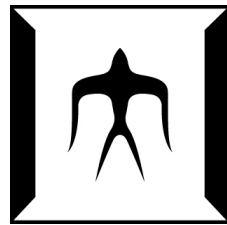


論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Description Based Approach for Flexible Integration of Web Services and Web Applications
著者(和文)	郭俊霞
Author(English)	Junxia Guo
出典(和文)	学位:博士(学術), 学位授与機関:東京工業大学, 報告番号:甲第9348号, 授与年月日:2013年9月25日, 学位の種別:課程博士, 審査員:徳田 雄洋,米崎 直樹,佐伯 元司,権藤 克彦,西崎 真也
Citation(English)	Degree:Doctor (Academic), Conferring organization: Tokyo Institute of Technology, Report number:甲第9348号, Conferred date:2013/9/25, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Description Based Approach for Flexible Integration of Web Services and Web Applications



Junxia Guo

Advisor: Professor Takehiro Tokuda

Department of Computer Science

Graduate School of Information Science and Engineering

Tokyo Institute of Technology

A thesis submitted for the degree of

Doctor of Philosophy (PhD)

June 2013

Abstract

Mashup technology integrates information or functionality from two or more sources to build a new Web page or application. Web services and applications are widely used Web sources. Various research work and industrial tools have appeared to ease the creation of mashups. However, the existing approaches have limitations of the integration of Web applications without programming. Moreover, the cooperation of Web applications and REST services without programming in the mashups is not supported.

We propose a description based approach that allows users to build mashups with general Web applications and REST services without programming. This approach supports the cooperation between the Web applications and REST services in mashups. A Java-based library is designed to help the users to build mashups through programming more easily. The implementation and experiments show that our approach is efficiently applicable to the general Web applications and REST Services.

Acknowledgements

I am deeply grateful to many people for supporting this work. I would like to express my gratitude to Professor Takehiro Tokuda of Tokyo Institute of Technology for supervising this work and dissertation. I have greatly appreciated his continuous support and valuable advice. I would like to thank Assist. Prof. Tomoya Noro and all members of Lab Tokuda for their helps and supports. I received a lot of insightful comments and suggestions that became great help to improve my work. My senior, Dr. Hao Han, gave me helpful comments and encouragements.

This work would not have been possible without support from my family. I am deeply grateful to my husband, who gave me generous support and warm encouragement to complete my work. I would to extend my gratitude to my parents. This dissertation would not be completed without all of the long-time support they have given to me, therefore this dissertation is dedicated to my family.

Contents

List of Figures	viii
List of Tables	x
1 Introduction	1
1.1 Motivation	1
1.2 Organization of the Thesis	3
2 Background and Related Work	5
2.1 Background	5
2.2 Related Work	7
3 Partial Information Extraction	10
3.1 Information Extraction Technologies	11
3.2 Feature-Expression of Our Extraction Method	12
3.3 Extraction of Static Web Contents	13
4 Hide-and-Display Method	18
4.1 Overview	18
4.2 Structural Analysis	19
4.3 Selection of the Extracting Web Contents	21
4.4 Information Extraction	22
4.4.1 Dynamic Web Contents	24
4.4.1.1 Compression Method	26
4.4.1.2 Visibility Control Method	27
4.4.2 Static Web Contents	28
4.5 Extracted Result	28

5	Description Based Mashup of Web Applications	30
5.1	Overview of the Approach	30
5.2	Mashup Application Template	32
5.3	Definition of Description Files	33
5.4	Process of DBMWA Method	36
5.4.1	Generation of Mashup Components	37
5.4.1.1	Cooperation of Web Applications	38
5.4.1.2	Component Consisted of Multiple Segments	39
5.5	Implementation of DBMWA Method	41
6	Integration of Web Applications and REST Web Services Without Programming	44
6.1	Architecture of M-WA&RS Method	45
6.1.1	Extension of the Segment Information Description File	46
6.1.2	Extension of the Hide-and-Display Method	49
6.1.2.1	Converting the Result of REST Service to HTML Format	49
6.1.2.2	Generating Segment from REST Web Service	50
6.2	Implementation of M-WA&RS Method	51
7	A Method for Flexible Integration through End-user Programming	55
7.1	Constituents of the Library	57
7.1.1	Functions for Extraction	57
7.1.2	Functions for Supporting Integration	60
7.2	Building Mashup Applications Based on the Library	62
8	Evaluation	69
8.1	Experimental Results	69
8.1.1	Applicability of Hide-and-Display Method	69
8.1.2	Extraction Accuracy of Hide-and-Display Method	71
8.1.3	Applicability of M-WA&RS Method	72
8.1.4	Applicability of the Java Library for End-user Programming	75
8.2	Comparison With Existing Approaches	78
8.3	Security Factors	78

CONTENTS

9 Conclusion	80
Bibliography	82
A Mashup Application Template	89
B MSD and SID File of Mashup Application Example 3	102
C Part of the MSD and SID File Used in Mashup Application Example 4	107

