The technology that integrates various types of Web contents to build a new Web application through end-user programming is widely used nowadays. However, the Web contents do not have a uniform interface for accessing the data and computation. Most of the general Web users access information on the Web through applications until now. Hence, designing a uniform and flexible programmatic interface for integration of different Web contents is unavoidable. In this paper, we propose an approach that can be used to analyze Web applications automatically and reuse the information of Web applications through the programmatic interface we designed. Our approach can support the flexible integration of Web applications, Web services and Web feeds. In our experiments, we use a large number of Web pages from different types of Web applications and achieve the integration by the proposed programmatic interfaces. The experimental results show that our approach brings to the end-users a flexible and user-friendly programming environment.

**Keywords:** Programmatic interface; integration; Web application; Web service; partial information extraction; end-user programming.

1. Introduction

Information on the Web is supplied in different styles. From the viewpoint that end-users get information from the Web, there are mainly two styles, Web application
and Web service. Web mashups are Web applications developed using existing contents and services available online [27], which bring end-users the furthest possibility to reuse the information on the Web. Research [29] shows that half of the survey respondents would like to build mashup applications. A majority of the survey respondents consider themselves non-programmers. Being distinguished from the professional programmers, end-user programmers are people who have computational needs, but do not want to become professional programmers [18]. They prefer to use third-party technologies with simple interface and little programming to meet their needs. End-user programmers outnumber professional programmers now, about 90 million [20]. The respondents’ experience with Web technologies are on the low end except HTML in research [29].

Based on the above facts, we found that there are two technical obstacles for end-user programmers to reuse the information on the Web. The first one is that different services have different interfaces. For example, client-side programming is necessary for SOAP type services, while REST type services could be accessed through HTTP request. The other one is that most general Web applications do not have open API. For example, we can get the basic information of a selected country from the BBC Country Profiles [31], but this Website does not provide this functionality as a Web service. Furthermore, with the universal usage of client-side scripts on the Web pages, how to reuse the parts created by client-side scripts is also very important.

In this paper, we propose an approach which supports the mashup developers to focus on mashup scenarios when they build mashup applications instead of paying too much attention to getting information from the Web. We wrapped several functions into a Java-based library, including two functions that can extract any part(s) from general Web applications. The extraction will be done according to the given parameters. End-user programmers can get both static and dynamic information from general Web applications easily by calling different functions. Here we use the static information to represent the information shown on the Web page that can be found directly in the HTML source file, such as the text between HTML code’s span tag. And we use the dynamic information to represent the information which is shown on the Web page but cannot be found directly in the HTML source file, such as the HTML code generated by client-side scripts dynamically. We assume the users of our approach are those who have the knowledge of building a simple Web application with HTML and JavaScript. With this approach, end-user programmers can build mashup applications easily with the information from the Web.

The reason that we design this interface library for end-user programming is due to several problems based on the analysis.

(1) If a developer (either end-user programmer or professional programmer) wants to reuse the functionality of an existing Web application in his/her software development or reuse a component partially, he/she usually has to program from scratch.
(2) GUI-based methods, such as Yahoo Pipes [40], are dependent on the specific platforms, systems or browsers. For example, the components generated by Yahoo Pipes cannot be used by other systems.

(3) Some functionalities cannot be generated by current non-programming methods, such as complex or user-customized functionalities. For example, certain kinds of component (security related) can only be reused based on the source code or from data flow level.

The rest of this paper is structured as follows. In the next section we present an overview of the related work. In Sec. 3, we explain the partial information extraction method we proposed for the easy integration of information from Web applications with Web service APIs. In Sec. 4, we illuminate our approach and the structure of the Java-based library, which is based on the partial information extraction method. In Sec. 5, we give the details of our implementation and an evaluation of our approach. Finally, in Sec. 6, we conclude our approach and discuss our future work.

2. Related Work

The techniques that can bring developers an environment to create mashup applications easily and efficiently have developed quickly. This accelerated the actual development of mashup applications. According to the ProgrammableWeb [35], there are more than 7,050 mashup applications by April 2013 and there are about three new mashup applications generated every day on average. Daniel et al. [3] also indicate that end-users\(^a\) will play more roles instead of waiting for the IT experts’ support when developing mashup applications.

As another point of view, the Web sources for building mashup applications can be Web applications and Web services. In the case of using Web services, programming to access the service APIs is a natural and widely used method. However, in the case of Web applications, there is no uniform programmatic interface that could be used to access the data and functionalities. In this paper, we focus on how to integrate the information of Web applications with Web services through end-user programming easily and flexibly. The existing methods that are used to build mashup applications, which use information from Web applications, can be classified into three groups.

