ON DESIGN OF BROWSER-ORIENTED DATA EXTRACTION SYSTEM AND THE PLUG-INS

Jui-Yuan Su  
*Department of Information and Telecommunications Engineering, Ming Chuan University, Taiwan, R.O.C,*  
rysu@mcu.edu.tw

Der-Johng Sun  
*Department of Computer Science, National Chiao Tung University, Taiwan, R.O.C*

I-Chen Wu  
*Department of Computer Science, National Chiao Tung University, Taiwan, R.O.C*

Lung-Pin Chen  
*Department of Computer Science and Information Engineering, Tunghai University, Taiwan, R.O.C.*

Follow this and additional works at: https://jmstt.ntou.edu.tw/journal

Part of the Electrical and Computer Engineering Commons

**Recommended Citation**

DOI: 10.51400/2709-6998.2318  
Available at: https://jmstt.ntou.edu.tw/journal/vol18/iss2/5

This Research Article is brought to you for free and open access by Journal of Marine Science and Technology. It has been accepted for inclusion in Journal of Marine Science and Technology by an authorized editor of Journal of Marine Science and Technology.
ON DESIGN OF BROWSER-ORIENTED DATA EXTRACTION SYSTEM AND THE PLUG-INS

Acknowledgements
The authors would like to thank the National Science Council of the Republic of China, Taiwan for financially supporting this research under Contract Numbers, NSC 94-2213- E-009-028, and NSC 95-2221-E-029-033.
ON DESIGN OF BROWSER-ORIENTED DATA EXTRACTION SYSTEM AND THE PLUG-INS

Jui-Yuan Su*, Der-Johng Sun**, I-Chen Wu**, and Lung-Pin Chen***

Key words: data extraction model, browser-oriented data extraction, URL-oriented data extraction, plug-ins, visualization tools.

ABSTRACT

Web data extraction systems currently not only extract data on web pages but also need to navigate to the target correctly. Most traditional web data extraction systems extract URLs directly from web pages, and then access next pages using the extracted URLs. This data extraction approach is herein called the URL-oriented data extraction approach in this paper. However, currently, more and more web pages use script functions, such as JavaScript, to access next pages and may hide URLs inside these functions, making it difficult to extract URLs.

In order to solve this problem, a new data extraction approach, named the browser-oriented data extraction (BODE) approach, is proposed to be built on top of browser objects access pages by simulating users’ operations on browsers to invoke script functions. A data extraction system based on this approach is called the BODE system.

Based on the BODE approach, this paper designed a BODE system with the following contributions: (a) Define a scripting language, named the BODED (Browser-Oriented Data Extraction Description) language, which instructs the BODE system to extract data. (b) Design a plug-ins that can be used to extend the functionalities of the BODE system. (c) Design a visualization tool to support the data extraction in the BODE system. (d) Illustrate the plug-in mechanism of the BODE system by automating the playing of the game Connect6 over an Internet game site.

I. INTRODUCTION

With the rapid growth of the World Wide Web (WWW), a huge amount of information has been published as Web pages. Hence, it is significant to collect needed information from web pages. Users usually want to extract the specific data from web pages, instead of retrieving the whole page. A system to extract data is called a data extraction (DE) system. For example, extract the prices of products from online auction and shopping websites for price comparison.

An example of extracting all article items from a simplified bibliography archive site with two-level category pages is shown as in Fig. 1. The main-category page in Fig. 2 lists categories associated with hyperlinks to the corresponding sub-category pages. The sub-category pages in Fig. 3 list subcategories associated with hyperlinks to the corresponding article-list pages. An article-list page, as illustrated in Fig. 4, lists details of article references in the same subcategory. At the end of the article-list page, a hyperlink is linked to the next article-list page for more references in the same subcategory.

The traditional approach [2, 3, 6, 12, 16, 17, 20, 21] for data extraction extracts plain text or URLs directly from web pages, and uses these extracted URLs to retrieve next pages via the HTTP requests. This approach is called the URL-oriented DE approach. However, more and more web pages include scripting languages, such as JavaScript [19], or use AJAX [9] to make the presentation and navigation of web pages more flexible and friendly. These pages usually use JavaScript functions to access other pages or make pages animated based on the DOM model [10].

