Partial Information Extraction Approach to Lightweight Integration on the Web

Junxia Guo[1], Prach Chaisatien[1], Hao Han[1,2],
Tomoya Noro[1], and Takehiro Tokuda[1]

[1] Department of Computer Science, Tokyo Institute of Technology
Ookayama 2-12-1-W8-71, Meguro, Tokyo 152-8552, Japan
{guo,prach,han,noro,tokuda}@tt.cs.titech.ac.jp
[2] Digital Content and Media Sciences Research Division, National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda, Tokyo 101-8430, Japan
han@nii.ac.jp

Abstract. We present partial information extraction approach to lightweight integration on the Web. Our approach allows us to extract dynamic contents created by scripts as well as static HTML contents. Our approach has three application areas: automatic generation of Web services from Web applications, automatic integration of Web applications with Web services on desktop computers, and automatic integration of mobile phone applications with Web applications and Web services on mobile phones.

Keywords: partial information extraction, lightweight integration, desktop computers, mobile phones.

1 Introduction

The purpose of this paper is to present partial information extraction approach to lightweight integration on the Web and show that there are three important application areas of our approach in the lightweight integration.

The traditional way of lightweight integration on the Web is to integrate Web services by writing a script or a program invoking those Web services. This approach is natural but not directly applicable to the integration of the Web applications which do not have Web service APIs.

Some of the other existing approaches to lightweight integration on the Web are as follows. The first approach is to integrate the Web sources in predefined library using GUI interface as in Yahoo Pipes [24]. Users do not need to do programming, but they cannot add new Web sources into the library. The second approach is to use static Web contents extracted from Web applications as in Dapp Factory [11]. Users can use information extracted from Web applications and RSS to do lightweight integration, but they cannot extract dynamic Web contents such as the clock part of Localtimes.info [18] created by a script.

Our partial information extraction approach is a technique to extract partial contents from Web pages in one Web site according to GUI-based definition of

F. Daniel and F.M. Facca (Eds.): ICWE 2010 Workshops, LNCS 6385, pp. 372–383, 2010.

partial contents of a sample Web page in the Web site. Our technique is able to extract both static and dynamic contents created by scripts using the method called hide-and-display method.

Thanks to the hide-and-display method, our approach allows us to construct Web service functions from Web applications which do not have Web services. Our approach allows us to integrate both Web applications and Web services on desktop computers using descriptions. Our approach also allows us to integrate Web applications with Web services and mobile phone applications on mobile phones using descriptions.

The orgnization of the rest of this paper is as follows. In Section 2, we explain the detail of our partial information extraction approach. Section 3 explains the methods that can generate Web services from Web applications automatically. In Section 4, we explain our method to integrate Web applications with Web services on desktop computers. We present the method that integrate Web applications with Web services and mobile phone applications on mobile phones in Section 5. We evaluate our approach in Section 6. Finally, in Section 7, we give our conclusion.

2 Partial Information Extraction Approach

Classical partial information extraction methods allow us to extract parts of Web pages which are static data such as texts and images.We refer to these methods as first-generation methods. First-generation methods can extract titles and contents from news site articles, for example. More and more Web sites generate Web pages containing client-side scripts such as JavaScript and Flash instead of ordinary static HTML pages. Our partial information extraction method allows us to extract parts of Web pages dynamically created by client-side scripts.

We use the term static Web content to represent the content shown on the Web page directly in the HTML source file, for example texts and images. And we use the term dynamic Web content to represent the content created by client-side scripts dynamically, which is shown on the Web page but cannot be found directly in the HTML source file.

Our partial information extraction approach is as follows.

For static Web content, we supply two kinds of extracting methods. One method is to extract the target part in text format excluding the tags of the HTML document. For example, the extracted information of a picture is the value of attribute "src" of node <img>, and the extracted information of a link is the value of attribute "href" of node <a>. The details of the extracting algorithm can be found in [3]. Another method is to use the same method as dynamic Web content, if the users want to keep the same layout of target parts with the original Web page. Users can choose the second method by setting the type of extracting Web content as "dynamic".

