Virtual Browsing Environment for Mashups

Chun-Hsiung Tseng

Department of Computer Information and Network Engineering, Lunghwa University of Science and Technology

16F., No.95-1, Ln. 245, Sec. 2, Sichuan Rd., Banqiao Dist., Xinbei City 220, Taiwan (R.O.C.)

lendle_tseng@seed.net.tw

***Abstract*—More and more users today tend to use web as "a system of interlinked of services accessed via the Internet" instead of "a system of interlinked hypertext documents accessed via the Internet". Most such services rely on Mashup technologies. However, current technologies adopted are actually restrictive. In this research, the author proposes a virtual web browsing environment that is aimed at solving Mashup-related issues. The virtual web browsing environment simulates a real browser, that is, it has to be capable of parsing web pages into DOM structure and interpreting scripts. Scripts defined in these web pages will then be exposed as web services. In that way, to do Mashups, one simply "invoke" script functions defined on the target pages. The author believes this can be a good advance for Mashup-related technologies.**

***Keywords*— Mashup, Web Services, XML, Transformation, Information Extraction**

1. INTRODUCTION

The traditional definition of the World Wide Web (WWW) is "a system of interlinked hypertext documents accessed via the Internet". However, web users (human or non-human) today do expect/request more functionalities than traditional web infrastructures can provide. For instance, some users simply want to extract only what they want from the full data set provide by a web site. A practical example is the RSS service which extracts only a small portion of data (news titles, for example) from the source web site. Besides simple extraction, sometimes, users may even want to aggregate information extracted multiple data sources. For example, most online (web-based) stock services are capable of gathering related highlights of a specified stock code from several news web sites. From the author's point of view, these examples tell us that current web usage is undergoing a huge evolvement. More and more users today tend to use web as "a system of interlinked of services accessed via the Internet" instead of "a system of interlinked hypertext documents accessed via the Internet". Note that the term "service" refers to financial service, marketing service, news service, and data service, etc. We have already seen some partial advances. For example, igoogle [1] allows users to customize not only the look of their homepages but also the functionalities/services provided by their own homepages. With the help of gadget system, users can put desired tools onto their homepages. Furthermore, they can even make their own gadgets! These gadgets are capable of connecting to other web sites, extracting needed data, aggregating acquired data into desired format, and presenting the aggregated data with styles that can be partially customized by users. As a result, we have the imagination that we are utilizing the web as services.

What the gadget system tries to achieve is nowadays named as "Mashup", that is, the process of using and combining data, presentation or functionality from two or more sources to create new services [2], [3]. However, current technologies adopted are actually restrictive. Existing Mashup techniques typically adopt the following mechanisms:

1. Server-side service invocation. That is, the service consumer sends a request to a pre-specified endpoint. A program in the server-side will intercept the request, extract needed parameters, process the request, and then return the result. After receiving the response, the service consumer then decodes the result and display the result with a pre-specified format.
2. Web page scraping. Only few web sites offer server-side services. To achieve Mashup, most cases still rely on web page scraping. That is, the service consumer still sends a normal http request to the target web site. After the result web page is received, the service consumer tries to parse the result, and

extract the needed part. Then, the service consumer displays the parsed result with a pre-specified format.

The above mechanisms can by no means be broadly adopted. First of all, most existing web sites fail to support the first mechanism. Implementing server-side services require higher technical skills; running and maintaining server-side services incur higher costs and overhead. Even if a web site provides server-side services, in most cases these services can not cover all functionalities provided by the web site. Moreover, the lack of broad adoption further postpones the overall acceptance of standards of server-side services such as SOAP and REST [4], [5]. On the other hand, the web page scraping approach is rather unstable. Despite of the research efforts spent on it, existing web page scraping mechanisms are still too sensitive to modifications of the target web page. With high possibilities, web page scraping procedures have to be re-written whenever the target web pages are modified.

In this research, the author proposes a virtual web browsing environment that is aimed at solving the above issues. The virtual web browsing environment simulates a real browser, that is, it has to be capable of parsing web pages into DOM structure and interpreting scripts. Scripts defined in these web pages will then be exposed as web services. In that way, to do Mashups, one simply "invoke" script functions defined on the target pages through the virtual web browsing environment. The author believes this can be a good advance for Mashup-related technologies. Since the research is still ongoing, in this paper, we represent our concept and some partial results.

