Abstract. Recent innovations in Web technologies enable end-users to easily use various services through their Web browsers. Examples of such services include various kinds of database service and various sorts of analysis tools. This evolution in Web applications causes difficulty in reusing functions embedded in Web applications. The purpose of this study is to develop an environment where even an end-user can easily wrap complicated Web applications to reuse their embedded functions. In this paper, we propose a framework that enables end-users to wrap Web applications through instruction by demonstration. IntelligentPad architecture enables end-users to dynamically define functional linkages among wrapped Web applications through drag & drop and paste operations.

1 Introduction

During the last several years, the Web has evolved into a world-wide information pool where people can share intellectual resources. Information publishers represent their intellectual resources in the form of semi-structured documents on the Web. Hyperlinks among documents allow users to navigate from one document to another related document. An input-form in an HTML document enables users to submit data to a server-side program, which is called Web application. A Web application is an application program that has an HTML-based front-end for users to utilize remote services. Today, many companies and researchers provide Web applications, such as various kinds of database services and various sorts of analysis tools.

Recent innovations in Web technologies enable end-users to easily use various services through their Web browsers. CGI (Common Gateway Interface), ASPs (Active Server Pages), Java’s Servlet, JSP (JavaServer Pages) and PHP (PHP: Hypertext Preprocessor) are all running at server-sides to dynamically provide HTML-based front-end interfaces to users. Some sites use frames to present documents with multiple embedded document views, or to popup certain information aside. Some sites use cookies to maintain a session.

The evolution of Web application technologies causes various difficulties in reusing embedded functions in Web applications. For example, if a user wants to send an output from one Web application to an input-form of another Web application, he or she needs to repeat complicated and sometimes tedious copy-and-paste operations. Firstly, he must input some data on the first Web application and navigate through result pages to obtain some result data. Secondly, he must copy a part of the result data using his mouse, and paste this copy into the input-form on the second Web application. Finally he must submit the input-form and navigate through the result pages to obtain some result data from the second Web application. The user must repeat these three steps, if he wants to apply the same processing for different inputs. Currently, in order to automate this process, users must write a program called a wrapper in a programming
language such as Perl or Java, or based on a specialized framework such as SOAP, to perform the required processing. Maintenance of a large number of wrappers is not easy in practice. It is time consuming to rewrite a wrapper program every time the syntax of the Web application page changes.

From this point of view, we propose a new framework that enables users to wrap arbitrary Web applications through instruction by demonstration. IntelligentPad architecture enables end-users to define functional linkages among wrapped Web applications through drag & drop and paste operations. In our previous works[14, 16], we presented the simplest version of our wrapping method, and discussed its application to mobile computing[16]. In [17], the whole strategy for the wrapping and functional linkage using IntelligentPad architecture was given. In this paper, we focus on Web application wrapping through instruction by demonstration.

This paper is organized as follows. In Section 2, we address some fundamental requirements concerning the end-users’ reuse of Web applications. Our methodology for the Web application wrapping through instruction by demonstration is described in Section 3. Details of the IntelligentPad architecture are presented in Section 4. Related work is discussed in Section 5, and we conclude in Section 6 with our future research plan.

2 Reuse of Web Applications : Requirements and Our Approach

In bioinformatics, for example, there are already many different kinds of database services, analysis services, and related reference information services; most of them are available as Web applications. However they are hard to interoperate with each other. This has two reasons. Different Web applications use different data formats. In addition, there is no way on the client side to connect the output of one Web application to the input form of another Web application other than making a copy of the appropriate output text portion on the source page and pasting it in the input form of the target page. While SOAP allows you to write a program to functionally integrate more than one Web service, it is hard to use for non-programmers.

Before we describe our approach to solve this problem, we address some fundamental
requirements concerning the end-users’ reuse of Web applications.

A framework where end-users can reuse Web applications requires the following capabilities.

1. Easy specification of input and output portions of Web applications to reuse embedded functions in them.

2. Easy definition of functional linkage between Web applications to compose a new integrated application.

For the first requirement, we will propose a Web application wrapping method based on the instruction by demonstrating an example sequence of user operations. Intelligent-Pad architecture enables end-users to define functional linkage between wrapped Web applications through drag & drop and paste operations. The use of IntelligentPad technologies achieve the second capability.