For the dynamic Web content, we use hide-and-display method instead of using the extracting method of static Web content to keep the functionality of dynamic Web content. This is because the scripts usually use DOM operation

to control the dynamic parts of Web pages and sometimes access to the elements outside of the target parts such as the hidden values. Therefore, if we just extract the target parts, the original executing environment of scripts may be broken. The hide-and-display method works as follows. First, we extract the whole HTML document of the target Web page. Then, we hide all HTML nodes in the extracted HTML document of the target Web page. Finally, we show the target Web contents and do the necessary settings. The details of the hide-and-display method can be found in [4].

In order to describe the necessary information for the extraction and emulation, we use XML-based notation called Web Application Contents Description Language (WACDL). The WACDL file includes the information about the target Web contents, such as the start page URL of the Web application, the path of target Web content, the type of the Web content and so on. The detail about the definition and format of the WACDL file can be found in [2].

By the Web contents extraction method and the hide-and-display method, we can get any parts from any Web applications, and maintain the functionalities of dynamic Web contents. Based on this description-based partial information extraction method, we can generate Web services from general Web applications automatically. And we can integrate both Web applications with Web services on desktop computers. Also we can integrate Web applications with Web services and mobile phone applications on mobile phones based on this method. We explain the three kinds of applications in the following sections.

3 Automatic Generation of Web Service Functions

Our partial information extraction approach allows us to generate Web service functions in mainly two ways: static construction and dynamic construction. In static construction, expected input and output information is extracted from Web pages of one Web site and construct a table. This table offers a Web service function returning output information for given input information. In dynamic construction, input information is sent to expected Web application and output information is extracted from the resulting Web page.

An ordinary Web service responds to the requests from users by returning the data from server-side information resources, usually a database. For our Web service construction, the information resources are the Web applications. Thus, our method needs to extract the information from the Web applications to generate the resulting tables. Based on our partial-information-extraction-method and resulting-table-query-method, users can generate Web services as described in the following steps: Firstly, users select the target parts from Web pages to generate the extraction patterns,which consist of the names and data types for the selected parts. Secondly, users run the constructed Web services at a proxy server, and configure Web service interfaces for them. Thirdly, the Web services use the extraction patterns to extract the partial information statically or dynamically according to the requests of users to create the resulting tables. Finally, the Web services search for the desired information from the resulting tables and respond to the users with the corresponding field values.

We refer users to run our constructed Web services at a proxy server. In the proxy server, the extraction patterns are used to extract the partial information and the resulting tables are generated. By using designed interface, the users send the requests to proxy server and get the responses from the resulting tables.

Usually, for the users of an ordinary Web application that is suitable for Web service construction, there are two basic types of methods to send their requests. The first type is to enter a keyword into an input field by keyboard and click the submit button by mouse to send the query. The second type is to click a link or an option in drop-down list in Web page by mouse to view a new Web page. For the first type, the URLs of target Web pages are changed by the requests of users. We call them Dynamic URLs. For the second type, the URLs are not changed by the requests of users. We call them Static URLs. In order to get the URL from all kinds of Web applications, we use HtmlUnit [15] to emulate the submitting operation.

In the Web applications, some information is static or changed periodically. We do not need to extract them dynamically, especially the static information with static URLs, after the users send the requests every time. We define three kinds of extraction frequencies for Web services: dynamic extraction, periodic extraction and static extraction.

1. Dynamic Extraction: The Web services extract the partial information dynamically after the users send the requests. It is suitable for the information extraction from real time system or the search result Web pages with dynamic URLs such as CNN news search result pages.
2. Periodic Extraction: The Web services update the resulting table by extracting the partial information at a regular interval such as one hour, one day or one week. For example, the weekly ranking of hot items is updated once a week at Rakuten [19].
3. Static Extraction: The extracted information is unchanged in a long period such as the basic information of a country/region. For example, the capital city and area information of most countries are unchanged and the population and life expectancy information remain unchangeable in a long period at BBC Country Profiles. Users can set the extraction frequencies for the constructed Web services according to the actual update frequencies of target Web applications.