- Build the mashup application through programming completely, including the process that extracts information from Web applications. This could be thought of as the most flexible method. However, research on end-user programming has found that programming is difficult for novices for a number of reasons [13]. They must enter syntactically correct code, find appropriate operations, and debug any programming errors that may arise [12].

\(^a\)We use end-user to represent end-user programmer in the following sections.
Need no programming, but the information that can be treated is limited to certain format, or the layout of the mashup application is limited. A well-known example is Yahoo Pipes, which offers a platform and provides a number of user-configurable modules to build mashup applications. Developers can use Web feeds, Web services and retrieved text from Web pages by configuring modules to do integration. It does not deal with the dynamic contents, and it is not easy to retrieve text from Web pages in Yahoo Pipes. Mixup [28] is a development and runtime environment for UI integration [4]. It can help developers to quickly build complex interfaces for easy integration by using the Web service APIs available. Mashup Feeds [21] and WMSL [19] support integrated Web services as continuous queries. They create new services by connecting the Web services using join, select and map operations. Like these methods, IBM QEDWiki (IBM Mashup Center) [2] and some other service-based methods [16, 25, 30] are also limited to the combination of existing Web services, Web feeds or generated Web components.

"Data Mashup Platform” proposed in [5] can use four kinds of data resources (database, restful sensor resource, web service and local file) and the predefined mashup logics to generate “mashup services”. Developers can configure the resources and achieve the mashup services through data mashup editor that the authors provided. The Web sources it can use do not include the information that is published in Web application format. This approach does not provide a user interface for designing mashup applications.

Metabrain [23] focuses on the access of implicit information from the World Wide Web, which meanwhile designs an interface for visualization. It extracts information by using search engines and three kinds of techniques, which are the numbers of results, Lexico-Syntactic Patterns and Term Co-Occurrence. It seems to be more effective when answering questions, because that it can tap implicit knowledge from the web.

Nachouki and Quafafou [17] present their methods for mashuping Web data sources and services based on semantic queries. They use wrapper for information extraction from the Web. The extracted static data sources could be used together with Web services. Meanwhile, they are concerned with the problem of semantic reconciliation of multi-data sources in the Web mashing process.

Even our previous work [9, 6] has the advantage that allows developers to build mashup applications with any part(s) of general Web applications without programming. It still faces the challenges that the layout is fixed and the Web sources are limited to Web applications.

Cao et al. [1] point the problem that end-users are not just coders but also designers. They will not be content with the settled layouts when building mashup applications. Thus, the approaches that extract information from Web applications automatically to help end-users building mashup applications occurred. These approaches could serve as keeping the flexibility of design and saving the time of development. The approach we propose in this paper should be included in
this group. The following approaches are focused on how to use the information of Web applications to make integration with Web services through end-user programming.

Dapp Factory [32] can wrap information extracted from Web pages or RSS [36] into other data format, such as Dapp XML, RSS feed, Google Gadget and so on. Research [8] also proposes a Web information extraction method to realize the integration of Web application and designs a Java-based class package with three functions. However, both approaches cannot extract dynamic Web content from Web pages, for example, the clock part of Localtimes.info [34].

Marmite [26] is an end-user programming tool for creating Web-based mashups and implemented as a Firefox plug-in using JavaScript and XML User Interface Language. It supports a data flow architecture, where data is processed by a series of operators in a manner similar to Unix pipes. It can use a basic screen-scraping operator to extract the content from Web pages and output the data in different formats. However, the action that end-users can take is limited to the operators supplied.

Vegemite [14] aims to address the building of ad-hoc mashups that key artifact is data collection. It is an end-user programming-by-demonstration method. Vegemite works based on the data-table extracted from the Web. The extraction algorithm is designed to get data from a Web page according to the example element that the end-user selected, which is similar to Sifter [10] and Karma [24].

Some approaches are proposed to construct Web services based on the Web applications for end-user programming. Pollock [15] can generate Web services from the form-based query interfaces of a Web site. It generates wrappers using XWrap, WSDL files, Web site-related information and translates the SOAP-based XML message into HTTP-based URL message orderly. Finally, it publishes the details of the virtual Web service into UDDI. However, this approach needs to generate wrappers for every Web site. H2W [22] generates Web services based on the page-transition of existing Web applications instead of the data structures behind them. Research [11] proposes a solution of Web Service Gateway to “wrap” existing Web sites into Web services, which can inherit all features from the sites while can be enriched with other Web service features. Research [7] also provides a Web services generation method based on information extracted from existing Web applications. These approaches take a great deal of time and skills to create such services in a proxy server run between the target Web applications and end-users, and it is extremely unlikely that the constructed Web services can support all the needs of end-users in building mashup applications.

