must be specified. The image submissions and the query form can be extracted and denoted by two variables, `&form1` and `&imglink`, respectively. The connection can be specified by elements `QueryForm` and `QueryParam`. How the values of these variables are extracted for Node1 will be described in Section 2.1.

Edge 3 is an edge that has identical source and destination node as depicted in Figure 1. Therefore, it is a self-looping edge. Like Edge 2, the query type of Edge 3 is an HTML form, where `QueryForm` is specified by variable `&form2` and `QueryParam` refers to variable `&nextTen`. During the run-time, Node1 will form a self-loop. As described above, virtually any logical browsing session can be expressed in WNDL.

Element `Timeout` is also a subelement of Edge. `Timeout` contains the control information of the event handling for timeouts. In WNDL, the number of `retry` attempts

and the time interval between each attempt can be specified. The specified time interval is equal to the time bound of a timeout event. If all attempts fail, the executor of WNDL 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. Defined again here, a Web page class is a set of Web pages with similar layout templates such that one data extractor can be applied to successfully. A Web page class usually represents the pages that are generated by a CGI program. The number of Web pages that a CGI program can generate is innumerable.

In WNDL, each node is a container of data in the pages of a Web page class. The contents extracted from the Web page class of a node will be encoded as a database table, whose attributes must be defined in a *schema* in advance. For example, for the Web page class of Node2 shown in Figure 1, we want to extract the information of the retrieved papers into a table with the following four attributes: authors, title, source (where the paper published), and PMID (PubMed ID). Figure 3 shows how they are defined in WNDL (see the definition for Node2). Since each Web page contains twenty search results, the correct output for this node should be a table of twenty records with these four attributes.