Web services are self-contained and self-describing. The most-used style architectures of Web services are SOAP and REST. SOAP stands for Simple Object Access Protocol. Google implements their Web services to use SOAP, and we can find SOAP Web services in a number of enterprises' software. REST stands for Representational State Transfer. A number of new web services are implemented using a REST style architecture these days rather than a SOAP one, such as the Web services of Yahoo, Flickr [13] and del.icio.us [12]. Compared with SOAP, REST is lightweight and easy to build, and provides the readable results.

Figure 1 shows a simple example that uses the automatic generated Web service to get the top ten news' title, link URL and update time from CNN News [10].

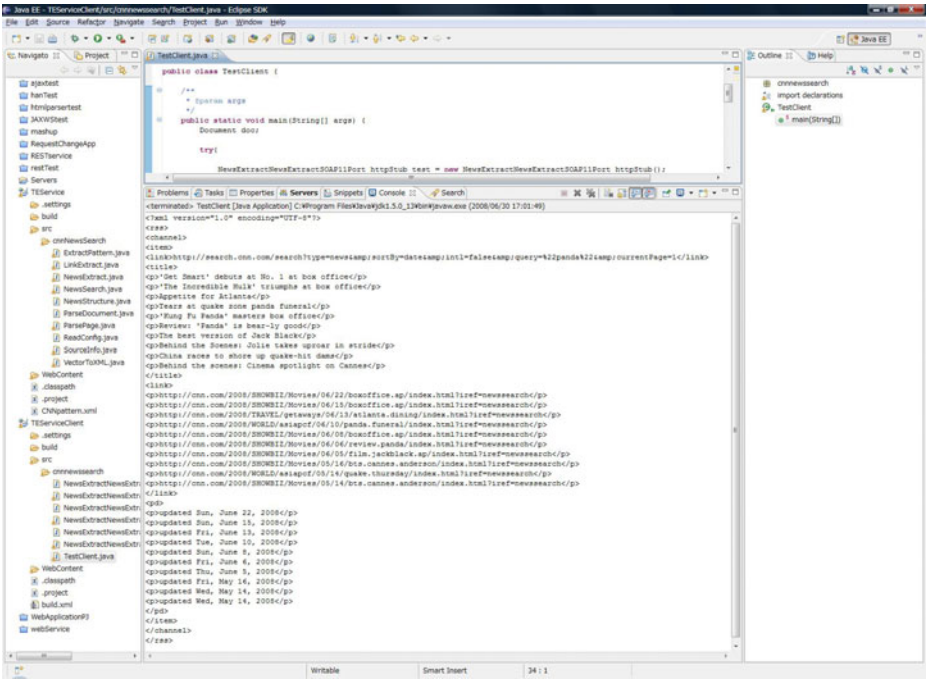

Fig. 1. Response of an Automatic Generated Web Service

4 Lightweight Integration on Desktop Computers

Our partial information extraction approach allows us to integrate Web applications with Web services. Web applications usually return resulting Web pages of much information. Our partial information extraction can select an expected part of the resulting Web pages.

To use parts of Web applications as mashup components, a normal method is that we use partial information extraction method to extract certain parts from Web applications and use them as mashup components. Most existing research work can only extract static Web content to generate mashup components. But based on our approach, we can extract not only static content but also dynamic content to generate mashup components. Furthermore, it allows us to deliver information between the generated mashup components.

Based on our approach,we build an example mashup application which integrates four components extracted from four Web applications: Localtimes.info[18], BBC Country Profiles[9], Weatherbonk.com[21] and ABC News[6]. After giving a country name, for example "Japan", and clicking the "Search" button, we can get a resulting Web page as shown in Figure 2, where the following components can be found.