List of Figures

2.1	Top APIs for Mashups	6
3.1	Process of Fixing the Location of a Web Content	14
3.2	Static Web Content	15
3.3	Dynamic Web Content	15
3.4	Text Extraction	16
3.5	Object Extraction	16
3.6	Link Extraction	17
3.7	Continuous Type	17
4.1	The Process of Hide-and-Display Method	20
4.2	Examples of Request-Submit Functionalites	21
4.3	Path Reader	22
4.4	Script codes in HTML Tag	25
4.5	Script codes in Script File on Web Server	26
4.6	Extracting Result	29
5.1	Description Based Mashup of Web Applications	31
5.2	Layout of Mashup Application Template	32
5.3	Generator of Description Files	35
5.4	Description Based Mashup of Web Applications	37
5.5	Process of Information Transfer	39
5.6	Mashup Application Example 1	42
5.7	Mashup Application Example 2	43
6.1	Architecture of the M-WA&RS Method	46
6.2	Converting the Returned Result of Services into HTML Format	49

LIST OF FIGURES

6.3	Mashup Application Example 3	51
6.4	Mashup Application Example 4	53
6.5	Image of Using Web Service API	53
7.1	Overview of Using the Library	57
7.2	Mashup Application Using the Library	63
7.3	Mashup Application Using the Library	64
7.4	Information Extracted from World Travel Guide	67
7.5	Information Extracted from Yahoo Travel	68
8.1	Average time of Create Information of One Segment	75

List of Tables

5.1	Contents of the Mashup Application Example 1	42
6.1	Contents of the Mashup Application Example 3	52
6.2	Contents of the Mashup Application Example 4	54
7.1	Parameter Definition of <i>ExtractDynamicContent</i> Function	59
7.2	Parameter Definition of <i>ExtractFmREST</i> Function	60
7.3	Web Sources List of Travel Information Mashup Application	66
7.4	Functions Used in Travel Information Mashup Application	67
7.5	Code Size of Functionalities in Travel Information Mashup Application .	68
8.1	Applicability of Hide-and-Display Method	70
8.2	Experimental Results about Extracting Accuracy	71
8.3	Mashup Novels for User Study of M-WA&RS Method	73
8.4	Basic Information of End-users' Mashup Applications	77

Listings

4.1	Example Codes of Emulating InputBox-type-submission	23
4.2	Show Target Web Contents	27
5.1	Merge the XML Type of Extracting Results	40
6.1	Convert JSON Format to XML Format	50
7.1	Call <i>ExtractDynamicContent()</i> by Using JavaScript	65

Chapter 1

Introduction

1.1 Motivation

With the development of World Wide Web, people can do many kinds of things through the Web. For example, people can check news, enjoy videos, make friends, chat with friends, do shopping and so on. With the usage of Web, more and more information has become available on the Web. The Web can be considered as an infinite source of information. Information on the Web is supplied in different forms. There are mainly two kinds of forms, Web application and Web service.

Web services are built to do communication between two electronic devices over a network. The W3C defines a web service as a software system designed to support inter-operable machine-to-machine interaction over a network. Web services can be designed in different styles. The two most common styles of Web services are Simple Object Access Protocol (SOAP) and REpresentational State Transfer (REST) (22). Web applications are built based on the Web technic, which users can access through Web browser or client side application to get or transfer data. A Web application is hosted in a browser-controlled environment, such as a Java applet, coded in a browser-supported language, such as JavaScript, HTML. Web applications are popular due to the ubiquity of web browsers, and the convenience of using a web browser as a client. Many of the Web applications around us are data-intensive, which indicates applications whose main purpose is presenting large amount of data to their users (10).

By using the information published on the Web, people can create new Web applications easier and more efficient. The technology that integrates information or

functionality from two or more existing Web sources(Web services or Web applications) to create a new Web page or application, which is called mashup technology, has been improved quickly in recent years. Integrating the published Web services through end-user programming is a natural and flexible method to build a new Web application based on the information from the Internet. However, this kind of integration has a limitation of the information resource, because that many Web sites publish information in Web application format instead of Web services. To use Web applications as the Web sources of mashup application, we need to extract information from the Web applications first. Partial information extraction technology is widely used to address this problem.

Nowadays more and more Web sites generate Web pages containing client-side scripts such as JavaScript and Flash instead of ordinary static HTML pages. These scripts create dynamic HTML pages and provide modern interfaces to Web applications such as AJAX applications. Unfortunately, for partial information extraction of Web pages, existing methods cannot extract dynamic Web contents. Here we use the term static Web content to represent the content shown on the Web page that can be found directly in the HTML source file, for example text. And we use the term dynamic Web content to represent the content that is created by client-side scripts dynamically, which is shown on the Web page but cannot be found directly in the HTML source file. In this thesis, we propose an approach that can extract both the dynamic Web contents and static Web contents from Web applications.

Furthermore, the widely used mashup technology has become very close to the users without programming experience. Unfortunately, most of the current Web sources are not aimed towards the users without programming experience. In this thesis, we propose a description based approach that allows the users to build mashup applications easily and efficiently without programming. We design an XML based description language file which can be generated easily to record the necessary information for generating the mashup application automatically. In addition, the cooperation of the Web applications can be implemented without programming.