Different from the above approaches, in this paper, we propose a partial information extraction approach for programmers to integrate both dynamic and static

\[\text{Now it has become a part of Yahoo!}\]
Web contents of general Web applications easily. Compared with the existing work, our approach has the following features:

(1) It can use the information from general Web applications in the same way as Web service APIs, which means a uniform interface is available for end-users to reuse the information on the Web easily.

(2) End-users can add customized items easily through our approach in mashup applications.

(3) Only a small number of codes are needed for end-users to build mashup applications with any part(s) (both static and dynamic contents) of the Web pages and service APIs.

3. Partial Information Extraction from Web Applications

Developers can use a Web service in the format that sends a request to the Web service API with the parameters in the required format and receives the result as a response. However, the Web applications are designed to respond to the actions that users make through Web pages. In fact, for information interaction, Web applications usually provide some request-submit functionalities for users. For example, search engine applications provide a text input field on the Web page for users to input the query keywords. Most of the Web applications use the same or similar layout-generator file to render the resulting pages of the same request-submit functionality. This kind of Web pages is so called template-based Web pages, which are the data sources of our information extraction task.

Using the above-mentioned features of Web applications, we propose a partial information extraction method which can use the information of general Web applications in the same way as Web service APIs, as shown in Fig. 1. Before extracting information from Web applications, emulate a certain functionality of the Web application that the end-user specified. For example, we can emulate the process that input the “Parameter(s)” into the text input field and press the submit button. As a result, we could get a “Target Page”. Then we extract the specified information from this page. Finally, we output the “Extracted Result”.

3.1. Emulation of functionalities in Web applications

Processing a submission-request in the Web application could be thought of as that of an agent begins to process a form by presenting HTML document with the fields in their initial states. It is possible for users to modify the fields, constrained by the field type etc. When a user indicates that the form should be submitted, data set of the form is processed according to its method. Generally, to submit a request, there are “POST” method and “GET” method. As a simplification, “GET” is basically for just getting data like sending query keywords to a search engine, whereas “POST”
may involve anything, like updating a file, or ordering an online product, or sending an E-mail. Naturally, for a “safer” mode, it is possible for some Web sites to use encrypted codes or randomly generated codes during request submitting, which might cause side effects when emulating the functionality. Additionally, JavaScript is used for checking the validity of request in a submission process frequently. Consequently, these diverse processing methods result in difficulties in manually parsing HTML or URL template to get target Web pages.

In order to get the target Web pages from various Web sites, we use HtmlUnit [33] to emulate the submitting operation instead of possibly ineffective mechanisms, such as manual URL-templating. The emulating process is realized by a programmatic description of actual operation based on the type of the functionality.
Three basic types are widely used considering diverse elements and components in a form. One is entering a keyword into the input field by using the keyboard and clicking the submit button by mouse to get the target Web page, which we call “InputBox” type. Another one is selecting an option in a drop-down list in a Web page by mouse to view the target Web page, which we call “OptionList” type. The third one is clicking a link in a link list in a Web page by mouse to go to the target Web page, which we call “LinkList” type. Now our approach can deal with the emulation of these three types of functionalities and other optional elements such as checkboxes and radio buttons are also considering. According to the type of the functionality, the emulating process is shown as follows.

(1) In the case of an InputBox, the input box node is found according to the path string\(^c\) of input area, and its value is set. After that, we emulate the click function of the search button to get the target Web page as described in Fig. 2.

```java
// initial a web client object
WebClient objWebClient = new WebClient();
objWebClient.setRefreshHandler(new ThreadedRefreshHandler());
// enable JavaScript
objWebClient.setJavaScriptEnabled(true);
// enable cookies
objWebClient.setCookiesEnabled(true);
objWebClient.setThrowExceptionOnScriptError(false);
objWebClient.setPrintContentOnFailingStatusCode(false);
// load start page
HtmlPage StartPage = (HtmlPage) objWebClient.getPage(StartPageURL);
// get form
HtmlForm objForm = getSubmitForm(StartPage);
// get text input box
HtmlTextInput objTextInput =
    (HtmlTextInput)objForm.getInputSubmit().elementAt(0);
// get submit button
HtmlInput objSubmitInput =
    (HtmlInput)objForm.getInputSubmit().elementAt(0);
// input the value as keyword
objTextInput.setValueAttribute(Keyword);
// submit request to get the returned target result webpage
HtmlPage ResultPage = (HtmlPage) objSubmitInput.click();
```

Fig. 2. Example codes of emulating inputBox-type-submission.

\(^c\)We use path string to fix the location of nodes in a Web page. The detailed information about the path string could be found in [6].
(2) In the case of an OptionList, text comparison will be done between all selecting options in the specific functionality to find the keyword, then we emulate the setting function of the selecting option to get the target Web page.

