Analysis and Evaluation of UI Component Integration for Large-Scale Web Content Reuse

Hao Han* and Yinxing Xue†
*Department of Information Science, Faculty of Science
Kanagawa University, Japan
han@kanagawa-u.ac.jp
†Temasek Laboratories
National University of Singapore, Singapore
tslxuey@nus.edu.sg

***Abstract*—Mashup has promoted new creativity and functionality to Web applications through the integration of information/knowledge from multiple websites. However, there is no uniform interface to access the data, computations and user interface provided by the different kinds of Web content. Without open APIs, it is difficult to integrate Web applications with other Web content. In this paper, we present a framework for flexible and lightweight integration of Web content for personal use. We propose a simple Extensible Markup Language (XML) based Web content description language to define the Web content and configure the mashup applications. We also conduct an exploratory analysis and evaluation of user interface components for large-scale reuse.**

Keywords*–*UI Component; API; Integration; Retrieval; Security.

I. Introduction

The availability of information/knowledge is increasing explosively on the Web with the development of the Internet, however useful information is not always in a form that supports end-user requirements. This is because there are no uniform interfaces to access the data/computations (application logic) or user interfaces provided by the different kinds of Web content. For reuse and integration, the content used in the majority of the current mashup applications is typically obtained from third party sources through public Web service Application Programming Interfaces (API). The integration of general Web applications requires additional efforts including programming and configuration compared to single-type Web service integration. Such integration is beyond the skills of typical users and restricted to specific technologies or domains. In this paper, we present a description-based framework of flexible and lightweight integration of Web content for customized reuse. We define Web Content Description Language (WCDL) to configure the mashup Web applications. Furthermore, we conduct an analysis and evaluation of User Interface (UI) components for further large-scale reuse.

The remainder of the paper is organized as follows. In Section II, we present the motivation for our study and an overview of the related work. In Section III, we explain description-based integration. We analyse and evaluate the proposed approach for large-scale reuse in Section IV. Finally, we conclude our approach and discuss future work in Section V.

II. Motivation and Related Work

The Majority of Web mashup technologies are based on a combination of Web services and Web feeds. Yahoo! Pipes [1] is a composition tool to aggregate, manipulate, and mashup Web services or Web feeds from different websites with a graphical user interface. Mixup [2] is a development and runtime environment for UI integration [3]. It can quickly build complex user interfaces for easy integration using available Web service APIs. Mashup Feeds [4] and WMSL [5] support integrated Web services as continuous queries. They create new services by connecting Web services using join, select, and mapping operations. Similar to these methods, other service-based methods [6], [7], [8] are limited to the combination of existing Web services, Web feeds, or generated Web components.

For the integration of parts of Web applications without Web service APIs, partial webpage clipping methods such as C3W [9] and Marmite [10] are widely used. Users clip a selected part of a webpage, and paste it into a customized webpage. Marmite [10], implemented as a Firefox plug-in using JavaScript and XUL (XML User Interface Language [11]), employs a basic screen-scraping operator to extract the content from webpages and integrate it with other data sources. The operator uses a simple XPath pattern matcher and the data is processed in a manner similar to UNIX pipes. Intel MashMaker [12] is a tool for editing, querying, manipulating and visualizing continuously updated semi-structured data. It allows users to create their own mashups based on data and queries produced by other users and remote sites. However, these methods can only extract Web content from static HyperText Markup Language (HTML) pages. Moreover, it is not easy to realize the interaction between different Web applications such as Safari Web Clip Widgets [13], unless there is a special Web browser and plug-in.

The majority of the current methods are based on existing Web service APIs or Web feeds, require a professional Web programming ability [14], [15], or have other limitations (e.g., the components must be produced by portlets [16] or WSRP [17]). To address these issues, we propose a flexible and lightweight description-based mashup of Web content. Our integration framework facilitates the integration of various Web applications and services for typical users with minimal programming experience. The range of content is extended from Web services to the “traditional Web” (data, computations, and the UI components of general Web applications) including content dynamically generated by client-side scripts or plug-ins.