If the needed URLs are hidden inside these script functions, it becomes much harder to extract URLs. For example, if the article-list page in Fig. 4 is rewritten with a JavaScript function, as shown in Fig. 5, the URL of the next page is hidden inside the JavaScript program. In these pages, the elements in the DOM may also be dynamically generated by the JavaScript functions according to user actions such as mouse clicking or keystroking. It is hard for traditional DE systems to extract these dynamically generated data.

Currently, a new approach for data extraction is to leverage browsers to solve the above JavaScript problems, such as WebVCR [1], Lixto [4], and iMacro [11]. This approach is called the browser-oriented DE approach. Instead of extracting text from HTML files, the systems in this approach can
However, the DE systems in the browser-oriented DE approach usually encounter several page consistency issues caused by several factors, such as the changes of server content, the interferences of JavaScript functions. In [27], Wu et al. proposed a data extraction model, called the browser-oriented data extraction model to address and solve the consistency issues.

To supporting the DE systems in the browser-oriented DE approach, this paper designs a BODE system with the following three contributions. First, this paper introduces a description language based on XML [7] called browser-oriented data extraction description (BODED) language. The BODED language supports the capability to locate DOM elements in browsers and issue actions on these elements to trigger JavaScript functions. In BODED language, XPath [5], a W3C standard, is used to express the location of the elements. The script written in BODED language is called a BODED script.

Second, this paper designs a plug-in, so that more applications can be applied by adding plug-in code, called BODEDlet. For example, save the extracted data into databases, avoid accessing redundant pages, and traverse web pages with specific rules. In order to support plug-ins, the system has to hide the internal processing while avoiding consistency problems, as mentioned above. To solve these problems, this paper design the system based on façade design pattern [22].

Third, this paper designs a visualization tool that can help users write BODED scripts. The tool is a WYSIWYG tool that greatly reduces the time taken to develop BODED scripts.

Section II in this paper reviews the BODE model including the consistency issues. Section III introduces the BODED language. Section IV describes the BODEDlet plug-in mechanism in our system. Section V introduces the visual tools of our DE system. Finally, Section VI illustrates the plug-in mechanism of the BODE system by automating the playing of Connect6 [24] over a game site, Little Golem [14]. Section VII gives the concluding remarks.

II. REVIEWS

In this section, we review the BODE model and discuss the consistency issues. Subsection II.1 introduce the BODE model. Subsection II.2 introduce the consistency problems that may occur in the BODE system.
1. BODE Model

The architecture of a BODE model is depicted in Fig. 6. A DE system following this model is called a browser-oriented DE system or a BODE system. A BODE system inputs a DE script and outputs the extracted data after following the instructions in the DE scripts. The BODE system retrieves documents from the Internet which is considered as an external environment. Such retrieval operations are called page requests.

A BODE system consists of the following modules: a controller, a network module, an extracted data pool (abbr. EDP), and a set of browser objects. The controller initially inputs a DE script, and then follows the instructions of the script to generate a sequence of DE actions on browser objects. The DE actions are classified into the following three classes.

1. Extraction actions. An extraction action extracts or locates the data from the designated browser object and places the extracted data into the EDP.
2. URL actions. A URL action sends a page request through the network module to retrieve a document and then designates a browser object to hold the document.
3. Page actions. A page action performs an operation on the designated browser object. E.g., clicking on a hyperlink, fill text into a text field, moving backwards to the previous page, or moving forwards to the next page, etc.

A set of browser objects holds HTML or XML documents to be extracted. The data in these documents can be accessed or processed via the Document Object Model (DOM) [10], which is a W3C standard model.

The network module is used to make all the page requests from the external environment. The network module includes protocol handlers, such as HTTP [8] and FTP, and services, such as proxy and caching services.

The EDP in the BODE system holds all the extracted data which may be either directly output or used to generate DE actions from the controller.