(3) In the case of a LinkList, we compare the text of all child links of the specified functionality node to look for the keyword, then we emulate the click function of the link to get the target Web page.

3.2. Information extraction

There are many kinds of information on Web pages such as a piece of text, a map or a group of photos. We classified them into two categories, dynamic contents and static contents. According to their features as shown in Figs. 3 and 4, we use different methods to extract them.

3.2.1. Static contents

For the static Web contents, we defined six kinds of data types, which are single text, continuous text, single object, continuous object, single link and continuous link. Table 1 shows the definitions.

For the contents of text type, the value of the corresponding node is extracted. For the contents of object type, we extract the value of the attribute src of the corresponding node. For the contents of link type, the embedded link value of the corresponding node is extracted. For the contents of continuous type, we extract

Fig. 3. Example of static Web content.
the related value of the corresponding node and its sibling nodes. The extracted information would be organized into XML format.

3.2.2. Dynamic contents

Dynamic Web contents are generated by the client side script codes. Therefore, to extract a dynamic content, we consider how to locate the relative script codes first. The following three kinds of situations frequently occurred.

(1) Script codes or the codes which can illustrate the relationship are nested in the relative HTML tag. Figure 5 shows two examples. In the case shown in A, the content can work correctly if we extract the \texttt{<IFRAME>} node completely. However, in the case shown in B, including the HTML tag (the upper part), the relative script codes (the lower part) are necessary. In this example, there are no other script codes (for example other functions are called in it) need to be extracted. Nevertheless, calling functions in script codes are normally used. In that situation, the automatic detection and extraction are difficult.

<table>
<thead>
<tr>
<th>Table 1. Definitions of six kinds of data types.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single</strong></td>
</tr>
<tr>
<td>Text</td>
</tr>
<tr>
<td>Object</td>
</tr>
<tr>
<td>Link</td>
</tr>
</tbody>
</table>

Fig. 4. Example of dynamic Web content.
A: Script codes are embedded in the HTML tag

```html
1007 <div class="countryTitle" onclick="SwitchMenu(\"sub1\")">\n1008 <img src="/images/night.gif" height=11 width=11>\n1009 (font size=2")Asia\t</a> has 6076 people with a Clock widget</font>\n1010 </div>
```

function SwitchMenu(e){
  if(document.getElementById(e)){
    var el=document.getElementById("masterdiv").getElementsByTagName("span");
    if(el.style.display == "block"){ 
      for(var i=0; i<el.length; i++){
        if((el[i].className) == "countrySubmenu") {
          el[i].style.display = "none";
        }
      }
    }
    else{
      el.style.display = "block";
    }
  }
}

B: Codes could illustrate the relationship are embedded in the HTML tag

Fig. 5. Script codes in HTML tag — situation 1.

(2) Script codes locate at a part which has no hint to illustrate the relationship with the relative HTML tag. Figure 4 shows an example of this situation. For this kind of situation, the automatic detection and extraction are difficult.

(3) Script codes are located at a script file on the server of the Web site. This situation usually accompanied with situation one or two. An example is shown in Fig. 6, in which part of the script codes is nested in the relative HTML tag while the other parts are in a script file on the server. In this example, the location of the script file is given by an absolute path, thus the content can work correctly with the extracted HTML tag. If the location is given by a relative path, it cannot work correctly.

The above analysis shows that it is difficult to detect and extract the location of dynamic contents’ related script codes automatically. As a result, we choose the method that keeps all the script codes in the HTML document instead of extracting them.

We propose two methods for the extraction of dynamic Web contents. The first method is to extract all the script tags and the HTML tags of the specified Web contents. We call it Compression Method. The second method is to extract the whole HTML page and then hide the HTML tags except the specified Web contents.
We call it Visibility Control Method. Although the Compression Method does not work in some cases, the size of the extracted result is much smaller than Visibility Control Method, which can load faster in the mashup application. We explain the details in the following subsections.

3.2.3. Compression method

The Compression Method is implemented by the following steps.

- Extract the whole HTML document of the target Web page.
- Analyze the target Web page obtained and mark the extracted contents and all of its child-nodes.
- Keep the `html`, `script`, `body` tags and the marked tags, and delete the other tags in the HTML document.

Using the above steps, we can get a compressed HTML file from the original source file of the target Web page. In fact many Web sites use a relative path to give the path of the library files. To maintain the functionality of the extracted Web contents, we add a `<base>` tag in the extracted HTML document as a child node of `<head>` tag. The `href` attribute of the `<base>` tag will be set with the value of the domain URL of the target Web page. By adding the `<base>` tag, all files invoked by relative path can be found correctly and the functionality of the Web content can be ensured to work correctly.
However, if the DOM operation or hidden value of the HTML document is used, the dynamic content that is extracted by using this Compression Method cannot work correctly. For example, the script codes in Fig. 5(b) use the following DOM operations. Thus, we propose another method which we call Visibility Control Method.

```javascript
1. document.getElementById(obj);
2. document.getElementById("masterdiv").getElementsByTagName("span");
```

### 3.2.4. Visibility control method

The process of Visibility Control Method is comprised of the following four steps.