III. Description-Based Integration

The propose of framework is based on the description and integration of Web content, which includes Web services, Web feeds and Web applications. As illustrated in Figure 1, we begin by describing the target Web content in a WCDL file. Then, client requests are sent to the target websites. According to the defined WCDL file, partial information is extracted from the responding webpages if the target Web content derives from Web applications. The XML-based content (returned from the Web services or extracted from the responding webpages) is transformed into an HTML format and the target Web content is integrated.

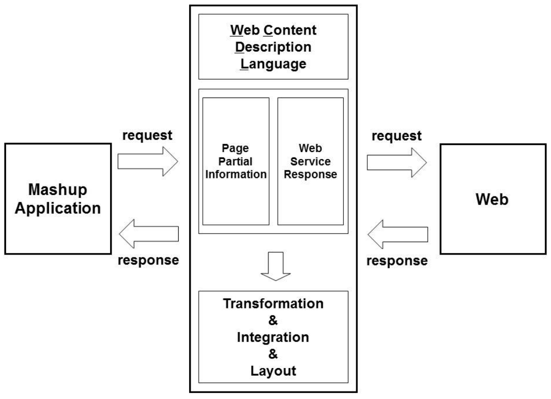

Figure 1. Outline of the proposed description and integration approach/system

“WCDL”is XML-based and intelligible to typical users. In this paper, a typical user is specified as a user who has basic computer knowledge and operation skills without professional programming ability/experience. Compared with standard Web services, it is not easy to access and integrate Web applications because web applications are designed for browsing, not for parsing by program. Without interfaces such as Simple Object Access Protocol (SOAP) [18] or Representational State Transfer (REST) [19], an extraction is used to acquire the target content, and emulation is utilized to realize the automated process of sending requests and receiving responses.

As indicated in Figure 2, WCDL contains the following items to describe the necessary Web application information reflecting end-user operations (e.g., users find the text input field, input the keywords, submit the request, and search for the target content in the response page). *StartPage* is the Uniform Resource Locator (URL) of the webpage of the target Web application where the request of the end-user is submitted. *InputType* is the type of request-input element such as InputBox (text input field), OptionList (drop-down option list), or LinkList (anchor list) in the StartPage. *InputArea* is used to locate the request-input element in the StartPage. If there are other elements with the same InputType in the StartPage, we must define the InputArea.

ContentType is the type of target content: static or dynamic. Static content is the unchangeable portion displayed on a webpage after the page is fully loaded and during the viewing process. It contains two kinds of information: property and structure. The property is a text (character string), image (instance of graph), link (hyperlink), or object (instance of video or other multimedia file). The structure is a single, list, or table. Dynamic content is the portion dynamically generated or changed by client-side scripts or plug-ins in a dynamic HTML page according to the users’ operations. *ContentArea* is used to locate and extract the target content in the response webpage using an XPath expression. *ContentStyle* is the layout of the target content in the integrated resulting webpage. It is typically limited to static Web content. The extraction results are in an XML format and the style refers to the XML Stylesheet Language Transformation (XSLT) files defined by the end-user. If the webpage containing the target content is a fixed page (e.g., a static webpage with an unchangeable URL), the values of *InputType* and *InputArea* are null (*StartPage* is the target page).

```
<target type="application" ID="...">
    <StartPage>
      URL of webpage where request is submitted
    </StartPage>
    <InputType>
      Type of request-input element in StartPage
    </InputType>
    <InputArea>
      Path of request-input element in HTML document
      of StartPage
    </InputArea>
    <ContentType>
        <type>
          Type of a target part
        </type>
    </ContentType>
    <ContentArea>
        <content>
          Path of a target part in HTML document of
          response webpage
        </content>
    </ContentArea>
    <ContentStyle>
        <style>
          Layout of a target part in resulting webpage
        </style>
    </ContentStyle>
</target>
```

Figure 2. Description of a Web application

Web services and feeds are self-contained and self-describing, and communicate using open protocols such as HTTP. The most widely used style architectures of Web services are SOAP and REST. Recently, new Web services are implemented using the REST architecture rather than SOAP because REST is lightweight, easy to build, and provides readable results. Furthermore, Web feeds such as Atoms or RDF Site Summary (RSS), are usually considered a kind of simplified REST service.