In figure 1, we summarize our approach of reusing functions embedded in behind Web applications. Users can create wrapper pads for Web applications based on the programming by demonstration (PBD). A WebNavigationWrapperPad has the facility to record and replay operations that are performed by a user. During its definition phase, WebNavigationWrapperPad records user’s operations on the Web browser interface. Users can define data I/O ports called slots on intermediate pages that are returned by Web applications. Parameters in the recorded operations, such as values submitted to forms, can be modified through corresponding input slots. WebNavigationWrapperPad automates navigation using the modified values. Users can also define arbitrary document portions in pages that are returned by the Web application to work as output slots.

Using IntelligentPad architecture, users can functionally combine these wrapped Web applications by connecting slots through the use of drag-and-drop and paste operations. Users can also reuse wrapped Web applications in collaboration with wrapped local legacy applications.

3 Web Application Wrapping by Demonstration

In this section we present our method of the Web application wrapping based on instructions by demonstration. Programming by Demonstration (PBD) enables users to construct a script program by simply performing actions in the user interface. With PBD, the user instructs the system to watch what I do, and a PBD system creates generalized programs from the recorded actions.

3.1 Modeling Navigation within Web Applications

Many Web applications use more than one page to output and input data to and from users. For example, the Web application shown in Figure 2 uses three pages for its input and output. On the first page, this application requests users to input data. The second page displays a list of result items. Users may select one of these items and click its anchor to jump to the third page, where they will get result data from this application.

Figure 3 shows a site-specific navigation model for PubMed paper search, which answers queries of the form “Search PubMed for words w”.
In order to simplify the description of our method, we first model the Web application as a directed graph. We denote a graph of a Web application $WA$ by $G(WA)$. For a Web application $WA$, graph $G(WA)$ contains a node for each page $p$ in $WA$. In $G(WA)$, there exists a directed edge $(p, q)$ labeled with an operation $op$, if and only if $op$ on $p$ causes a jump to the page $q$ in $WA$. Examples of operations includes opening a page specified by a url, navigation along a hyperlink, and submitting some data to a form.

We denote a navigation in a Web application as follows:

$$p_0 \xrightarrow{op_0} p_1 \xrightarrow{op_2} \cdots \xrightarrow{op_{k-1}} p_k,$$

where $p_0, p_1, \ldots, p_k$ are Web pages, and each $op_i$ is a user-operation at page $p_i$ to jump to the next page $p_{i+1}$. Formally, a navigation in a Web application $WA$ is a path in the graph $G(WA)$.

We call a sequence of user-operations $op_0, \ldots, op_{k-1}$ a navigation path. By a navigation path $op_0, \ldots, op_{k-1}$, we identify the Web page $p_k$ in a Web application $WA$ as long as $op_0, \ldots, op_{k-1}$ is a path in $G(WA)$.

$$p_k = op_0.\cdots.op_{k-1}.$$  

We may fetch the page $p_k$ with a URL directly. In general, however, every pages is not accessible with URLs. For example, Web pages that are returned as a result of POST are not specified by URLs.

By modifying an operator $op_i$ with $op'_i$, we get another Web page $p'_k$ of $WA$ through the same navigation from $p'_{i+1}$ to $p'_k$.

$$p_0 \xrightarrow{op_0} p_1 \xrightarrow{op'_1} \cdots \xrightarrow{op'_{i-1}} p_i \xrightarrow{op'_i} p'_{i+1} \xrightarrow{op'_{i+1}} p'_{i+2} \cdots \xrightarrow{op'_{k-1}} p'_k,$$

Example of modifications contains substitution of input-value for an input-form on $p_{i-1}$.

The page $p'_k$ that are obtained by the modified navigation-path is represented by

$$p'_k = op_0.\cdots.op_{i-1}.op'_i.\cdots.op_{k-1}.$$
3.2 User Operations within Web Pages

A user-operation \( op \) is an operation that users can perform with ordinary Web browsers such as Netscape Navigator or Internet Explorer. Examples of user operations include open of a specified url, click of an anchor, and submission of some data to a form.

In our framework, we use the following 5 operations.

\[
\text{user-operation} ::= \begin{align*}
\text{Open}(url) \\
\text{Click}(\text{anchor-elm}) \\
\text{Set}(\text{input-elm}, \text{value}) \\
\text{Submit}(\text{form-elm}) \\
\text{Back}()
\end{align*}
\]

An Open\((url)\) operation opens the page specified by given URL. A Click\((\text{anchor-elm})\) follows hyperlink with the anchor-element \( \text{anchor-elm} \). A Set\((\text{input-elm}, \text{value})\) sets the given value to the specified input-form. and a Submit\((\text{form-elm})\) operation submits the specified form to the appropriate Web Server to jump to the next page.