- Extract the whole HTML document of the target Web page.
- Hide all HTML nodes in the extracted HTML document of the target Web page by setting the display property of style attribute to “none”.
- Show the Web contents that are specified as extracting contents.
- Modify the corresponding HTML source to avoid the loading of external files, which are not within the target Web contents, such as image and video files. Because these files are not shown in the resulting page and take time to load. For example, we set the “src” attribute of `<img>` tag to be null to avoid loading the image file.

The process that adds the `<base>` tag with the value of the domain URL of the target Web page is also necessary.

There are mainly two differences between the above two methods for extracting dynamic contents.

(1) The applicability of Compression Method has a limitation when DOM operation or hidden values is used in HTML document, while the Visibility Control Method does not have this limitation.

(2) If both methods are applicable, the loading time of the extracted dynamic contents using Compression Method is about half of the Visibility Control Method.

### 4. Design of the Java-Based Library

As shown in Fig. 7, we wrapped several functions that can extract any part(s) from general Web applications according to the given parameters into a Java-based library. Developers can get both static and dynamic information from general Web applications easily by calling different functions with different parameters without building Web services. In addition, the functions that may be used in the integration
are also wrapped in, such as merging XML files according to a given XPath, converting JSON format to XML format and so on. This library consists of two classes of functions. One class includes the functions that are used to do extraction. The other class includes the functions that can be used to realize the various procedures of integration.

4.1. Functions for extraction

Two functions for extraction are supplied in the library to handle different kinds of Web contents. Functions return character strings as the output results.

(1) Extracting Dynamic Content from Web Application

String[] ExtractDynamicContent(keyword, submitPage, inputType, inputArea, extractContents, methodtype, output)

This function is used to extract dynamic Web contents from general Web applications. As is mentioned, we designed two methods (specify by the “methodtype” parameter) to extract dynamic Web contents. The result of this function has two elements. The first element is the extracted contents in HTML format, which could be the same with the original Web page or a part of the original Web page according to the selected method. The second element is the information extracted to transfer to other Web applications as keyword or Web service APIs as parameter. The parameters of the function are listed in Table 2. A coding example of using this function will be shown in Sec. 5.
Extracting Static Content from Web Application

String[] ExtractStaticContent(keyword, submitPage, inputType, inputArea, extrContents, contentsType, output)

This function is used to extract static Web contents from general Web applications. The result of this function has two elements. The first element is the extracted contents in XML format. Developers can convert this XML file into HTML format by using the `transXMLtoHTML` function which will be explained in Sec. 4.2. The second element is the information extracted to transfer to other Web applications as keyword or Web service APIs as parameter.

The definition of parameters `keyword`, `submitPage`, `inputType`, `inputArea`, `extrContents` and `output` are the same as those listed in Table 2. The parameter `contentsType` contains the data type and the XML’s node name of the extracting target Web content, which is recorded as `NodeName:DataType` in an array format. NodeName is any developer-given description to describe the extracting information. DataType is a selection from {single text, single link, single object, continue text, continue link and continue object}.

4.2. Functions for supporting integration

Four functions are designed to support the integration as follows.

(1) Converting XML Format to HTML Format

The extracted result of static Web contents is in XML format. The results received from the servers of Web service APIs also could be in XML format. In order to show the information on Web pages, we need to convert the information into HTML format. The explanation about the Path-Reader application could be found in [6].
format. The function that is used to convert information from XML format to HTML format is defined as following:

\[ \text{String ConvtXMLtoHTML(originalXML, styleFile)} \]

where, originalXML is the original information in XML format, and styleFile is the location and name of the style file. The style file is an XSL file, which is a Style Sheet for XML [39]. The result of this function is in HTML format.

(2) Converting JSON Format to XML Format

The results received from the servers of Web service APIs could be in different formats. The XML format, JSON format and Plain-Text format are widely used. Considering about the easy integration of information from different Web sources, we design a function to convert data in JSON format to XML format. Thus developers can unify the information received from different Web sources into XML format.

After converting the data into XML format, developers can show the JSON format result of a Web service API in a HTML page by using the \texttt{transXMLtoHTML} function with several lines of code. In addition, developers can merge multiple Web services’ results in JSON format easily with the function \texttt{CombineXMLFiles} that we explain later in this section. The function which is used to convert data from JSON format to XML format is defined as follows:

\[ \text{String ConvtJSONtoXML(originalJSON)} \]

where originalJSON is the original data in JSON format. The result of this function is in XML format.