As illustrated in Figure 3, we use the following items to describe REST Web services in WCDL. *BaseURL* is a reference to a method of the target Web service. It contains the hostname, service name, version number, and method name. *Key* is necessary in some Web services. *Query* is the query string of the request URL. The value of each <parameter> is the name of a query parameter. The actual query parameters follow the method (start with a question mark) and assume the form *parameter=value*, where the query parameters and values are URL encoded. Multiple query parameters are separated by an ampersand. The *Type* of a method specifies the format to send requests to the target Web service. The majority of REST API requests use the “GET” method. For the “POST” method, there is no question mark following the method and the query parameters are passed in the “POST” data block.

ContentStyle is the layout of the response content in the integrated resulting webpage. It refers to a layout (template) file. For websites that provide both REST and SOAP Web services, we select the REST Web services as our default selection. For SOAP Web services, we transform these into REST queries (see Yahoo Query Language [20]). The transformation is a semi-automatic process and requires some manual configuration based on the description of the SOAP services in the WSDL [21] files. Once the specification of the target server endpoint URL, namespace declarations, and body elements (Uniform Resource Identifier (URI) of the target object, method name, parameter names) of the SOAP envelope are retrieved, a REST query is generated automatically in the proposed system.

```
<target type="service" ID="...">
    <BaseURL>
      Reference to a service and method
    </BaseURL>
    <Key>
      API key of Web service
    </Key>
    <Query>
        <parameter>
          Name of query parameter
        </parameter>
        ...
    </Query>
    <Type>
      GET or POST method
    </Type>
    <ContentStyle>
      Layout of response content in resulting webpage
    </ContentStyle>
</target>
```

Figure 3. Description of a REST Web service

Based on the abovementioned description in the WCDL file, we search for the target Web content from the Web application. There are two steps during this process. First, we consider the response webpages as the target webpages. Then, we search for the target parts in the response webpages.

Web applications normally provide the request-submit functions for the end-users. For example, search engine applications provide the text input field in the webpage for keyword inputting. Users input the query keywords, submit the request to the server side, and receive the response webpages. To submit a request, there are generally different methods such as “POST” and “GET”. For a “more secure” mode, some websites utilize encrypted codes or randomly generated codes during request submitting, which are side-effects of information security. JavaScript can be run as part of submission for validating a request. Hidden objects, which represent hidden input fields in an HTML form, are also widely used to record different submitting options. Consequently, these diverse processing methods can make it difficult to manually parse the HTML or URL template to acquire the target webpages. To obtain the response webpages from all types of Web applications automatically, we implement HtmlUnit [22] to emulate the submitting operation, instead of using the URL templating mechanism. The emulation is based on the event trigger of the element of *InputType* within the *InputArea* of *StartPage*.

ContentArea is used to determine the target parts from the webpage. In the tree structure of an HTML document, each path represents the root node of a subtree and each subtree represents a part of the webpage. Response webpages generally have the same or similar layouts if the requests are sent to the same request-submit function. During node searching, if a node cannot be found using a path, a similar path can be tentatively used to search for the node [23].

Once the target parts are located, the content is extracted from the nodes in text format, except for the tags of the HTML document, based on the corresponding *ContentType*. The extracted static content is in an XML format and can be transformed into an HTML document by *ContentStyle*. For dynamic content, we use an effective Hide-and-Display method to control visibility, as the static content extraction method cannot retain the functionality of client side scripts. Currently, many websites use Dynamic HTML (DHTML) or Document Object Model (DOM) [24] scripting technology to create interactive webpages. These contain HTML, client-side scripts, and Document Object Model. The scripts change variables (including the elements outside the target parts such as the hidden values) programmatically in an HTML document and affect the look and function of the static content. If we remove the other parts from the target parts using a traditional static extraction method, the original execution environment of scripts would be broken and the scripts could execute abnormally. Thus, we retain all the parts of each webpage and change the visibility according to the following steps to display the wanted target parts only and hide the other parts of the webpage by setting the property “display” of attribute “style” of all the nodes to “none” (*style.display=“none”*).