A navigation path is a sequence of user operations. Using a navigation path, we can represent how to access a Web page.

\[
\text{navigation-path} ::= \begin{align*}
\text{user-operation} \\
\text{user-operation} \cdot \text{navigation-path}
\end{align*}
\]
3.3 Identification of HTML-Elements

There are many ways to characterize the position of specific HTML-elements in a DOM tree.

Many researchers have worked on developing robust wrapping methods of Web applications\[3, 25, 19\]. Some wrapping methods have robustness for changes in formatting. From the view point of computational learning theory, it is impossible to deal with every kinds of formatting change. We do not focus on such robustness in this paper. We focus on how instantaneously users can create wrappers of Web applications.

We use HTML-paths to specify elements in a DOM tree. The HTML-path expression is the specialization of the XPath expression\[30\].

An HTML-path is a concatenation of node identifiers along a path from the root to the specified element. Each element identifier consists of a tag name and an index $i$, where this node is the $i$th sibling that has the same tag name. We define the syntax of HTML-paths as follows:

$$HTML-path ::= \text{tagname}[i]$$  
$$| \text{HTML-path/tagname}[i].$$

Figure 4 shows an HTML document with its DOM tree representation of a Web application. Using HTML-paths, we can access texts in a DOM tree.
For example, The circled portion in the document in Figure 4 corresponds to the circled node whose HTML-path is

$$\text{HTML}[1]/\text{BODY}[1]/\text{FORM}[1]/\text{INPUT}[1].$$

This is the HTML-path of an input element of this Web application.

To access the detail of an HTML element, we use the following two extensions added to the path-expression.

- $\text{HTML-path}/@\text{attribute-name}$
- $\text{HTML-path}/\text{text()}$

The function $@\text{attribute-name}$ selects an attribute value named with the specified name. We can set and get the value of the attribute. The function $\text{text()}$ selects a text value of the element. For example, consider the DOM tree in Figure 4. The attribute value identified by “$\text{HTML}[1]/\text{BODY}[1]/\text{A}[1]/@\text{href}$” is the string:

“http://www.meme.hokudai.ac.jp”.

The text value pointed by “$\text{HTML}[1]/\text{BODY}[1]/\text{A}[1]/\text{text()}$” is the string “Anchor Text”.

We will extend HTML-paths with navigation-paths. *Operational HTML-paths* are defined as follows.

$$\text{Operational HTML-path} ::= \text{navigation-path}/\text{HTML-path}$$

Using an operational HTML-path, we can extract data in a Web page that is not specified by a particular url address, and we can wrap Web application into a component.
3.4 Operation-based Wrapper of a Web Application

WebNavigationWrapperPad is a general wrapper for a Web Application. Figure 5 shows an abstract architecture of a WebNavigationWrapperPad. A WebNavigationWrapperPad is a kind of Web browser. Its facility to render Web documents is implemented by wrapping Internet Explorer’s API[24].

A WebNavigationWrapperPad has user-defined HTML-node slots as its connection interface. These slots are named with operational HTML-paths. The view of this pad shows current page that is returned by a Web application. The value of an HTML-node slot named with an operational HTML-path $p$ is the text value of the HTML-node at $p$.

In its recording phase, WebNavigationWrapperPad records user’s operations into a navigation-path.

If the value of a slot of an input-form is changed, then the WebNavigationWrapperPad modifies the recorded navigation-path that the user performed previously, and automatically replays it. While the wrapper replays the modified navigation-path, the wrapper updates the values of slots defined within the loaded pages (Figure 6).

As we describe in Section 4, IntelligentPad architecture allows users to define functional linkage among wrapped Web applications through slot-connection.

3.5 User Interface for the Wrapping

To define a wrapper of a Web application, users may just open the Web page they want to wrap using a WebNavigationWrapperPad. When a user performs a sequence of operations on a WebNavigationWrapperPad to browse the Web, the WebNavigationWrapperPad records these operations into a navigation-path. If a user defines an HTML-node to work as a slot, the pad associates this slot with the operational HTML-path. The values of the user-specified slots are accessed through navigation-paths and its HTML-paths on the DOM-trees.