Component 1. The local clock of the given country's capital city from the Localtimes.info Web site. The local clock is created by client-side script and

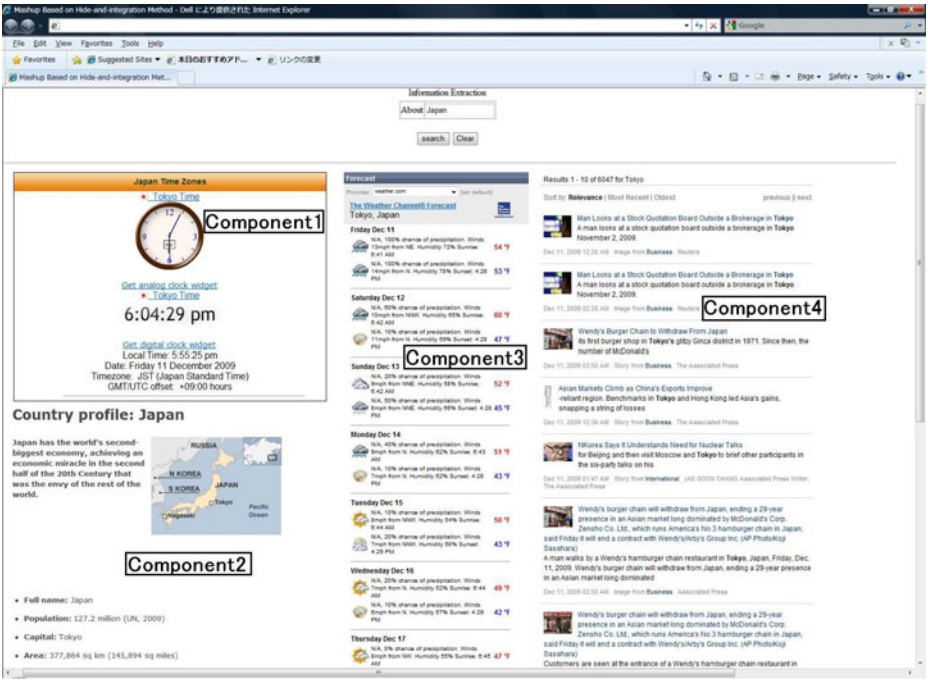

Fig. 2. Example of Mashup Web Application

the image changes every second. This component receives input parameter from the input-box of the user-interface.

Component 2. Introduction, location map and basic information of the given country name from BBC Country Profiles. The capital city's name extracted here will deliver to component3 and component4 as searching keyword. This component also receives input parameter from the input-box of the user-interface.

Component 3. The capital city's weather information from Weatherbonk.com. This component receives input parameter–capital city's name from "Component 2".

Component 4. The latest news articles from ABC News about the capital city. This component receives input parameter–capital city's name from "Component 2".

We create the mashup components one by one according to the recorded order in the WACDL file. As a result, the components can only use the information extracted from the component recorded before itself as the searching keyword. We support two kinds of basic information transfer styles with the above restriction. One style is that transfer information from one component to other all components. The other one style is that transfer information one by one sequentially. We also support the mix of these two kinds of information transfer styles in one

mashup application. We do not support the information transfer from multiple components to one component by now. We do not have technical problem on addressing this kind of usage with our approach, but it is different to create the rule of the combination which is extracted from multiple Web applications. We currently only support string type of information to be transferred between mashup components. Because there are many potential problems for other types of information. For example, the numerical value, although we can treat it as string when we extract or transfer the data. But we must make sure that the numerical value's format of the receiving component is same as the extracted data.

5 Lightweight Integration on Mobile Phones

Traditionally integration of Web applications with Web services on mobile phones uses construction of integrated pages rather than construction of integrated mobile phone applications. Our partial information extraction approach allows us to construct mobile phone applications by integrating Web applications with not only Web services but also mobile phone applications on mobile phones. Mobile phone applications deliver unique capabilities such as GPS location service, voice recognition and camera/image processing applications. Based on our partial information extraction approach, we present a description-based approach for the integration of mobile mashup applications combining partial extracted Web applications and mobile phone applications.

First, we present the image of a mobile mashup application example shown in Figure 3 to demonstrate how the description-based approach is used in composing a mashup application. The example consists of the integration of a mobile phone's user input interface with external Web resources. The mashup application aids users by shortening the interaction sequences. These sequences may conventionally require users to copy and paste one output from one section for using in another section. Furthermore, the partial extraction method reduces the amount of data and programming efforts using the native device programming code (such as network connection and HTML tags/text processing).