Except for the dynamic content generated by client-side scripts, both the Web service response and the extracted partial information from Web application are in an XML format. We use an XSLT template processor (*ContentStyle* in WCDL) to transform the XML data into HTML or XHTML documents. After the description, extraction, and transformation, we integrate the content (in an HTML format) from different Web applications or services into a resulting page. We use iframe (or div, span) tags as the default Web content container. This is an HTML element that makes it possible to embed an HTML document into another HTML document. Whereas a regular frame is typically used to logically subdivide the Web content of one webpage, an iframe is more commonly used to insert Web content from another website into the current webpage. Moreover, iframe is supported by all mainstream browsers.

We illustrate an example of the generation of integrable content using a BBC news search. As illustrated in Figure 4, after the user inputs the search keywords and sends the request, the proposed system sends the request to each target website and receives the response content, which is presented as an integrable content segment.

We also generated an example of a mashup Web application. It integrates parts from the following seven Web applications/services/feeds and implements a country-information search function. Table I provides a description of each target Web content. Target A (dynamic contents, mouse move event trigger, client JavaScript): real-time local time; Target B (mashup application): weather information from a mashup application integrated by a weather service and Google Maps service. The weather information part is created by client-side scripts that can respond to click and span events; Target C (static text and image): country’s location, basic information and leader’s photo; Target D (search engine application): latest corresponding news articles; Target E (flash interaction with

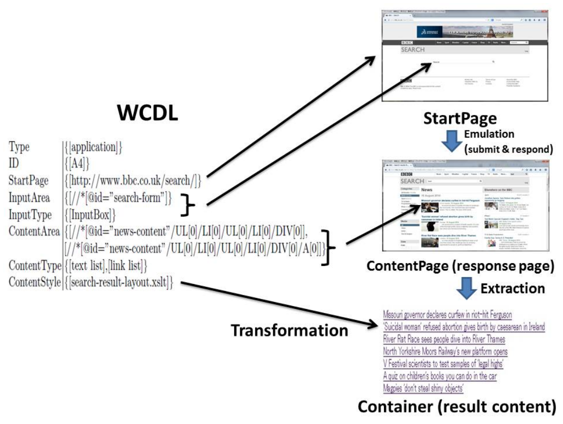
WCDL
Type {[application]}
ID {[A4]}
StartPage {[http://www.bbc.co.uk/search/]}
InputArea {[//*[@id="search-form"]}
InputType {[InputBox]}
ContentArea {[//*[@id="news-content"/UL[0]/LI[0]/UL[0]/LI[0]/DIV[0]],
[//*[@id="news-content"/UL[0]/LI[0]/UL[0]/LI[0]/DIV[0]/A[0]]}
ContentType {[text list],[link list]}
ContentStyle {[search-result-layout.xslt]}
StartPage
Emulation (submit & respond)
ContentPage (response page)
Extraction
Transformation
Missouri governor declares curfew in riot-hit Ferguson
'Suicidal woman' refused abortion gives birth by caesarean in Ireland
River Rat Race sees people dive into River Thames
North Yorkshire Moors Railway's new platform opens
V Festival scientists to test samples of 'legal highs'
A quiz on children's books you can do in the car
Magpies 'don't steal shiny objects'
Container (result content)

Figure 4. WCDL describes the flow of generation of integrable content

JavaScript): photos displayed with a map that can respond to click events and present the relevant pictures; Target F (Web service): simple introduction to the country; and Target G (RSS/Atom feed): videos about the country. As illustrated in Figure 5, after the user inputs the country name and sends the request, the mashup Web application sends a request to each target website and receives the response Web contents, which are presented in a resulting integrated webpage after layout rearrangement.

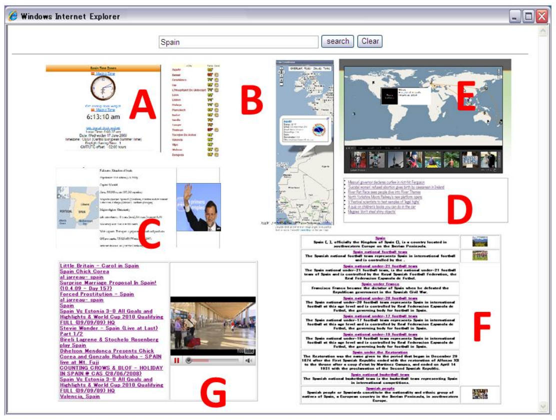

Figure 5. Resulting webpage of an example mashup application

Compared to the single mixing of static webpage segments, in the proposed system, the integrated segments retain their original active functionalities and can be customized into a uniform layout style in a resulting webpage.