To hide HTML-paths from users, the system calculates HTML-path expressions...
from users’ operations. Using his mouse, users can directly specify any HTML-node to work as a slot.

There are two ways for users to specify an HTML-node to work as a slot:

- specify an HTML element to work as a slot,
- specify a text portion to work as a slot.

If a user clicks with the right mouse button on a region that he wants to use as a slot, a popup menu will be shown.

Selecting “HTML as Slot” in the popup menu, the user can use the HTML-element at this location as a slot. Let pagepath be the navigation-path of the current page, and htmlpath be the HTML-path of the element (Figure 7).

1. If the element is a textinput element or textarea element, WebNavigationWrapperPad installs a slot named pagepath/htmlpath/@value.
2. If the element is an anchor element, WebNavigationWrapperPad installs a slot named pagepath/htmlpath/@href.

Selecting “Text as Slot” in the popup menu, the user can use the text string contained in selected element to work as a slot. Let p be the HTML-path of the selected HTML-element. The WebNavigationWrapperPad installs a slot named p/text().

In Figure 7, the user selects a text input element to use this as a text input slot. WebNavigationWrapperPad installed the slot named

\[ op_0,op_1/HTML[1]/BODY[1]/FORM[1]/INPUT[1]/@value, \]

where \( op_0,op_1 \) is the current navigation-path of the page shown on this pad.

Figure 7: User interface for the slot definition.
In Figure 8, a user creates a wrapper pad that wraps PubMed service. Firstly, the user may open the target NCBI page on a WebNavigationWrapperPad. Secondly, using his mouse, he may specify regions that he wants to work as slots. Then the slots will appear as sunk shaded regions on this page. The user may keep the background Web page either visible or make it invisible. Thirdly, the user may embed other pads, such as TextPads and WebBrowserPad, into these slots. Finally, the user may arbitrarily relocate and resize the embedded pads to design their layout. If the user inputs some data to the TextPad that is embedded in the input-form slot, then the WebNavigationWrapperPad will send this form value to the server. As a result, the WebBrowserPad shows a new document that is retrieved by PubMed.

4 Re-using Web Application through Functional Linkage

In this section, we present how to visually define functional linkage among Web Applications. We apply the IntelligentPad architecture to this problem.

IntelligentPad architecture allows users to combine media objects (called pads), such as multimedia documents and application programs, through their view integration. Each pad has slots as data I/O ports. Through drag-and-drop and paste operations, users can connect one pad to a slot of another pad. This operation simultaneously creates both a composite view and a functional linkage through a slot connection.
Figure 9: User interface and connection interface. If a user paste $P_1$ on a slot of $P_2$, primary slot of $P_1$ is connected to the slot of $P_2$.

4.1 IntelligentPad Architecture

IntelligentPad[29] represents each component as a pad, a sheet of paper on the screen. A pad can be pasted on another pad to define both a physical containment relationship and a functional linkage between them. When a pad $P_1$ is pasted on another pad $P_2$, the pad $P_1$ becomes a child of $P_2$, and $P_2$ becomes the parent of $P_1$. No pad may have more than one parent pad. Pads can be pasted together to define various multimedia documents and application tools. Unless otherwise specified, composite pads are always decomposable and re-editable.

Each pad has both a standard user interface and a standard connection interface. The user interface of a pad has a card like view on the screen and a standard set of operations like ‘move’, ‘resize’, ‘copy’, ‘paste’, and ‘peel’. Users can easily replicate any pad, paste a pad onto another, and peel a pad off a composite pad. Pads are decomposable persistent objects. You can easily decompose any composite pad by simply peeling off the primitive or composite pad from its parent pad. As its connection interface, each pad provides a list of slots that work as connection jacks of an AV-system component, and a single connection to a slot of its parent pad(Figure9).

To set up data connection between pads, IntelligentPad uses three standard messages, set, gimme and update. We show an outline of these three messages in Table 1.

<table>
<thead>
<tr>
<th>Message</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>set slotname value</td>
<td>a child sets the specified value to its parent's slot</td>
</tr>
<tr>
<td>gimme slotname</td>
<td>a child requests its parent to return the value of its specified slot</td>
</tr>
<tr>
<td>update</td>
<td>a parent notifies its children that some slot value has been changed</td>
</tr>
</tbody>
</table>

Table 1: A summary of three standard messages.