The objective is to integrate a mashup application which translates a product's bar-code into a product name, search for related information via Web applications based on our information extraction approach, and then display the output on the mobile phone. The following must be considered prior to the configuration process:

- Active components such as the Bar-code scanner component and the Web display component. These components require user interaction.
- Web application components which can use parameters generated from the Web extraction tool (i.e., Amazon.com [7] and Goodreads.com [14]).

First, users use the Web extraction tool shown in Figure 4 to generate parameters for their integrations, and then begin to configure each component in the

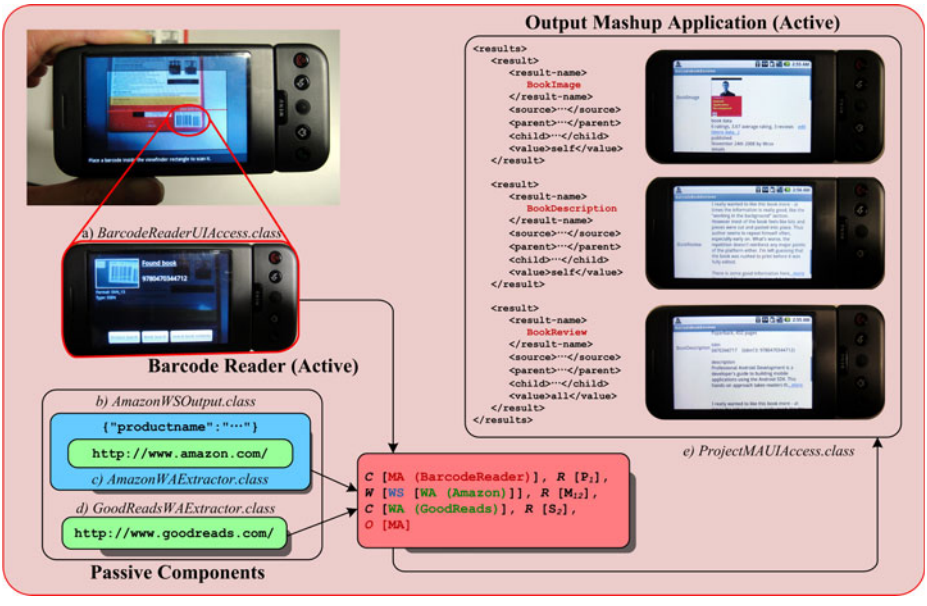

Fig. 3. The Image of a Book Reviews Bar-code Reader Mashup on Mobile Phone

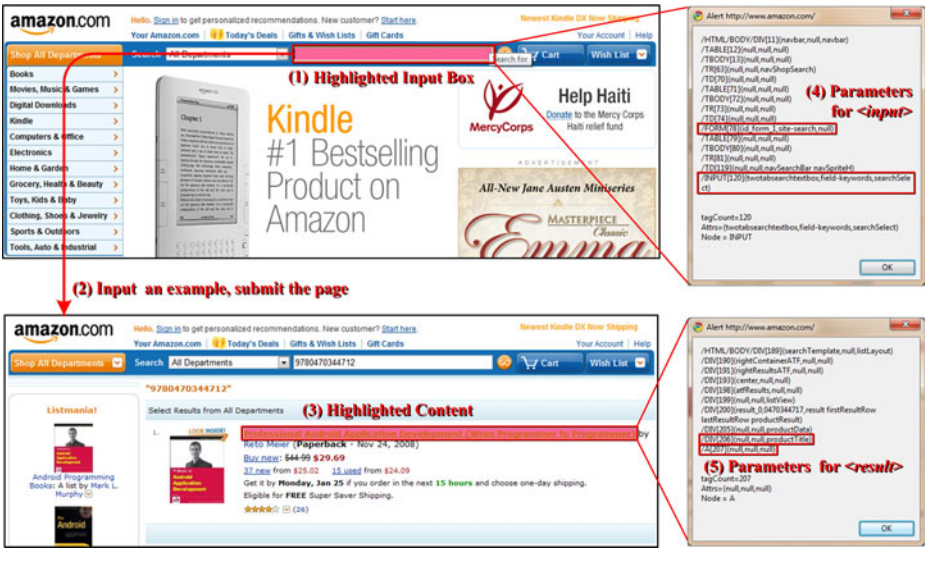

Fig. 4. Sample use of the Web extraction tool

integration. Users are required to install the bar-code reader application manually and need to know the Android's Intent configurations of the bar-code reader application.