2. RELATED WORKS

2.1. Mashup

Currently, the usage of the WWW is undergoing a fundamental transformation. Rather than simple web surfing, web users nowadays request for an integrated view of information on the WWW. The way to compose information around the WWW is called "Mashup". In addition to ordinary end users, the transformation also affects enterprises [6]. According to [7], Mashup is the hallmark of the web 2.0. and refers to "an ad hoc composition technology of Web applications that allows users to draw upon content retrieved from external data sources to create entirely new services."

[8] implemented a Mashup tool named as "Damia", which was specialized for enterprises. Damia focused on ingestion of a larger set of data sources such as Notes, Excel, and XML. The implementation consists of a browser-based user interface, a server with an execution server, and a set of APIs for searching, debugging, and managing Mashups. [9] proposed another Mashup tool named as "Marmite", which was developed with the "End User Programming" concept. The implementation was capable of extracting content from one or more web pages and then processing it in a data flow manner. According to their study, both programmers and ordinary spreadsheet users had little difficulty using the system. [10] utilized "Programming-by-Demonstration" techniques to build a Mashup tool named as "Vegemite". With their system, users demonstrate how to collect data from desired web pages and Vegemite will record users' actions into scripts and then repeat these scripts to complete data extraction. In addition to research works, "Yahoo! Pipes" [11] is another popular Mashup tool. "Yahoo! Pipes" is a composition tool to aggregate content from around the web. With "Yahoo! Pipes", users can grab data from feeds, RSS, and some pre-defined web services, etc. Then, users can define data flows to aggregate grabbed data.

Although the number of existing Mashup tools/services is huge, current mechanisms for performing Mashup is actually very limited. According to [8], [12], existing Mashup mechanisms rely on either web service APIs (REST or SOAP) or web page scraping technologies. As mentioned in Section 1, both approaches have drawbacks and can require substantial amount of programming. According to [12], existing Mashup approaches require users to:

1. Understand web service APIs (if available) in order to invoke them.
2. Know how to perform web page scraping if no web service API available.
3. Know the data structure of input/output.

These are actually high entrance barriers for ordinary Mashup creators/programmers. Although tool supports are available in some cases, these tools can by no means being effective in all circumstances. As a result, although Mashup technologies attract research attentions, there are only few products on the market.

2.2. Web Information Extraction

Due to cost and difficulty issues, only few web sites provide web service APIs for Mashup. Until now, most Mashup services/tools such as [9], [13] actually rely on the web page scraping approach. Thus this approach can deserve more exploration.

Web page scraping falls into the category of "Web Information Extraction", which refers to the technology for extracting data from pages. According to [14], the purpose of web information extraction systems is to transform web pages into program-friendly structures. Unlike information retrieval (IR), which focuses on how to identify relevant documents, the purpose of information extraction is to produce structured data for post-processing. Web information extraction can be divided into the following steps: acquiring of input, analyzing of input, performing of extraction, and generating of output [15]. Among them, the most important step is the analyzing of input since this step covers several critical tasks including recognizing regions and discovering regularities. However, this step can be very tedious and difficult due to the fact that HTML is actually a language for the visual representation purpose only and the HTML source of a web page may be changed frequently. Moreover, even if two web pages contain similar contents with similar semantics, the representation may be very different [16].

Roughly speaking, web information extraction systems can be divided into four different types: manually constructed systems, supervised systems, semi-supervised systems, and unsupervised systems [14]. Manually constructed systems such as [17] require users to use provided programming or query languages for extraction and thus become extremely expensive. Supervised systems such as [18] require users to provide a set of pre-labeled web pages for training and then will automatically infer rules for extracting data from similar web pages. Semi-supervised systems such as [19] require no pre-labeled training pages, but post-efforts from the user is required to choose the target pattern and indicate the data to be extracted. Unsupervised systems, on the other hand, require no user interactions. However, they may only be applicable for data-rich regions, for example, the table part, of a web page. [20] is an unsupervised web information extraction system.