2. Consistency Problems

This section investigates two consistency problems for BODE systems: determinism and semantics consistency. The issue of determinism is discussed in Subsection II.2.1, while the issue of semantics consistency is discussed in Subsection II.2.2.

1) Determinism

A BODE system can be distinguished as deterministic and non-deterministic. If a BODE system described in above always produces the same output results for the same DE script, even when running at different times, then it is said to be deterministic. Otherwise, the system is non-deterministic.

In fact, a BODE system is very likely to be non-deterministic. Consider the following cases.

1. Externally, for the same given URLs, the web servers may generate different documents at different times.
2. Internally, the DE system may use time-related functions or use multi-threads, which may produce different results at different times.

The first case above is almost inevitable. Fortunately, caching documents for all page requests can solve this problem. Thus, in order to simplify our discussion about DE systems, we assume that for the same given URLs, the web servers always generate the same documents, even at different times during the period of data extraction.

The second case above that causes non-determinism (as described above) is mainly related to the design of DE systems.

Determinism is important in a BODE system because if the web pages are changed during data extraction, the behavior of the system may be unpredictable. If the system is designed in single thread and does not use time related functions such as timer or random number, then the system will be deterministic.

2) Semantics Consistency

This subsection addresses another consistent issue, called semantic consistency, which may occur when navigating to multiple destined pages. Navigating to multiple destined pages is called multi-way navigation in this paper. For example, a DE system needs to navigate from the main category page (in Fig. 2) to all the subcategory pages (in Fig. 3). An example with the problem of semantics consistency is illustrated as follows.

1. Locate the hyperlink DOM nodes (linked to the subcategory pages) in the browser object. Designate the first node.
2. Issue a mouse click event on the designated node.
3. Load the corresponding subcategory page into the same browser object.
Fig. 7. Data extraction on two pages with one browser only.

Fig. 8. Data extraction with browser replication.

4. Extract data from the page.
5. Go back to the main category page in the same browser object.
6. Designate the next node and repeat Steps 2-5 for the next node, until all nodes are processed.

However, after Step 5 is executed, the browser object could contain a page with different category hyperlinks (or even with hyperlinks missing), if the operation at Step 2 triggers JavaScript functions to make the change on the page. Thus, all the hyperlinks are either dislocated or gone. For example, in Fig. 7, the hyperlink to the next subcategory page al.html disappears and the hyperlink to db.html appears in a different place. Thus, subsequent data extraction becomes unpredictable. This violates the semantics of multi-way navigation that requires clicking on the original two hyperlinks, not the unpredicted new ones. Such a phenomenon is called semantic inconsistency, defined in Definition 1. The above demonstrates the following problem. Even if the whole system is deterministic, data extraction may still have unpredictable as in Fig. 7.

Definition 1. Consider the extracted data in the EDP containing locations of some DOM nodes of a page in a browser object at some time t. Assume that these DOM nodes are changed due to some events issued to the page after time t. A DE system with access to the locations of the changed DOM nodes is semantic inconsistent. Otherwise, the DE system is semantic consistent.

III. BODED LANGUAGE

In the BODE system, an XML-based script is used to instructs the controllers to generate a sequence of actions in the system, as described in [27]. The primitives of the language...
are introduced in Subsection III.1. The features of multi-way navigation are described in Subsection III.2. The issues of consistency are discussed in Subsection III.3.

1. Basics in BODED Language

The BODED language is an XML based language. The script written in BODED language is called a BODED script. The XML elements in the BODED script are called script elements. The element BODED is the outermost element that encloses the entire BODED script. This element contains two types of elements, INIT and PAGE. The INIT element specifies the initial document located at a given URL, which is specified in the attribute URL. The PAGE elements specify page scripts, which instruct the DE system to process the associated pages. The child elements of the PAGE elements are executed sequentially, one at a time.

Consider a simple example of a BODED script shown in Fig. 9. The script is to extract the title, the author and the publication of each article in article list pages as shown in Fig. 4. The INIT element indicates to allocate a new browser, and complete the following tasks: (1) perform a URL action to retrieve the initial web page located at the URL, http://bode.csie.nctu.edu.tw/de.html, and (2) process the retrieved web page by following the instructions of the page script named PaperList specified in the attribute page.