(3) Combining XML Files

To ease the rearrangement of extracted or received information, we design a function to combine the XML files into one. The function that is used to combine XML files according to the given XPath is defined as follows:

\[ \text{String CombineXMLFiles(originalXMLFiles, xPathStr)} \]

where originalXMLFiles is the XML format files recorded in an array, and xPathStr is the XPath used to specify the nodes that the data of these nodes in different XML files is merging targets. The result of this function is in XML format.

(4) Emulating Functionality of Web Application

One of the features of our partial extraction method is emulating a certain functionality of Web application. We supply this feature as a function. It is defined as follows:

\[ \text{String Emulation(submitPage, inputArea, inputType, keyword, returnType)} \]

where the parameters submitPage, inputArea, inputType, keyword and returnType have the same definition as those listed in Table 2. The parameter returnType could be a value selected from \{url and HTMLcode\}. When the returnType is “url”, the result of this function is the URL of the Web page that we got through the emulation. When the returnType is “HTMLcode”, the result of this function is the HTML code of the Web page that we got through the emulation.

At this time, our approach can respond well to the HTTP request types of functionalities in Web applications. However, there are other kinds of request types
in Web applications, such as the requests using Ajax or jQuery technique. We will design a method to track these kinds of requests in future.

5. Implementation and Evaluation

Based on this Java-based library, developers can build mashup applications more easily with the information from Web applications and Web service APIs by programming. We build an example mashup application by using this library, which is shown in Fig. 8. This mashup application is designed to show the travel information about a country and the popular destinations in the country. It has the following functions.

(1) By selecting a country from the country list, the information in four aspects would be displayed. In the order of left-right and up-down, they are:

(a) the map of the country with a marker on the capital;

![Fig. 8. Mashup application using the library by end-user programming.](image-url)
(b) the travel guide, simple introduction and some facts of the country. The travel guide is extracted from World Travel Guide [38] and the simple introduction and some facts are extracted from BBC Country Profiles;
(c) the local clock of the capital extracted from Localtimes.info, which updates time every second;
(d) the popular destinations extracted from World Travel Guide and Yahoo Travel [41].

(2) By clicking the marker on the capital, the photos of the country’ tourism spots will be shown. The photos are extracted from World Travel Guide and displayed in circular mode automatically.

(3) By clicking any city name listed in the Popular Destinations list, the information about the city will be displayed in a new page, as shown in Fig. 9. In the order of left-right and up-down, they are:
(a) the map of the country with a marker on the selected city;
(b) the weather information of the selected city which is gotten from a REST service API;
(c) the latest travel news extracted from Travel News [37].
(d) the popular destinations extracted from World Travel Guide and Yahoo Travel, which is the same part as shown in Fig. 8.

(4) By clicking the marker on the city, the photos about the city’s tourism spots will be shown. The photos are extracted from World Travel Guide and displayed in circular mode automatically.

The Web sources that we used in this mashup application are shown in Table 3. The functions that we used in the Java-based library are listed in Table 4. We give the code sizes of the actual program in Table 5 as a kind of user study. At the same time, we implement the same mashup application by programming from scratch. The code size for extracting dynamic contents is about 1300 lines. The code size for extracting static contents is about 1200 lines. The code size for emulating the submission function of a Web application is about 260 lines (this is an average value of different kinds of input types) which prove that our approach could realize the integration of Web applications and Web services without writing so many codes by using the functions defined in the Java-based library. The following codes show how to call the `ExtractDynamicContent()` by using JavaScript. The variable “result” is used to get the extracted result. It could be a string in XML or HTML format according to the type of extracting method.

```javascript
function searchTravelImg(countryNM){
    var strInput = countryNM;
    var strStartPageURL = "http://www.worldtravelguide.net/destinations/";
    var strInputType = "LinkList";
    var arrExtractContent = new Array(1);
    arrExtractContent[0] = "/HTML/BODY/DIV(page,null,null)/DIV(page-inner,null,null)";
    var strInputArea = "";
    var strMethodType = "Visibility\Control\Method";
    var strOutput = "null";
    var result = new Array(2);
    //get images from world travel guide
    result = ExtractDynamicContent(strInput, strStartPageURL, strInputType, strInputArea, arrExtractContent, strMethodType, strOutput);
    return(result[0]);
}
```

*The code size is in the case that there is one extracting item. If the number of extracting items increase, the code size will increase because of the parameter numbers.*
We got the help of six people to build mashup applications in which the Java-based library we designed was used. Four of the participants in this experiment are the first-year students of graduate schools, and two of them majored in Computer Science. The other two participants are undergraduate students in the fourth year. They all took at least two courses that involve programming, but do not think of themselves as expert programmers. None of them had the experience of building mashup applications. Five of the participants used certain Web service API once, and the other one has no experience of using Web service API. However, they all have basic knowledge about HTML language.