3. THE VIRTUAL WEB BROWSING ENVIRONMENT

As stated earlier, the virtual web browsing environment is an environment that simulates the work of a real browser. It is actually a server-side application waiting for requests from client programs. A typical usage scenario is a client program requests for the invocation of a script function residing in a given web page. After requests from clients being received, it fetches resources (HTML pages, script definitions, etc.) from specified urls, parses fetched resources, executes specified scripts in a simulation environment, and then returns the results. To realize these functionalities, the environment consists of five parts: the web service interface, the resource fetcher, the DOM simulator, the JavaScript simulator, and the data transformer. In the following sub sections, these parts will be described in detail. Figure 1 illustrates a typical usage scenario and these parts.

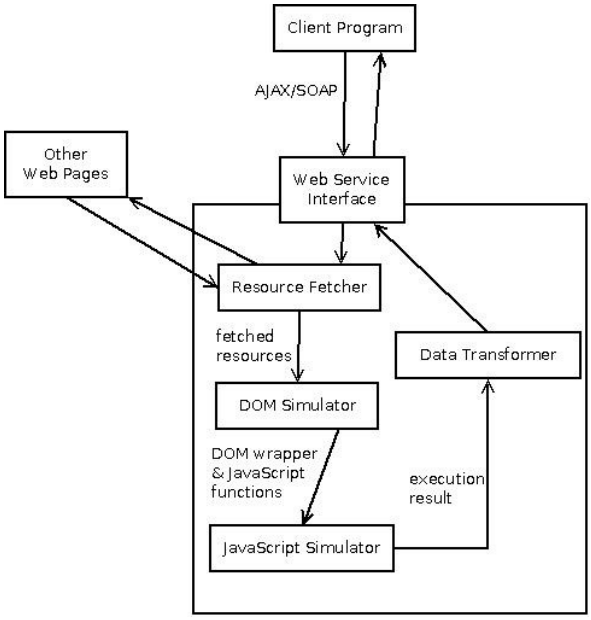

Figure 1. An overview of the Virtual Web Browsing Environment.

3.1. The Web Service Interface

For interoperability, the environment has a web service interface based on the SOAP over HTTP protocol. Although the design of the environment is mainly for web-based clients, the selection of the SOAP over HTTP protocol makes it possible for other types of clients to access the environment. For web-based clients, the simplest way to access the environment is by issuing AJAX [21] requests to the web service

interface. For non-web-based clients, there are numerous libraries such as [22] can be used for generating SOAP over HTTP compatible packets.

3.2. The Resource Fetcher

The resource fetcher is responsible for fetching resources (and dependencies) requested by a client program. In simple cases, the resource fetcher simply connects to the specified url and retrieve the returned data. For example, to fetch a http:// based url, the resource fetcher sends a "GET" command to the server hosting the url and receives the response. However, in practice, the retrieved resource may have dependencies on many other resources. As a result, the resource fetcher has to cooperate with the next component, the DOM simulator, to resolve the dependencies.

Another challenge is the protected resources. A protected resource may require authentication and secured connection. In such situation, we can utilize [23] to achieve the goal. Up to now, we are able to realize HTTP BASIC authentication and SSL connection with [23]. Since these are the most widely adopted protection mechanism for web based resources, our solution should be sufficient in most cases. In the future, we should extend our implementation to provide support for other popular protection mechanisms, such as FORM based authentication.

3.3. The DOM Simulator

Before a real browser can render a web page, it must at first parse the web page into DOM objects. The DOM simulator is thus an essential component of the virtual browsing environment. For each HTML resource retrieved by the resource fetcher, the DOM simulator performs the following steps:

1. Tidy the HTML resource by correcting improper tags.
2. Parse the fixed HTML resource into DOM objects.
3. Wrap the generated DOM objects with a DOM wrapper. The DOM wrapper simulates APIs listed in [24].
4. Cooperate with the next component, the JavaScript simulator, by injecting an instance of the DOM wrapper into the run-time space of the JavaScript Simulator.
5. Cooperate with the JavaScript simulator by extracting scripts defined in the target HTML page and registering the scripts into the JavaScript simulator.