Each pad is embedded in one parent at most with its connection to one of the parent
4.2 Functional Linkage of Wrapped Web Applications through View Integration

A pad that wraps a Web application provides slots for some of the original’s input forms and output text strings. Since wrapped Web applications are pads, you may combine wrapped Web applications together through drag&drop paste operations. Through data linkage between slots specified by a user, these wrapped Web applications cooperate with each other (Figure 11).

Figure 12 shows a composite tool that integrates DDBJ’s Blast homology search [9], GenBank Report service[8] and PubMed’s[23] paper reference service. Blast service allows us to input a sample DNA sequence, and outputs genes with similar DNA sequences. We have specified the input form and the accession number of the first candidate sequence to work as slots. The accession number works as an anchor linking to a GenBank Report Web page containing the detail information about this gene. Its corresponding slot contains the URL to the target GenBank Report page. We have pasted a WebNavigationWrapperPad with its connection to this second slot. As a result, this child WebNavigationWrapperPad shows the corresponding GenBank Report page. This page contains bibliographic information about the related research papers. We have visually specified the title portion of the first research paper to work as a slot of this pad. We have also wrapped the PubMed service with its input form working as a slot. PubMed service returns a list of full documents that contains given keywords. We have made this slot work as the primary slot. By pasting this wrapped PubMed service on the WebNavigationWrapperPad showing a GenBank Report page with its
connection to the title slot, you will obtain a composite tool that functionally integrates these three services.

The new integrated application in Figure 12 was composed within several minutes by a biologist who has no programming expertise.

5 Related Works

There are lots of preceding research studies on wrapping of Web pages by demonstration. However, there are few research studies that allow end-users to wrap Web applications by defining input and output port for them.

WebVCR[1] provides sophisticated interface to record and replay users’ actions. It replays a series of browsing steps in “smart bookmarks”, which are shortcuts to Web contents that require multiple steps to be retrieved. Creation and update of smart bookmarks is a simple process involving only the usual browsing actions by the user. However, the system does not support the definition of I/O ports for Web applications. For example, end-users could not modify the parameters for an input-form.

Bauer and Dengler[3] have also introduced a PBD(Programming by Demonstrations) method in which even naive users can configure their own Web based information services satisfying their individual information needs. They have implemented the method into InfoBeans[2]. By accessing an InfoBox with an ordinary Web browser, users can wrap Web applications. By connecting channels among InfoBeans on the InfoBox, users can also integrate them functionally together. However, it seems difficult for users to reuse a part of composite Web applications defined by other users.

W4F[26], which is semi-automatic wrapper generator, provides a GUI support tool to define an extraction. The system creates a wrapper class written in Java from user’s demonstration. To use this wrapper class, users need to write program codes. DebyE[11] provides more powerful GUI support tool for the wrapping of Web applications. WbyE stores the extracted text portions in XML repository. Users have to use another XML tool to combine extracted data from Web applications. LExIKON[12] learns an underlying relation among objects within a Web page from a user-specified ordered set.
Figure 12: Dynamic gene annotation with visually wrapped Web applications. This was composed within several minutes by a biologist who has no programming expertise.

of text strings. There is no GUI support tool for the join of two extracted relations. WebView[10] allows us to define customized views of Web contents. However it seems difficult for end-users to create a new view that integrates different Web applications.

6 Concluding Remarks

In this paper, we have proposed a Web application wrapping method based on a sequence of user operations on the target Web application. This framework is based on the IntelligentPad architecture. Users can visually wrap Web applications into visual components, and visually combine them together to define functional linkages among them. Users can also visually define functional linkages between wrapped Web applications and such local tools in pad forms as chart drawing and spreadsheet tools to compose a single integrated tool.

For future work, we are planning to introduce facilities to support debugging. We have not implemented debugging tools such as error notification for defined wrappers. WebNavigationWrapperPad may detect format changes when the extraction is failed. Reasonable heuristics can be used to detect format-changes.

We also need to discuss solution to the copyright problem. Copyright policies have been reconsidered and modified every time when people introduced new media technologies. Whenever a new media technology is introduced, the consensus on new copyright policies gradually coevolves with new copyright protection and/or license management technologies. We have been, and are observing such coevolution of new policies with the Web technologies. Some have established closed services on the Web that are exclusive to their members, while others have established a closed network, such as the I-mode cellular phone network in Japan by NTT DoCoMo. Many other types of license and account management are currently tried on the Web. The same situation will occur for
our technologies.

References