In a page script, VAR elements are used to perform extraction actions that extract data inside the browsers. In the page script PaperList in Fig. 9, the first three VAR elements are used to extract the titles, the authors, and the publications of all papers into the variables, named Title, Author, and Publication, respectively. These variables are stored in the EDP.

For expressing data extraction rules, the MBODE system uses standards XPath language [5], a WC3 standard. An XPath expression can be used to locate some DOM elements in the HTML documents or extract the content of these elements. In a BODE system, it is very important to locate DOM elements, since this allows the BODE system to perform page actions on the located elements subsequently, e.g., issuing a mouse click event.

In BODED language, the extraction rules are specified in the attribute xpath of the VAR element. According to XPath, the three variables in Fig. 9 contain pointers to the extracted data or elements. The three variables are saved into database via a plug-in mechanism named BODEDLET, which is described in more detail in Section IV.

In addition to the extraction action, using the EVENT elements specifies the other two actions, URL actions and page actions. For example, the page script AccessNext in Fig. 10 can be used to navigate a web page referenced by the last URL in Fig. 4. The VAR element extracts the hyperlink to the next bibliography page into the variable NextLink. Then, the EVENT element, the attribute type with the value URL indicates to issue a URL action with a URL contained in the variable NextLink, specified in the attribute onvar, and the attribute page indicates to use the page script Next to process the next page. The operations in both VAR and EVENT elements can be combined into a single EVENT element as in Fig. 11.

The BODED language supports the recursion of page invocations. If the value of page attribute in the EVENT element is changed from Next to AccessNext in Fig. 10, then the system repeatedly invokes the next pages until none are extracted in the event.

Now, suppose that the reference to the next article list page is a JavaScript function as shown in Fig. 5, instead of a URL. The value of the attribute type in Fig. 11 is simply modified to other external events, such as ONCLICK, as shown in Fig. 12.

In addition to ONCLICK and URL, the BODED language also includes the following event types, ONMOUSEUP, ONMOUSEDOWN, ONDBCLICK, ONKEYDOWN, ONKEYUP, ONKEYPRESS and FILL. For simplicity, in BODED script, if the resulting data extracted by an XPath expression is a string, then the value of the attribute type is, by default, URL;

Fig. 9. A BODED script to extract data from the HTML file in Fig. 4.

Fig. 10. The page script AccessNext.

Fig. 11. The simplified page script AccessNext.

Fig. 12. The PAGE AccessNext with a click event.
The attribute of the page script click event on the hyperlink in the variable SubCatLink EVENT specify navigation perform multi-way navigation. The hyperlink and use the page script a new browser for loading the subcategory page specified by MainCategory categories listed in the main category page in Fig. 2. In page scripts, the BODE system uses multi-way Data Extraction AccessNext to issue a click event to submit the form to access the next page.

For the above example, the BODED script can be written as shown in Fig. 13 (a) to simulate the user’s operation. In this script, the first EVENT element indicates to issue an external event Fill that is to fill the query field with the value “Extraction”. Then, the second EVENT element indicates to issue a click event to submit the form to access the next page. The attribute page indicates to use the page script AccessNext to process the next page.

2. Multi-way Data Extraction

The BODED language supports a feature, called multi-way navigation. In page scripts, FOREACH elements are used to specify FOREACH scripts that allow the BODE system to perform multi-way navigation.

Consider the page script MainCategory in Fig. 14. The BODE system uses MainCategory to traverse all the categories listed in the main category page in Fig. 2. In MainCategory, the VAR element extracts a set of hyperlinks to the subcategory pages and actually put these hyperlinks into the variable Subcat. The FOREACH script (inside the page script MainCategory) is executed once for each hyperlink stored in the variable Subcat. The variable named SubCatLink is created which contains the hyperlink. The EVENT element in the FOREACH script indicates to issue a click event on the hyperlink in the variable SubCatLink. The attribute page of the EVENT element indicates to allocate a new browser for loading the subcategory page specified by the hyperlink and use the page script Subcategory to process the subcategory page. Note that the value of xpath "." in the EVENT element indicates to apply the XPath rule to the DOM subtree rooted at the DOM node stored in the variable SubCatLink.