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

```

<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

how to join two tables extracted from adjacent nodes for the output. That way, data extracted during the browsing session can be aggregated in user defined manners.

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 for different Web page classes without changing the program codes. In our implementation, we apply *Softmealy* [Hsu and Dung, 1998] and *IEPAD* (see Section 3) as the data extractors. Other declarative data extractors can be applied, too. The value of `ExtractRule` 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 shown in Figure 1. `Node1` represents the result of the entrance connection and will be used to extract the paths to the next pages. `Node2` represents query result pages returned from the search form of PubMed.

In `Node1`, the information we are interested is the `<Form>` HTML tag block in this page. Some Web sites use user session ID mechanism to recognize HTTP requests from identical user to keep track of a user session. This helps Web servers to determine the contents of the Web pages they should respond for different users accordingly. In some Web sites, HTTP clients (i.e., a browser) need this ID in order to continue navigation, whereas some Web sites use this ID optionally. Since PubMed does not belong to any of the above categories, the HTML query form can be extracted and used directly. If session ID is used, we have to start from a static link to extract the query form and the dynamically generated session ID for the following steps.

There are four sets of extraction rules for `Node1`. The first and second sets of the extraction rules 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 third and the fourth sets of the extraction rules extract the IN-

PUTs as the query parameters, which are bound to variables `&imglink` and `&nextTen`. `Node2` represents the query results returned from the search engine of PubMed. The information we are interested in is the attributes of retrieved papers, including authors, title, source, and PubMed ID. The complete WNDL script for PubMed is shown in Appendix.

The schema of the output data is specified 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 `Node2`. Section 2.3 explains how to specify the schema.

Composing extraction rules is not trivial and is another research problem itself. In Section 1, we have reviewed several systems designed to generate extraction rules from training examples. Section 3 presents our new approach from the perspective of pattern mining, which minimizes the need of human intervention in the generation of extraction rules.

2.2 Architecture and Implementation of WNDL Executor

The WNDL executor is composed of three components: executor kernel, page fetcher, and data extractor. Figure 4 shows the relationship between them and the order of execution steps. A WNDL script can be considered as the configuration file of a Web wrapper agent that wraps a logical Web site. The configuration file defines the behavior of the Web wrapper agent. During the execution, the executor kernel invokes the page fetcher and the data extractor according to the order specified in the DWM map, 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, timeout, user authentication and mal-formed 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. One page may go through this process multiple times if there are more than one set of the extraction rules required for this page. The extracted data will be returned to the executor for further processing.

2.3 WNDL Authoring Tool

In the early version of WNDL [Huang, 2000], the script is designed to be written by programmers. In this version of WNDL, the script can be generated automatically by an authoring tool. This authoring tool allows a user to generate a WNDL script in a programming-by-example manner, that is, the user simply browse the Web to show the authoring tool

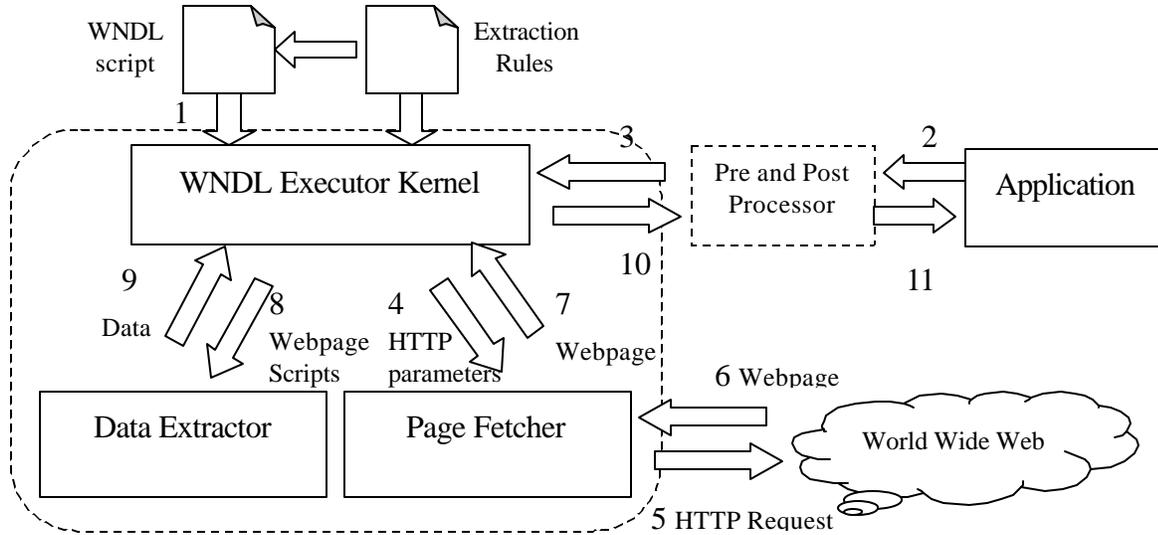


Figure 4: Architecture of WNDL Wrapper Executor

an example user session and the authoring tool will generalize the example into a WNDL script that describes this user session. Figure 5 shows a snapshot of its interface after generating the complete WNDL script for PubMed. The authoring tool is equipped with *IEPAD* and the wrapper induction system *Softmealy* [Hsu and Dung, 1998] to generate extraction rules for the data extractors. With this authoring tool, it takes only four steps to generate a WNDL script.

1. Open the front page of the target Web site by specifying its URL as using a Web browser;
2. Create nodes by clicking the “Add Nodes” button when browse to a new Web page class;
3. Invoke IEPAD to generate extraction rules for each node;
4. If more than one node is needed, go back to step 2.

As shown on the left frame of Figure 5, the example script contains two nodes with five sets of extraction rules.

We can create the edges as described below. The first edge is created to reach `Node1`. This is accomplished by clicking the submit button with parameters `term` and `db` set to value “aids” and “PubMed”. The submission of this query is monitored by the system and compared to all forms contained in the front page of NCBI. The submitted parameters are recorded in `QueryParam`, which can be a constant value specified in the script or a variable bound to other query terms specified by the user during the execution time. For `Node1`, we also need to generate four extraction rules as well as specifying 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 one of the four types: static link, form, submit button or image button. An edge can be created by a static link or a form with submit/image IN-

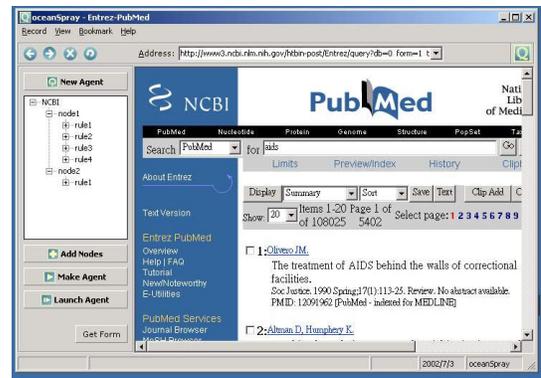


Figure 5: Snapshot of the authoring tool of WNDL

PUT. For each `Edge` attribute, the user has to specify the destination node. Two `Edge` attributes pointing to the same destination can be combined to create a dynamic edge. `Node2` is created similarly.

Once all the nodes and edges are specified, the complete WNDL script can be generated by clicking the “Make Agent” button. The authoring tool also provides a “Launch Agent” button for invoking the executor to test the generated WNDL script.

3 Information Extraction based on Pattern Discovery (IEPAD)

IEPAD can automatically discover extraction patterns without user-labeled training examples [Chang *et al.*, 2003; chia Hui Chang and Lui, 2001]. This feature is accomplished by

applying sequential pattern mining to discover the repeats in a Web page that contains multiple entries of data, such as tables, itemized lists, etc. The pattern discoverer applies the PAT-tree based sequence mining technique [Morrison, 1968] to discover possible patterns, and a multi-level analyzer, which conducts several multiple string alignments for attribute extraction. A user interface of IEPAD has been implemented and integrated with the authoring tool of WNDL for users to generate extraction rules of Web pages. IEPAD discovers a set of candidate patterns for the users to select and then generate labeled training examples for Softmealy [Hsu and Dung, 1998; Hsu and Chang, 1999]. Softmealy is a wrapper induction system that can generalize the labeled training examples into extraction rules for the WNDL executor to extract data from input Web pages.

4 Conclusion

In this paper, we presented a tool to exploit online Web data sources using reconfigurable Web wrapper agents. We described how these agents can be generated and executed based on the script language WNDL and extraction rule generator IEPAD. This tool has been applied successfully in a number of real-world projects. One of the largest projects aims at integrating seven different classes of public information (e.g., government press releases, public announcements, schedules, public hearing minutes, etc.) available on the Web sites of 1,000 government agencies. A team of about twenty part-time college students uses our tool to build agents during their off hours. In four months, the team has successfully built about 3,000 software agents to accomplish the task. We believe this tool can boost the productivity of the Web and help facilitate Web information integration.

Acknowledgements

We acknowledge the contribution of the alumni of AIIA Lab: Hung-Hsuan Huang, Chien-Chi Chang and Elan Hung for implementing the early version of the system. This research is supported in part by the National Science Council in Taiwan under grant NSC-90-2213-E-008-053.

A Complete WNDL Script for PubMed