They built six mashup applications and reused the information from 20 applications. The basic information about these applications is listed in Table 6. In addition, some participants did tests with the extracting functions which are not used in their mashup applications. According to the questionnaires, a problem occurred once in using Visibility Control Method to do dynamic extraction. There is an unexpected blank space in the extracted content. This is because the parent nodes of the selected contents must be shown in this method and some parent nodes on Web page have background attribute. Now this method is designed to keep the same display mode as the original Web page. We would like to do more user-study on this fact to decide whether or not we should introduce change to this method. The participants’ comments show that functions in the Java-based library we proposed could meet most of their requests in reusing the information of general Web applications in developing mashup applications.

<table>
<thead>
<tr>
<th>Web source (type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Maps (SOAP API)</td>
<td>Diagrammatic representation of an area.</td>
</tr>
<tr>
<td>GoogleGEO (REST API)</td>
<td>The response result of this API is in JSON format. We convert the result</td>
</tr>
<tr>
<td></td>
<td>into XML format by using the function in the Java-based library. Then</td>
</tr>
<tr>
<td></td>
<td>we extract the information of longitude and latitude and use this information</td>
</tr>
<tr>
<td></td>
<td>to locate the marker on the Google Maps.</td>
</tr>
<tr>
<td>World Travel Guide (Web Application)</td>
<td>Various kinds of travel information. Here, we extract the images, popular</td>
</tr>
<tr>
<td></td>
<td>destinations, travel guide of the given country and the images of the given</td>
</tr>
<tr>
<td>BBC Country Profiles (Web Application)</td>
<td>Information of countries. Here, we extract the simple introduction and</td>
</tr>
<tr>
<td></td>
<td>facts of the given country. We extract the name of the capital at the same</td>
</tr>
<tr>
<td></td>
<td>time as the “OP” item which is used as the parameter of the GoogleGEO API</td>
</tr>
<tr>
<td></td>
<td>to get the coordinate information later.</td>
</tr>
<tr>
<td>Localtimes.info (Web Application)</td>
<td>Local time clock of cities. Here, we extract the capital’s clock of the given</td>
</tr>
<tr>
<td>Yahoo Travel (Web Application)</td>
<td>Various kinds of travel information. Here, we extract the city names from</td>
</tr>
<tr>
<td>Google Weather (REST API)</td>
<td>The response result of this API is in XML format. We convert the result</td>
</tr>
<tr>
<td></td>
<td>into HTML format by using the function in the Java-based library.</td>
</tr>
<tr>
<td>Travel News (Web Application)</td>
<td>Latest travel news about the given keyword. Here, we extract the top five</td>
</tr>
<tr>
<td></td>
<td>news.</td>
</tr>
</tbody>
</table>
There are two parts when reuse the information of general Web applications. They are reusing the functionalities and reusing the extracted information of Web applications. When reusing the functionalities of general Web applications, if the submission type of the functionality is Request-Submit, our approach could meet the

Table 4. Functions used in travel information mashup application.

<table>
<thead>
<tr>
<th>Function</th>
<th>Process description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExtractDynamicContent()</td>
<td>Extract the images of the given country or city, and keep the functionality that displays in circular mode automatically.</td>
</tr>
<tr>
<td>ExtractStaticContent()</td>
<td>Extract the travel guide text information, simple introduction and facts of the given country.</td>
</tr>
<tr>
<td>CombineXMLFiles()</td>
<td>Combine the travel guide text information extracted from World Travel Guide with the text information extracted from BBC Country Profiles.</td>
</tr>
<tr>
<td>ConvtJSONtoXML()</td>
<td>Convert the response result of GoogleGEO into XML to get the coordinate information easily.</td>
</tr>
<tr>
<td>ConvtXMLtoHTML()</td>
<td>Convert the combined text information into HTML format.</td>
</tr>
</tbody>
</table>

Table 5. Code size of functionalities in travel information mashup application.