Based on the above description, the whole bibliography web site including two-level category pages, as shown in Fig. 3 and 4 can be extracted via the BODED script in Fig. 15.

All elements inside FOREACH scripts are executed sequentially, like page scripts. For example, in Fig. 16, the VAR element inside the FOREACH script extracts the URL (a hyperlink to a subcategory page), and the EVENT element then uses the URL (in the variable SubCat) to issue an URL event. The last element AllSubCatName is executed after the FOREACH script is finished.

Besides, BODED language adopts forward referencing and uses nested scoping for accessing variables. For example, in the page script in Fig. 16, the VAR element SubCatName and the EVENT element can forward reference to Subcat or
SubCatLink, but not to AllSubCatName.

In nested scoping, the last VAR element AllSubCatName can access the variable Subcat, but cannot access SubCatLink or SubCatName. Note that if the name of the variable SubCatName (inside the FOREACH element) is changed to SubCat, both variables of SubCat are different variables.

3. Consistency Issues in BODED

In the multi-way navigation operation in Fig. 14, the FOREACH element iterates through the extracted hyperlinks. When the EVENT elements are executed, the BODE system replicates browser objects as described in Fig. 8.

Browser replication can also be performed at the beginning of executing a FOREACH script. Consider the above page script MainCategory in Fig. 14. When starting to run the FOREACH script, the browser is replicated to a browser for each extracted element in the variable Subcat. Thus, no browser replications are required for the first events inside the FOREACH script.

IV. A PLUG-IN FOR BODE SYSTEM

This paper designs a plug-in in the BODE system, so that more applications can be applied by adding plug-in code, called BODEDlet. For example, save the extracted data into databases, avoid accessing redundant pages, and traverse web pages with specific rules.

In the following subsections, Subsection IV.1 describes the system architecture for supporting the BODEDlet. Subsection IV.2 describes the BODEDlets. Subsection IV.3 discusses the implementation for avoiding consistency issues. Subsection IV.4 describes the applications of BODEDlets.

1. System Architecture for BODEDlets

The architecture of our system is depicted in Fig. 17. The major differences from Fig. 6 are in the controller. The controller consists of the core controller, BODED interpreter, and BODEDlets.

The BODED script interpreter in our system initially inputs a DE script and then follows the instructions of the script to generate a sequence of messages to the core controller. The core controller receives the messages and generates a sequence of DE actions. During the execution of the DE script, the core controller may load and execute user defined BODEDlet [13]. The BODEDlet may also generate a sequence of messages to the controller for accessing the system.

The key idea of the controller is: For all script elements except FOREACH described in Subsection III.2, the core controller supports the corresponding API functions. Thus, the BODED interpreter simply translates script elements into the corresponding API functions and then invokes them. BODEDlets coded by programmers are allowed to access these API functions.

The design of the core controller follows the façade design pattern [22] by isolating both browser objects and data pool from both the BODED script interpreter and BODEDlets. An important reason is to maintain the consistency, as described in Subsection IV.3.

2. BODEDlet

BODEDlets are small programs that can be invoked by the core controller. Figure 18 illustrates an example of a page script using a BODEDlet, BODEDlet.Log, to log the operations in the page script MainCategory. Our system was implemented in C# in the .NET environment, since Web-browser objects were supported in C#. BODEDlet programs are written as classes and their executables are DLL files. Note that the system default library "BODEDLETLIB.dll" has some more useful BODEDlets. For example, save the extracted data into XML files (or databases), and remove redundant URL events.

The element BODEDLET in the page script contains at least the following two attributes.

- archive: Specify the DLL file path of BODEDlets.
- code: Specify the class name of the BODEDlet object.