In the process for generating this sort of mashup application, the Android's Android Application Package (APK) file [8] contains two class files concerning the access to user interface while the i-jetty's Web Application Archives (WAR) file [20] contains three class files. Each generated class file performs a specific task as follows:

a) BarcodeReaderUIAccess.class - accesses the bar-code reader interface via Intent.
b) AmazonWAExtractor.class - extracts the information from Amazon.com [7] using parameters from a).
c) AmazonWSOutput.class - generates JSON message from b).
d) GoodReadsWAExtractor.class - extracts information from Goodreads.com [14] using parameters from c).
e) ProjectMAUIAccess.class - accesses the user interface on the mobile phone to display output from d).

In intended use, after users manually install the APK file and the WAR to the mobile phone, user can access the bar-code scanner interface from the home screen.

6 Evaluation

This section evaluates our approach and application examples. Our work aims to support the flexible reuse of general Web applications in the lightweight integration on the Web. We build several mashup applications with various parts from different kinds of Web applications, and prove that our approach is applicable to general Web applications. We evaluate our approach from three aspects. First, we evaluate the extracting accuracy, which is the base of generating mashup components. Then we evaluate the efficiency of our approach for lightweight integration on desktop computers by comparing with other existing research works. Finally, we evaluate the efficiency of our approach for lightweight integration on mobile phones.

We choose several Web applications to generate components and build mashup applications. The experimental results about extracting accuracy are given in Table 1.

As the experimental results show, the average extracting accuracy is more than 90%. We checked the extracting results manually and counted the parts that had been extracted correctly. Our approach is based on the assumption that the searching result pages of the same request function are the same or similar within a Web application. But some of the searching result pages may be very different from other pages. As a result, among the all 201 countries and regions that we have checked in the BBC Country Profiles, we got correct results from 194 web pages. At present we support one path to describe the location

Table 1. Experimental Results of Extracting Accuracy

Web Application	Input Type for Search	Input Parameter	Num. of Checked Web Pages	Num. of Extracting Parts	Num. of Correct Pages	Percent of Correct Pages
abc News	Input Box	city name (Beijing, Tokyo ...)	194	1 (text, link)	194	100%
BBC Country Profiles	Option List	country name (Japan, Italy ...)	201	4 (text, image, bullet list)	194	96%
Localtimes	Link List	country name (Canada, Russia ...)	72	1 (dynamic content)	72	100%
Weatherbonk	Input Box	city name (Beijing, Tokyo ...)	194	1 (dynamic content)	189	97%
Yahoo Finance [22]	Input Box	stock name (AAPL, GOOG ...)	36	1 (dynamic content)	36	100%
Infoplease [17]	Link List	country name (Canada, Russia ...)	204	5 (image, text)	186	91%
Yahoo News [23]	Input Box	city name (Beijing, Tokyo ...)	189	1 (text, link)	189	100%
Total	—	—	1090	—	1060	97%

of "targetContent". We would like to support multiple paths to describe the location of "targetContent" and allow the users to decide how many paths they would like to use according to the accuracy they want.

We delivered the name of the capital city extracted from BBC Country Profiles to the ABC News and Weatherbonk. We got 194 (100%) correct results from the ABC News, and 189 (97%) correct results from the Weatherbonk. The cause of error is that some country/region names are the same as their capital city names. But in the Weatherbonk Web application, when we input the city's name, it is treated as a country name and we get a different page.