<table>
<thead>
<tr>
<th>Object type</th>
<th>Process</th>
<th>Code size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extracting Functions</td>
<td>Extract from Application</td>
<td>12</td>
</tr>
<tr>
<td>Web Services</td>
<td>Load Map</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Show Weather Information</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>Get Coordination Information</td>
<td>15</td>
</tr>
<tr>
<td>Supporting Functions</td>
<td>Combine XML Files</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Convert JSON to XML</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Convert XML to HTML</td>
<td>6</td>
</tr>
<tr>
<td>Others</td>
<td>Create Marker</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Mark City</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Country Selection</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>City Selection</td>
<td>23</td>
</tr>
</tbody>
</table>

There are two parts when reuse the information of general Web applications. They are reusing the functionalities and reusing the extracted information of Web applications. When reusing the functionalities of general Web applications, if the submission type of the functionality is Request-Submit, our approach could meet the

Table 6. Basic information of end-users’ mashup applications.

<table>
<thead>
<tr>
<th>Functionality of application</th>
<th>Web application in use (http:///)</th>
<th>Type of contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get information about the given keyword from two news Web sites, a photo Web site and wikipedia.</td>
<td>(1) news.baidu.com (2) news.sina.com.cn (3) <a href="http://www.topit.me">www.topit.me</a> (4) en.wikipedia.org/wiki/Main_Page</td>
<td>text, image, flash</td>
</tr>
<tr>
<td>Get the first searching result from three different Search Engines about the given keyword.</td>
<td>(1) <a href="http://www.baidu.com">www.baidu.com</a> (2) <a href="http://www.soso.com">www.soso.com</a> (3) cn.yahoo.com</td>
<td>text</td>
</tr>
<tr>
<td>Get information about a given country from several Web sites.</td>
<td>(1) localtimes.info (2) <a href="http://www.countryreports.org">www.countryreports.org</a> (3) travel.state.gov/travel/cis_pa_tw/cis/cis_4965.html</td>
<td>text, image</td>
</tr>
<tr>
<td>Get stock information from two Web sites about the given keyword (stock name or code).</td>
<td>(1) finance.sina.com.cn/stock/ (2) stock.cn.yahoo.com/</td>
<td>text, flash</td>
</tr>
</tbody>
</table>
request, except the submission which responds to the clicking action on Flash. According to the questionnaires, none of the participants got any problem at this point. For reusing the extracted information of Web applications, our approach could wrap both static and dynamic contents except that a relative path is used as a parameter in the script codes when extracting dynamic contents. According to the questionnaires, the total extracting actions of all participants are 40 times, besides a participant got wrong contents once. One pop-up advertisement which he did not select showed in the extracting result. This means we still need to improve the extracting algorithm.

Another information we got from the questionnaires is that if the extracted Web contents contains a large number of HTML codes, the extracting procedure may be slow down. There are three cases that the extracting time is longer than 30 seconds. The response time of the Web application’s server is one of the reasons. Besides that, we need to improve the extracting speed in the future.

6. Conclusion and Future Work

In this paper, we proposed a programmer-oriented (including end-user programmers and professional programmers) approach for building mashup applications easily, which is based on the partial information extraction method we proposed. This approach could help the mashup developers focus on the scenario design instead of how to get information from the Web sources. By using the Java-based library we supplied, the mashup developers can realize the personal Web service functions and implement the mashup applications with any part(s) from general Web applications easily. The experiments show that even end-user programmers can master the usage of our approach in short time, and our approach can meet their needs in building mashup applications.

As the future work, we would like to ease the integration process when using Web applications and Web service APIs, which could be implemented with little or no

<table>
<thead>
<tr>
<th>Functionality of application</th>
<th>Web application in use (http://)</th>
<th>Type of contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the top movie list from five different Web sites, including the released movies and the coming soon movies.</td>
<td>(1) theater.mtime.com/China_Beijing/ (2) ent.sina.com.cn/movie/top10bang.shtml (3) <a href="http://www.juchang.com/movie/">www.juchang.com/movie/</a> (4) <a href="http://www.moviefone.com/new-movie-releases?sort=release-date">www.moviefone.com/new-movie-releases?sort=release-date</a> (5) movies.yahoo.com/coming-soon/</td>
<td>text, image</td>
</tr>
<tr>
<td>Get the titles of the ten songs which are listed in top popular list in certain Web site. Then get the lyrics of these songs according to the titles in another Web site.</td>
<td>(1) list.mp3.baidu.com/top/top500.html (2) <a href="http://www.5ilrc.com/">www.5ilrc.com/</a></td>
<td>text</td>
</tr>
</tbody>
</table>

Table 6. (Continued)
programming. We will give the GUI for easier configuration and implementation of the integration using Web applications and Web service APIs without programming.

Appendix A. Index of Abbreviations

API: Application Programming Interface
DOM: Document Object Mode
GUI: Graphical User Interface
JSON: JavaScript Object Notation
REST: Representational State Transfer
RSS: Really Simple Syndication
SOAP: Simple Object Access Protocol
UDDI: Universal Description, Discovery and Integration
WSDL: Web Services Description Language
XSL: EXtensible Stylesheet Language

References