IV. Analysis and Evaluation

We developed a lightweight integration of Web content with simple description and easy configuration based on WCDL. WSDL and WADL [25] are used to describe a series of Web services with details such as abstract types, abstract interfaces, and concrete binding. It cannot be expected that all the tasks will be completed automatically using WSDL or WADL files. For example, the main resource of a WADL description for the Yahoo! News Search application [26] contains more than 40 parameter/attribute values. Compared with the

TABLE I. DESCRIPTION OF TARGET WEB CONTENT

Target	Item	Value
A	Type	{[application]}
	ID	{[A1]}
	StartPage	{[http://localtimes.info/]}
	InputArea	{[null]}
	InputType	{[LinkList]}
	ContentType	{[dynamic]}
	ContentArea	{[/HTML/BODY/DIV[0]/DIV[2]/DIV[1]/DIV[0]]}
	ContentStyle	{[null]}
B	Type	{[application]}
	ID	{[A2]}
	StartPage	{[http://www.weatherbonk.com]}
	InputArea	{[//*[@id="searchForm"]]}
	InputType	{[InputBox]}
	ContentArea	{[//*[@id="grid"],[//*[@id="bonkMapColumn"]}
	ContentType	{[dynamic],[dynamic]}
	ContentStyle	{[null]}
C	Type	{[application]}
	ID	{[A3]}
	StartPage	{[http://news.bbc.co.uk/1/hi/country_profiles/default.stm]}
	InputArea	{[null]}
	InputType	{[OptionList]}
	ContentArea	{[/HTML/BODY/DIV[0]/DIV[5]/TABLE[0]/TR[0]/TD[1]/ TABLE[2]/TR[1]/TD[0]/TABLE[1]/TR[0]/TD[0]/DIV[0]/ IMG[0]], [//*[@id="content"/UL[0]], [/HTML/BODY/DIV[0]/DIV[5]/TABLE[0]/TR[0]/TD[1]/ TABLE[2]/TR[1]/TD[0]/P[28]/TABLE[0]/TR[0]/TD[0]/ DIV[0]/IMG[0]]}
	ContentType	{[image],[text list],[image]}
	ContentStyle	{[automatch.xslt]}
D	Type	{[application]}
	ID	{[A4]}
	StartPage	{[http://www.bbc.co.uk/search/]}
	InputArea	{[//*[@id="search-form"]}
	InputType	{[InputBox]}
	ContentArea	{[//*[@id="news-content"/UL[0]/LI[2]/UL[0]/LI[0]/ DIV[0]], [//*[@id="news-content"/UL[0]/LI[2]/UL[0]/LI[0]/ DIV[0]/ A[0]]}
	ContentType	{[text list],[link list]}
	ContentStyle	{[search-result-layout.xslt]}
E	Type	{[application]}
	ID	{[A5]}
	StartPage	{[http://www.trippermap.com]}
	InputArea	{[null]}
	InputType	{[null]}
	ContentArea	{[//*[@id="maptabs"]}
	ContentType	{[dynamic]}
	ContentStyle	{[null]}
F	Type	{[service]}
	ID	{[W6]}
	BaseURL	{[http://en.wikipedia.org/w/api.php]}
	Key	{[null]}
	Query	{[action=opensearch],[format=xml],[search]}
	Type	{[GET]}
	ContentStyle	{[wiki-layout.xslt]}
G	Type	{[service]}
	ID	{[W7]}
	BaseURL	{[http://gdata.youtube.com/feeds/base/videos]}
	Key	{[null]}
	Query	{[q]}
	Type	{[GET]}
	ContentStyle	{[youtube-layout.xslt]}

programmer-oriented WSDL or WADL, our WCDL provides a shorter and simpler description format, and is applicable to the description of general Web applications. It is easier to read, write, reuse, and update any part of a mashup applications than to work with end-user programming methods. The proposed approach allows a users with no or minimal programming experience to integrate the Web content from multiple Web applications and services. Because the proposed approach is generally applicable to Web applications and services, any content from any kind of Web application becomes integrable and reusable.