The most challenging step is the first step, since some HTML pages do not follow the standard strictly. Currently, we simply try to fix unclosed tags and assume the rest of the target HTML page follows the XHTML standard. This assumption is reasonable, since most HTML pages nowadays are written using HTML authoring tools such as DreamWeaver and HTML pages generated by DreamWeaver do follow the XHTML standard. In the near future, we should integrate our component with third-party libraries, e.x., HTML Tidy [25], to fix more potential problems.

Besides, until now, we have only implemented DOM tree traversal APIs such as getElementById, getElementsByTagName, and getParentNode, getNextSibling, etc. and data retrieving APIs such as getNodeValue in the DOM wrapper. These APIs are usually invoked by script functions to locate specific nodes and to retrieve data on the nodes.

3.4. The JavaScript Simulator

The JavaScript simulator is a scripting engine supporting the execution of JavaScript functions defined on the target HTML page. There are several script languages allowed in an HTML page, e.x., JavaScript and VBScript. However, since JavaScript is the most widely adopted one among these languages, we only provide simulator for it.

Here, we adopt the Java Scripting environment [26]. It is a pluggable environment that supports various types of scripting languages. To support a new language, one simply registers a new interpreter into the environment and leaves other parts untouched.

The DOM simulator is responsible for registering declared JavaScript codes into the JavaScript simulator. Then the JavaScript simulator performs the following steps:

1. Evaluate the registered JavaScript codes. After this step, functions and global variables will be declared in the memory space of this session.
2. Execute the script function requested by the client program.
3. Cooperate with the next component, the data transformer, to perform needed transformation.
4. Return the result to the client program (possibly asynchronously).

The most challenging step is the third step. The virtual web browsing environment itself is a

server-side application. However, in some cases, a client program may expect a DOM node as the return value. In such situation, we have to serialize the requested DOM node into a string, apply necessary transformation (if requested), return the string to the client program, and then re-construct the DOM node. The data transformer will be detailed in the following sub section.

3.5. The Data Transformer

The data transformer comprises of three sub components: the serializer, the transformer, and the deserializer. Serializer and transformer are server-side components while deserializer is a client-side JavaScript-based component.

The tasks of serializer and deserializer are straightforward: the serializer converts a DOM node into a string and the deserializer converts the string back to a DOM node. The task of transformer deserves more explanation.

The goal of Mashups is to integrate data retrieved from multiple data sources (typically web sites). A major challenge of Mashups is the incompatibility of data from different data sources. For example, even if two HTML pages (from different web sites) contain data under the same subject, the markups may have completely different layout since the HTML language is, by definition, designed for layout instead of for data modeling. The author believes that this is the reason of why existing Mashups based on web page scraping can hardly achieve their goals. A possible solution is to transform HTML nodes from different web pages (but represent information of the same kind) into XML nodes adhering to the same XML schema. After the step, client program simply process the transformed XML node and does not have to worry about the original HTML data.

We only have partial results with regards to this part. There are several technologies that are capable of performing the transformation work. For example, XForms [27] and our previous research result, XUIB [28].

4. CONCLUSIONS AND FUTURE WORKS

In paper, we present our concepts and partial results of building a virtual web browsing environment to perform Mashups. Existing Mashup solutions typically rely on server-side service invocation or web page scraping. These approaches can not be widely adopted since there are only few web sites providing server-side services and web page scraping is rather unstable. With our solution, client programs can invoke JavaScript functions defined in other HTML pages and receive the results. To define JavaScript functions has much lower entrance barrier than to define server-side services, thus, the author believes that our approach is more adoptable.

Since the research and implementation is still ongoing, the most important future work is to complete the whole virtual web browsing environment. After the implementation being completed, we plan to integrate ontology-based methods into our result. Data from different web sites may not only have different layout but also different terminologies. Although the transformation capability implemented in the proposed solution can alleviate the data layout problem, the terminology impedance issue still remains. By adopting ontology-based methods, we can close the gaps between different sources further.