BODEDLET elements in the BODED script may specify parameters through PARAM elements. The element PARAM in the page script contains at least the following two attributes.
using BODED;

namespace BODEDletLib {
    abstract class BODEDLET {
        public BODED.PageContext pageContext;
        abstract public int Execute();
    }
}

Fig. 19. The abstract class BODEDLET exported by BODE system.

using BODEDScript;

class PageContext
{
    public Collection params;
    public Variable GetVariable(String varName);
    public Variables GetVariables();
    public object Extract(String xpath, String base);
    public void SetVariable(String varName, String xpath, String base);
    public void SetVariable(String varName, object value);
    public void IssueURLEvent(String url, String pageScriptName);
    public void IssueEvent(String varName, String type, String pageScriptName, String value /* for filling*/);
    ...
}

Fig. 20. A segment of signatures of the class PageContext.

- name: Specify the name of the parameter. This attribute is similar to the declared formal parameters of a function call in java language.
- value: Specify the value of the parameter.

During the runtime of the BODE system, the BODEDlets in DLL files are loaded and executed. The classes must inherit a system defined base class BODEDLET as shown in Fig. 19. The base class BODEDLET defines an abstract method Execute and a field pageContext. The BODEDlet implementers should override and implement their own Execute method. The field pageContext provides accesses to the contexts of the pages with which the BODEDlets are associated. BODEDlets can use the field to access or perform script operations supported in the BODED language, such as VAR or EVENT.

Specifically, when a BODEDlet was invoked, the core controller loads the BODEDlet and creates a PageContext object associated the current extracted page. Then, the BODEDlet can access or operate BODED operations on the page via fields and methods the PageContext object, as listed in Fig. 20.

These fields and methods are classified into three groups. The first group is related to parameters of the BODEDLET element. params is a collection which contains the name-value pairs of the PARAM elements in a BODED script. For example, in Fig. 18, params include only one pair for PageScript. The second group is related to variables. The methods GetVariable and GetVariables can obtain the variables, which contain the data extracted in a BODED script. The methods SetVariable issue the extraction actions, which set variables to the newly extracted data. The third group is related to events. The methods IssueURLEvent and IssueEvent can issue URL or page actions. The details of these methods are described in Appendix A.

3. Consistency Issues

The consistency issues in the web data extraction include determinism and semantic consistency. To guarantee the determinism, the browser objects and the system itself should be deterministic. Our system is designed to avoid most of the determinism problem. First, a cache is included to ensure the determinism of the web pages retrieved from web servers. Second, the browser objects should be determinism. The time related functions and the synchronization of XMLHttpRequest requests that are used in AJAX should be noticed. We follow the guideline in [27] to implement the browser objects. Third, the page actions cannot be issued directly from outside the system. The controller controls all the page actions in our system.

To guarantee the semantic consistency, the BODEDlet is designed to prevent from modifying the extracted data in variables. Specifically, since the system does not allow variables with the same names, the methods SetVariable are always to create new variables, not to modify variables.

Moreover, the methods IssueEvent and IssueURLEvent in the BODEDlet will automatically perform browser replications to avoid semantic consistency problem.

4. Applications

The capability for user defined functions to interact with web pages allows our BODE system to provide web automation applications.

The BODEDlet program for BODEDlet.Log, illustrated in Fig. 21, is a class extending the base class BODEDLET. Using pageContext, the above BODEDlet can access parameters params and global data such as logFile. The data params is a collection of parameters with type Parameter, whose name and value can be accessed via methods GetName and GetValue, respectively.

In fact, the BODEDlet system can support most of functionalities supported in BODED scripts, such as those for VAR and EVENT elements. The page script in Fig. 22 uses the BODEDlet SetVariable to set a variable, exactly like the element VAR. The BODEDlet program for SetVariable is shown in Fig. 23.