Currently, the proposed extraction method is based on the fact that response webpages returned from the same Web application use the same or a similar layout. If the response webpages use different layouts, the extraction precision would degrade because the paths of the target parts vary with the layouts of the webpages (XPaths are changed). Moreover, if the layout of the webpage is updated, users must change the value of *ContentArea* in the corresponding WCDL file (they can use tools such as Firebug [**?**] and are not required to read the source code of the HTML documents). In the proposed integration system, we use emulation to submit requests and acquire responses automatically without manual analysis of typical URLs for templates. Nevertheless, the emulation process of HtmlUnit is slow for some websites and requires more time than the URL templating mechanism if the websites rely on external scripts in the submitting process. The proposed approach is not yet fully automated and may not be efficient in emulation mode.

For the large-scale reuse of UI components, a challenge is to include efficient retrieval of the UI components to facilitate client-side integration. Users currently must locate functionality from websites using general search engines such as Google and Bing, or a library of predefined UI components that is also manually created and limited to a small scale. This conventional method of Web functionality retrieval is inefficient because current general Web search engines primarily use a content-oriented (non-functionality-oriented) search mechanism and information about functionality is beyond the searching scope of these current mainstream search engines. Moreover, this method leads to time-consuming manual verification and comparison if users wish to locate the most desirable and suitable component. In our past experiments and implementations, we were required to spend more than one hour to find and select each suitable component on average, although generating the WCDL for a segment usually required less than 5 minutes (15 senior students of CS were requested to independently complete the construction of mashup application integrating diverse content. The longest time consumed for the WCDL of a segment was 25 minutes and the average time was approximately 4 minutes).

To the best of our knowledge, there is no previous work on large-scale UI component retrieval. We designed a novel searching mechanism [27] that is implemented in the UI component retrieval system. A crawler is used to collect webpages. For webpages containing components, background information such as URL, title, and meta data is collected. For each submitting form and component, diverse attributes (e.g., ID, name, and XPath) are extracted as necessary information and other related information (e.g., alt, hidden values, and defined event trigger JavaScript functions) are extracted as optional information. This information is analyzed and classified to generate a recognition pattern for each Web UI and the index information is then created for component retrieval.

Table II displays a list of component properties as indexes of the BBC news search. *Category*: group of websites/webpages with similar subjects, which are extracted from the meta data of webpage; *Function type*: functionality of UI component; *UI*: UI component type (e.g., text input field, drop-down option list, anchor list); *Request candidate number*: input candidates are unlimited for a text input field or the number of options in an option/link list.“unlimited” is the best applicability of the interface and a larger number represents a better applicability. *Data volume*: size of data transferred between client and component/target website; *Speed*: average time (ms) for the request-response process; *Stability*: statistics of access success (whether the UI component normally runs) rate compiled periodically (100% implies it normally runs during webpage crawling); *Language*: primary language of the website; and *Script*: whether client-side JavaScript runs in the request-response process.

Each index has its unique characteristics. For a more objective evaluation of components, a supervised machine learning approach is employed to perform a construction of ranking models for the further recommendation mechanism in the proposed approach. Features are assigned to represent numerical weightings of indexes such as speed and stability, and manual assessment results are used as training/testing data. Clustering is used to partition components into groups (clusters). The components in the same group are more similar to each other than to those in other groups in *Category*. Users search for the components by providing the component type and other search keywords. Compared with the general content-oriented search engines, this component-oriented search mechanism can achieve faster and more effective component retrieval (within minutes).

TABLE II. PROPERTIES OF UI COMPONENT

Category	Application, BBC, Search, news
Function type	search
UI	text input field
Request candidate number	unlimited
Data volume	82.94 KB
Speed	2,759 ms
Stability	100%
Language	English
Script	no

However, actually, compared to non-context Web APIs, context-related UI components are coarse-grained. The property information extracted from meta data and tag names is not enough for precise analysis. Moreover, it is also difficult to measure the relative importance of a UI component like PageRank since the main functionality of a UI component is limited in the inner side of a website.