REFERENCES

[1] igoogle. http://www.google.com.tw/ig

[2] Mashup (web application hybrid). http://en.wikipedia.org/wiki/Mashup_%28web_application_hybrid%29

[3] Yu, J., Benatallah, B., Casati, F., and Daniel, F. 2008. Understanding Mashup Development. IEEE Internet Computing In Internet Computing 12, 44-52.

[4] SOAP. http://www.w3.org/TR/soap/

[5] Web Services Architecture. http://www.w3.org/TR/ws-arch/

[6] Jhingran, A. 2006. Enterprise Information Mashups: Integrating Information, Simply. In Proceedings of the 33rd international conference on Very large data bases, 2006, 3-4.

[7] Liu, X., Hui, Y., Sun, W., and Liang, H. 2007. Towards Service Composition Based on Mashup. In Proceedings of the IEEE international conference on Services computing, 2007, 332-339.

[8] Altinel, M., Brown, P., Cline, S., Kartha, R., Louie, E., Markl, V., Mau, L., Nq, Y., Simmen, D., and Singh, A. 2007. Damia: a data mashup fabric for intranet applications. In Proceedings of the 33rd international conference on Very large data bases, 2007, 1370-1373.

[9] Wong, J. and Hong, J. 2007. Making mashups with marmite: towards end-user programming for the web. In Proceedings of the SIGCHI conference on Human factors in computing systems, 2007, 1435-1444.

[10] Lin, J., Wong, J., Nichols, J., Cypher, A., and Lau, T. 2009. End-user programming of mashups with vegemite. In Proceedings of the 13th international conference on Intelligent user interfaces, 2009, 97-106.

[11] Yahoo! Inc. Pipes, http://pipes.yahoo.com/pipes/

[12] Lorenzo, G., Hacid, H., Paik, Y., and Benatallah, B. 2009. Data integration in mashups. ACM SIGMOD 38, 59-66.

[13] Ennals, R., and Garofalakis, M. 2007. Mashmaker: mashups for the masses. In Proceedings of the 2007 ACM SIGMOD international conference on Management of data, 2007, 1116-1118.

[14] Chang, C.H. and Girgis, M. 2006. A survey of web information extraction systems. IEEE Transactions on Knowledge and Data Engineering 18, 1411-1428.

[15] Jirkovský, V. and Jel'ınek, I. 2009. Proposing of modular system for web information extraction. In Proceedings of the 2009 international conference on Computer systems and technologies, 2009, 1-4

[16] Meng, X., Hu, D., and Li, C. 2003. Schema-guided wrapper maintenance for web-data extraction. In Proceedings of the 5th ACM international workshop on Web information and data management, 2003, 1-8.

[17] Arocena, G.O. and Mendelzon, A.O. 1998. WebOQL: Restructuring documents, databases, and webs. In Proceedings of the 14th IEEE international conference on Data engineering, 1998, 24-33.

[18] Laender, A.H.F., Ribeiro, B., and Silva, A.S. 2002. DEByE---Data extraction by example. Data and Knowledge 40, 121-154.

[19] Chang, C.H. and Kuo, S.C. 2004. OLERA: A semisupervised approach for web data extraction with visual support. IEEE Intelligent Systems 19, 56-64.

[20] Wang, J. and Lochovsky, F.H. 2003. Data extraction and label assignment for web databases. In Proceedings of the 12th international workshop on World wide web, 2003, 187-196.

[21] AJAX. http://en.wikipedia.org/wiki/Ajax_%28programming%29

[22] Java API for XML Web Services. http://en.wikipedia.org/wiki/Java_API_for_XML_Web_Services

[23] Apache: HttpClient. http://hc.apache.org/httpclient-3.x/

[24] W3C: Document Object Model. http://www.w3.org/DOM/

[25] HTML Tidy. http://tidy.sourceforge.net/

[26] Java Scripting. http://download.oracle.com/javase/6/docs/technotes/guides/scripting/programmer_guide/index.html

[27] W3C: XForms. http://www.w3.org/MarkUp/Forms/

[28] Lendle Tseng, Y.S. Kuo, Hsiu-Hui Lee, and Chuen-Liang Chen: XUIB: XML to User Interface Binding. In Proceedings of the 2010 ACM Symposium on Document Engineering, 2010, 51-60