Similarly, the page script in Fig. 24 uses the BODEDlet UniqueUrlEvent to issue an event, instead of using the element EVENT, in Fig. 14. The UniqueUrlEvent can also prevent from issuing the same URL events. The code of
namespace BODEDlet {
    public class Log : BODEDLET {
        override public int Execute() {
            // pageContext.logFile is a global log file.
            StreamWriter f = pageContext.logFile;
            // Log each PARAM.
            foreach (Parameter p in pageContext.params) {
                f.WriteLine("Log: " + p.GetName() + "=" + p.GetValue());
            }
            return 0; // the method is done correctly.
        }
    }
}

namespace BODEDlet {
    public class SetVariable : BODEDLET {
        override public void Execute() {
            // Get value of PARAM named XPath.
            Collection params = pageContext.params;
            String name = params.GetValue("name");
            String xpath = params.GetValue("xpath");
            // Extract a URL of sub category page
            Object url = pageContext.Extract(xpath, "");
            if (type == NULL) {
                // use default way to trigger the event.
                ...
            } else if (type == "URL") {
                if (!urlTable.Contains((String) url)) {
                    urlTable.Add((String) url, targetURL);
                    pageContext.IssueURLEvent((String)url, ps);
                } else {
                    // Issue an event according to the type.
                }
            }
        }
    }
}

namespace BODEDlet {
    public class UniqueUrlEvent : BODEDLET {
        static Hashtable urlTable = new Hashtable();
        override public void Execute() {
            // Get value of PARAM named XPath.
            Collection params = pageContext.params;
            String name = params.GetValue("name");
            String xpath = params.GetValue("xpath");
            // Extract a URL of sub category page
            Object url = pageContext.Extract(xpath, "");
            if (type == NULL) {
                // use default way to trigger the event.
                ...
            } else if (type == "URL") {
                if (!urlTable.Contains((String) url)) {
                    urlTable.Add((String) url, targetURL);
                    pageContext.IssueURLEvent((String)url, ps);
                } else {
                    // Issue an event according to the type.
                }
            }
        }
    }
}

namespace BODEDlet {
    public class SetVariable : BODEDLET {
        override public void Execute() {
            // Get value of PARAM named XPath.
            Collection params = pageContext.params;
            String name = params.GetValue("name");
            String xpath = params.GetValue("xpath");
            // Extract a URL of sub category page
            Object url = pageContext.Extract(xpath, "");
            if (type == NULL) {
                // use default way to trigger the event.
                ...
            } else if (type == "URL") {
                if (!urlTable.Contains((String) url)) {
                    urlTable.Add((String) url, targetURL);
                    pageContext.IssueURLEvent((String)url, ps);
                } else {
                    // Issue an event according to the type.
                }
            }
        }
    }
}

namespace BODEDlet {
    public class UniqueUrlEvent : BODEDLET {
        static Hashtable urlTable = new Hashtable();
        override public void Execute() {
            // Get value of PARAM named XPath.
            Collection params = pageContext.params;
            String name = params.GetValue("name");
            String xpath = params.GetValue("xpath");
            // Extract a URL of sub category page
            Object url = pageContext.Extract(xpath, "");
            if (type == NULL) {
                // use default way to trigger the event.
                ...
            } else if (type == "URL") {
                if (!urlTable.Contains((String) url)) {
                    urlTable.Add((String) url, targetURL);
                    pageContext.IssueURLEvent((String)url, ps);
                } else {
                    // Issue an event according to the type.
                }
            }
        }
    }
}

V. VISUALIZATION TOOL

Although our system allows users to write the BODED scripts, the BODED scripts may still be difficult to write directly. Therefore, the BODE system also provides users with a WYSIWYG visualization tool to reduce the time taken to develop BODED scripts.

Figure 26 shows the visualization tool of the BODE system. This tool contains five regions. The web page region displays the web pages. The XPath region displays the XPath rules used. The BODED script region displays elements of the BODED script hierarchically. The DOM tree region displays the DOM tree representing the web page displayed in the current web page region. Finally, the properties region: displays the attribute properties of the element selected in the BODED script region.

The visualization tool of our system has two modes, the editing mode and the execution mode. In the editing mode,
users edit BODED scripts using a WYSIWYG interface.

For data extraction, users can directly click on the data item displayed in the web page region. The DOM element corresponding to the clicked item is highlighted and its XPath is then displayed in the XPath region. Then the XPath can be saved into a `VAR` element.