For large-scale reuse of the existing content from Web services or even general Web applications, we must evaluate the scalability of the proposed approach as well as the reusability of the extracted content. On average, it requires approximately 100-1000 ms to extract the contents from a given webpage; there is not a significant difference between a static webpage and a dynamically generated webpage. In the proposed framework, the extraction and the integration are implemented in different components, which allows both steps to be executed in parallel. The extraction step can prebuild a database of possibly useful content that is extracted during crawling. The proposed approach is efficient in both steps and feasible for large-scale reuse, e.g., searching and integrating content from hundreds of thousands of webpages.

Regarding the reusability of the extracted content, the proposed approach realizes the separation of the data and the presentation. Using an XSLT template processor, we can transform the XML-format content into XHTML according to the desired layout in the XSLT file. In addition to the data, the extracted content may include UI components. The reusability of these UI components is not undermined as the proposed approach adopts an effective Hide-and-Display method: retaining the contextual client-side script that these UI components require for a normal execution, and only setting the irrelevant content from this client-side script as hidden.

Another important consideration in large-scale reuse is the security issue. As the proposed approach extracts content from general Web applications, the content from malicious webpages could be included. However, as the proposed approach does not directly reuse the runnable JavaScript code, the content-based reuse is not as vulnerable as script-based reuse. For UI components extracted from potentially malicious webpages, the proposed approach actually provides a protection mechanism by dividing the searched content or UI components into isolated subpages [28] called iframes. For different iframes, different permissions are granted to allow them to execute only legitimate operations. Even if an iframe contains malicious content, it will not contaminate the others. The modularity of iframes in our resulting webpage mitigates the security risks.

The content searched by the proposed approach is generally taken from public sources and includes many different types, e.g., encyclopedias, articles of news, blogs and posters from discussion forum, information of items or products, government records and financial data. Among these different types, some of them are in the well-defined format (financial data like currency exchange ratio provided by an open service), while some others are not (static text and image searched from public profiles provided by a Social Networking Service (SNS)). No matter the content is in the well-defined format or not, the proposed approach mainly aims to describe and integrate the content. We are not searching for private data, usually protected by encryption or encoding, which is ignored by the system.

V. Conclusion and Future Work

In this paper, we presented an effective approach to integrate content from Web applications and services/feeds for personal use. The proposed approach uses WCDL to describe the Web content and functionalities, and produces a lightweight integration using a Web content extraction and hide-and-display method. Using the proposed extraction and integration system, typical users can construct mashup Web applications easily and quickly. We also undertook an exploratory analysis and evaluation of UI components for large-scale reuse. This will provide reusable components in future developments.

As a future work, we plan to crawl a significant number of websites/webpages and construct a highly precise retrieval system for large-scale UI components. Moreover, we will explore additional flexible methods of integration and construct an open community for sharing mashup components.

VI. Acknowledgement

This work was supported by a Grant-in-Aid for Young Scientists B (No.26730158) from the Japan Society for the Promotion of Science (JSPS).

References

[1] Yahoo Pipes, URL: http://pipes.yahoo.com/pipes/ [accessed: 2015-02-15].

[2] J. Yu, B. Benatallah, R. Saint-Paul, F. Casati, F. Daniel, and M. Matera, “A framework for rapid integration of presentation components,” in The Proceedings of the 16th International Conference on World Wide Web, 2007, pp. 923–932.

[3] J. Yu, B. Benatallah, F. Casati, and F. Daniel, “Understanding mashup development and its differences with traditional integration,” Internet Computing, vol. 12, no. 5, 2008, pp. 44–52.

[4] J. Tatemura, A. Sawires, O. Po, S. Chen, K. S. Candan, D. Agrawal, and M. Goveas, “Mashup feeds: Continuous queries over Web services,” in The Proceedings of the 33th SIGMOD International Conference on Management of Data, 2007, pp. 1128–1130.