```
<map> <header inputValuesPath='./inputValues.txt'>
  <edge name='edge1' dest='Node1' method='post'
    url='http://www.ncbi.nlm.nih.gov/genome/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>
</map>
```

```
</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>
```

References

- [Chang *et al.*, 2003] Chia-Hui Chang, Chun-Nan Hsu, and Shao-Chen Lui. Automatic information extraction from semi-structured web pages by pattern discovery. *Decision Support Systems*, 35(1):129–147, 2003. Special Issue on “Web Retrieval and Mining”.
- [chia Hui Chang and Lui, 2001] chia Hui Chang and Shao-Chen Lui. IEPAD: information extraction based on pattern discovery. In *Proceedings of the Tenth International Conference on the World Wide Web*, pages 681–688, Hong Kong, China, 2001.
- [Consortium, 1999] World Wide Web Consortium. Web characterization terminology and definitions sheet, May 1999. Working Draft.
- [Hsu and Chang, 1999] Chun-Nan Hsu and Chien-Chi Chang. Finite-state transducers for semi-structured text mining. In *Proceedings of IJCAI-99 Workshop on Text mining: Foundations, Techniques and Applications*, pages 38–49, Stockholm, Sweden, 1999.
- [Hsu and Dung, 1998] Chun-Nan Hsu and Ming-Tsong Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems*, 23(8):521–538, 1998.
- [Huang, 2000] Hung-Hsuan Huang. Design and implementation of a configurable wrapper for web information integration. Master’s thesis, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, July 2000.
- [Kushmerick *et al.*, 1997] Nickolas Kushmerick, Dan Weld, and Robert Doorenbos. Wrapper induction for information extraction. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, Japan, 1997.
- [Morrison, 1968] D.R. Morrison. Patricia-practical algorithm to retrieve information coded in alphanumeric. *Journal of ACM*, 15(4):514–534, Jan 1968.
- [Muslea *et al.*, 1999] Ion Muslea, Steve Minton, and Craig A. Knoblock. A hierarchical approach to wrapper induction. In *Proceedings of the 3rd International Conference on Autonomous Agents*, pages 190–197, Seattle, WA, 1999.