When starting to perform data extraction, users simply switch to the execution mode. The edited BODED script is executed, and the extracted data are output to a new window.

According to our experiments, one BODED script could be written within 30 minutes with the help of the visualization tool, but takes about 360 minutes without the help of the visualization tool.

VI. EXPERIMENTS

In this section, we illustrate the plug-in mechanism of the BODE system by automating the playing of Connect6 [24] in Little Golem [14]. Connect6 is a kind of six-in-a-row game introduced by Wu [25] in 2005. Little Golem, a web-based game system, is one of popular game sites for Connect6.

The team led by Wu also developed a Connect6 program, named NCTU6, winning the gold in the 11th and 13th computer Olympiad [26, 28]. In order to compare the strength of NCTU6 with those of human players, they collaborated with us based on the BODE system to play their program with other players automatically in Little Golem.

Without the BODE system, playing games in Little Golem is operated as follows.

1. In the login page, login with user name and password.
2. Go to the page that shows the list of the current games. The game list is shown in the table of “Games where it is your turn”, as illustrated in Fig. 27.
3. For each game in the game list, access the page corresponding to the game, as shown in Fig. 28, and then do the next step.
4. Place a move in the board (in Block 2). In case of NCTU6, we let NCTU6 make a move by inputting the game history (in Block 1).

In order to let the BODE system play automatically, we first make use of the visualization tool (described in Section V) to create a BODED script by following the above four steps, except for that we randomly pick one move at Step 4. The element `<Page>` corresponding to Step 4 is shown in Fig. 29(a).

Then, in order to plug-in the program NCTU6 to play, we modify this `<Page>` as shown in Fig. 29(b). The element `<BODEDLET>` indicates to invoke a BODEDLet LGPlugin.dll which parses the game history from the variables `History` and `BoardRange`, then execute the NCTU6 program by inputting the history data, and finally make a click operation.
Fig. 28. The web page of one game in Little Golem.

(a)

(b)

Fig. 29. Connect6.xml.

VII. SUMMARY

A design of the BODE system was proposed which includes a simple script language, a plug-in, and a visualization tool. The contributions of this paper are summarized as follows:

- The scripting language, named the BODED (Browser-Oriented Data Extraction Description), is defined to instruct the BODE system to extract data.
- A plug-in, called BODEDlet is presented to extend the functionality of the data extraction system. At the same time, the consistency issues of BODE systems are also solved.
- A visualization tool is implemented to facilitate data extraction in BODE. In practice, we spent 30 minutes in designing data extraction for Yahoo! Web site, but 360 minutes without this tool.

An illustration of automating the playing of Connect6 [24] in Little Golem [14] is experimented to demonstrate the plug-in mechanism of the BODE system.

ACKNOWLEDGMENTS

The authors would like to thank the National Science Council of the Republic of China, Taiwan for financially supporting this research under Contract Numbers, NSC 94-2213-E-009-028, and NSC 95-2221-E-029-033.

APPENDIX A.

The methods of PageContext in Fig. 20 are summarized as follows.

- GetVariables(): This method gets all the variables that are accessible in the current page script.
- GetVariable(varName): This method gets the variable with name specified in varName.
- Extract(xpath,base): This method extracts data from the DOM tree rooted at the node pointed by a variable named base, and return the extracted data back. Currently, we only support XPath as the extraction rule.
- SetVariable(varName,object): This method stores the object object into the variable with the name specified by varName.
- SetVariable(varName,xpath,base): This method uses Extract method to extract data and then SetVariable(varName,object) to store the extracted data into the variable.
- IssueUrlEvent(url,pageScriptName): This method issues a URL event (with the URL specified in url) in the current page context named pageScriptName.
- IssueEvent(varName,type,pageScriptName, value): This method issues an external event in the current page context named pageScriptName. The event is issued on the DOM node in the variable whose name is specified by varName. The event type is specified in type. The value is used for the events that require parameters. For example, for the fill event, the filling string is specified in this parameter.

REFERENCES