[5] M. Sabbouh, J. Higginson, S. Semy, and D. Gagne, “Web mashup scripting language,” in The Proceedings of the 16th International Conference on World Wide Web, 2007, pp. 1035–1036.

[6] E. M. Maximilien, H. Wilkinson, N. Desai, and S. Tai, “A domain-specific language for Web apis and services mashups,” in The Proceedings of the 5th international conference on Service-Oriented Computing, 2007, pp. 13–26.

[7] E. Wohlstadter, P. Li, and B. Cannon, “Web service mashup middleware with partitioning of XML pipelines,” in The Proceedings of 7th International Conference on Web Services, 2009, pp. 91–98.

[8] Q. Zhao, G. Huang, J. Huang, X. Liu, and H. Mei, “A Web-based mashup environment for on-the-fly service composition,” in The Proceedings of 4th International Symposium on Service-Oriented System Engineering, 2008, pp. 32–37.

[9] J. Fujima, A. Lunzer, K. Hornbaek, and Y. Tanaka, “C3W: clipping, connecting and cloning for the Web,” in The Proceedings of the 13th International World Wide Web conference, 2004, pp. 444–445.

[10] J. Wong and J. I. Hong, “Making mashups with marmite: Towards end-user programming for the Web,” in The Proceedings of the SIGCHI Conference on Human factors in computing systems, 2007, pp. 1435–1444.

[11] XML User Interface Language, URL: http://developer.mozilla.org/en-US/docs/Mozilla/Tech/XUL [accessed: 2015-02-15].

[12] R. Ennals and M. Garofalakis, “MashMaker: Mashups for the masses,” in The Proceedings of the 33th SIGMOD International Conference on Management of Data, 2007, pp. 1116–1118.

[13] Creating Web Clip Widgets, URL: https://support.apple.com/kb/PH14092 [accessed: 2015-02-15].

[14] F. Daniel and M. Matera, “Turning Web applications into mashup components: Issues, models, and solutions,” in The Proceedings of the 9th International Conference on Web Engineering, 2009, pp. 45–60.

[15] J. Guo and H. Han, “Analysis and design of programmatic interfaces for integration of diverse Web contents,” International Journal of Software Engineering and Knowledge Engineering, vol. 23, no. 10, 2013, pp. 1487–1511.

[16] Java Portlet, URL: http://www.jcp.org/en/jsr/detail?id=286 [accessed: 2015-02-15].

[17] Web Services for Remote Portlets, URL: http://www.oasis-open.org/committees/wsrp/ [accessed: 2015-02-15].

[18] Simple Object Access Protocol, URL: http://www.w3.org/TR/soap/ [accessed: 2015-02-15].

[19] Representational State Transfer, URL: http://rest.elkstein.org/ [accessed: 2015-02-15].

[20] Yahoo Query Language, URL: http://developer.yahoo.com/yql/ [accessed: 2015-02-15].

[21] Web Service Definition Language, URL: http://www.w3.org/TR/wsdl [accessed: 2015-02-15].

[22] HtmlUnit, URL: http://htmlunit.sourceforge.net/ [accessed: 2015-02-15].

[23] H. Han and T. Tokuda, “A method for integration of Web applications based on information extraction,” in The Proceedings of the 8th International Conference on Web Engineering, 2008, pp. 189–195.

[24] HTML DOM Script Object, URL: http://www.w3schools.com/jsref/dom_obj_script.asp [accessed: 2015-02-15].

[25] Web Application Description Language, URL: https://wadl.dev.java.net/ [accessed: 2015-02-15].

[26] News Search Documentation for Yahoo! Search, URL: http://developer.yahoo.com/search/news/V1/newsSearch.html [accessed: 2015-02-15].

[27] H. Han, P. Gao, Y. Xue, C. Tao, and K. Oyama, "Analysis and design: Towards large-scale reuse and integration of Web user interface components," in Information Reuse and Integration In Academia And Industry. Springer Verlag, 2013, pp. 133–162.

[28] Y. Cao, V. Yegneswaran, P. Porras, and Y. Chen, "A path-cutting approach to blocking XSS worms in social Web networks," in The Proceedings of the 18th ACM conference on Computer and communications security, 2012, pp. 745–748.