As we mentioned, there are mainly two kinds of existing approaches of lightweight integration on the Web. The first kind of approach is to integrate the Web sources in predefined library using GUI interface as Yahoo Pipes, iGoogle [16], Mixup [5] and so on. The users do not need programming, but they cannot add new Web sources into the library. The second kind of approach is to use static contents extracted from Web applications, as Dapp Factory [11], C3W [1] and so on. Users can use information extracted from Web applications and other Web sources to do lightweight integration, but they cannot extract dynamic contents. Based on our partial information extraction approach, we can extract not only static parts but also dynamic parts to do lightweight integration with other

Web sources. Furthermore, we can deliver information between the extracted parts.

Our partial information extraction method can be used with mobile application to do lightweight integration on mobile phones. In the configuration process of the mashup application presented in the example shown in Figure 3, users are required to use our Web extraction tool. We use the tool to specify and generate parameters to point to one result from a resulting list. To show more results, more parameters have to be specified which causes users to repeatedly use the Web extraction tool to generate more parameters. In general programming, programmers can write extraction loops; however, extraction loops is not presented in our method. In the case where many results are preferred, it is recommended that wrap the information extracted from the Web application using the Web Service Interface Wrapper so that users can apply the wrapper's outputs with typical programming.

To conclude, our method to perform Web information extraction on mobile phones using description-based configurations still requires manual works. The solution is to make the data flows diverge to parse the DOM tree remotely with other Web servers. In this case, we have to redesign the system architecture to be able to connect to external Web servers to do the configured tasks. Mobile phone hardware constraints and network latency also yields time lag when performing Web extraction. It is better to reduce the complexity of the DOM tree before the parsing process begins.

7 Conclusion

We have presented partial information extraction approach to lightweight integration on the Web. Our approach allows us to extract dynamic contents created by scripts as well as static HTML contents. We presented three application areas of our approach. Automatic generation of Web services from Web applications, automatic integration of Web applications with Web services on desktop computers, and automatic integration of Web applications with Web services and mobile phone applications on mobile phones.

References

1. Fujima, J., Lunzer, A., Hornbak, K., Tanaka., Y.: C3W: Clipping, Connecting and Cloning for the Web. In: Proceeding of the 13th International Conference on World Wide Web (2004)
2. Guo, J., Han, H., Tokuda, T.: A New Partial Information Extraction Method for Personal Mashup Construction. In: Proceeding of the 19th European-Japanese Conference on Information Modelling and Knowledge Bases (2009)
3. Han, H., Tokuda, T.: WIKE: A Web Information/Knowledge Extraction System for Web Service Generation. In: Proceeding of the 8th International Conference on Web Engineering (2008)

4. Han, H., Guo, J., Tokuda, T.: Towards Flexible Integration of Any Parts from Any Web Applications for Personal Use. In: First International Workshop on Lightweight Integration on the Web (Composable Web 2009) (2009)
5. Yu, J., Benatallah, B., Saint-Paul, R., Casati, F., Daniel, F.: A Framework for Rapid Integration of Presentation Components. In: Proceeding of the 16th International Conference on World Wide Web (2007)
6. ABC News, http://abcnews.go.com/
7. Amazon.com, http://www.amazon.com/
8. APK file, http://developer.android.com/guide/appendix/glossary.html
9. BBC Country Profiles, http://news.bbc.co.uk/2/hi/country_profiles/default.stm
10. CNN News, http://www.cnn.com/
11. Dapp Factory, http://www.dapper.net/
12. Del.icio.us, http://delicious.com/
13. Flickr, http://www.flickr.com/
14. Goodreads.com, http://www.goodreads.com/
15. HtmlUnit, http://htmlunit.sourceforge.net
16. iGoogle, http://www.google.com/ig/
17. Infoplease, http://www.infoplease.com/countries.html
18. Localtimes.info, http://localtimes.info/
19. Rakuten, http://www.rakuten.co.jp/
20. WAR file, http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/WCC3.html
21. Weatherbonk.com, http://www.weatherbonk.com/
22. Yahoo Finance, http://finance.yahoo.com
23. Yahoo News, http://news.yahoo.com/
24. Yahoo Pipes, http://pipes.yahoo.com/pipes/