To use Web services as the Web sources of mashup application, there are mainly two kinds of methods. First kind is by end-user programming. Which is also the most natural method. Another kind of method is to use the mashup tools that can make the integration of Web service with no or little programming. Currently, these mashup

tools have the limitation that the only the Web service in the pre-defined library are usable. Extend the partial extraction method and the description language, we propose an approach that can use integrate the information extracted from the result of REST services without programming automatically.

Considering the flexibility of the integration towards the programmers, we design a Java based plug-in library that provides several kinds of partial information extraction functions. With this library, users can build mashup applications easier through end-user programming. Because that users can focus on the mashup scenario when they build mashup applications instead of how to get the information from Web sources.

The goal of this thesis is to present a description based approach for the flexible integration of Web applications and Web services. Which can help the users to focus on the designing of scenario when they build mashup applications.

The main contributions of this thesis are:

- The thesis presents an approach that can extract both dynamic and static Web contents from general Web application.
- Based on the above partial information extraction method, an approach that allows the users without programming experience to build mashup applications with any part(s) of general Web applications and transfer information between the mashup components created from general Web applications is proposed.
- By extending the method towards the process of Web application to REST Web services, users can build mashup applications with general Web applications and REST Web services without programming.
- This thesis also provides a Java based plug-in library that includes several partial information extraction functions which can help building mashup applications through end-user programming.

1.2 Organization of the Thesis

The rest of this thesis is organized as follows. Chapter 2 explains the background technologies and the related work of the thesis. Chapter 4 demonstrates the details about the partial information extraction method that can extract both static contents and

dynamic contents from general Web applications. Based on the location information of the specified parts, we can extract the parts automatically towards the Web pages that have the same or similar layout in a Web site.

Chapter 5 and Chapter 6 show the proposed approaches for the flexible integration of Web applications and Web services without programming based on the partial information extraction in order. Chapter 5 shows the approach for building mashup applications by using the parts of existing Web applications based on the information extraction and transfer. Chapter 6 shows the approach that can create the mashup applications easily with the existing general Web applications and REST Web services based on the partial information extraction method.

Chapter 7 discusses the approach for building mashup applications through end-user programming. A Java-based plug-in library is developed to support the integration through end-user programming. Chapter 8 provides the evaluation of our methods. Chapter 9 summarizes our contributions and concludes the thesis.

Chapter 2

Background and Related Work

2.1 Background

The word mashup comes from the music world which means a song or composition created by blending two or more pre-recorded songs, usually by overlaying the vocal track of one song seamlessly over the instrumental track of another (49). In Web development, a mashup is a Web page or application that uses and combines data, presentation or functionality from two or more sources to create new services (65).

In recent years, mashup applications have brought new creativity and functionality to Web applications. According to the ProgrammableWeb (52), there are more than 7,086 mashup applications by 10th, Jun. 2013 and there are about three new mashup applications generated every day on average.

The mashup applications are roughly categorized in following two groups:

- Presentation Mashups are used to facilitate the creation of information portals from different resources for presentation purpose. The well-known Web application iGoogle (37) is an example of consumer mashup application, which is a personal Web portal that can add Web feeds and Google Gadgets (such as news, calendar and so on).
- Data Mashups are used to integrate data and services from different resources, which aim to facilitate the data access and cross-referencing between resources. Many geo-based mashup applications are created by integrating the information from different resources into Web-based maps, such as Google Maps (25) and Yahoo Maps (73).

In this thesis, we are focus on the solution of personal-oriented Data Mashups. The Web sources that are used to build mashup applications mainly include Web applications and Web services. The traditional way of building mashup applications is using Web services by writing a script or a program to invoke those Web services. There are many kinds of Web services supplied in the Web. Figure 2.1 shows the top APIs of mashup applications and their percentage of utilization in community-created mashup applications given by the ProgrammableWeb at 12th, May 2011.

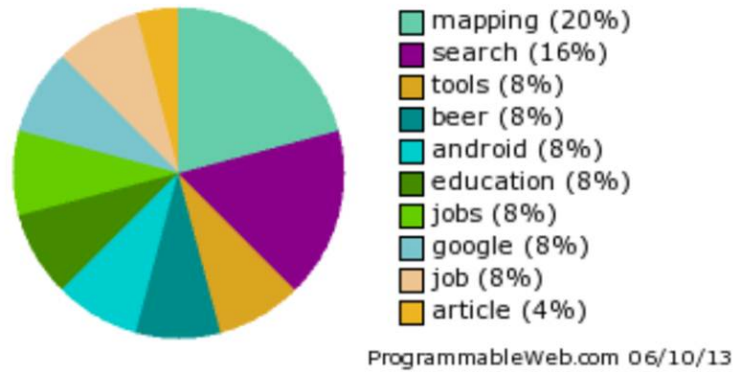


Figure 2.1: Top APIs for Mashups - Data from Programmableweb

Unfortunately, many existing Web sites publish information in Web application format instead of Web services format. For example, many news sites, such as the CNN news (15) and reuters news (53), have the functionality that give the news list according the given keyword. However, this functionality do not be supplied as a Web service. Thanks to the partial information extraction technology, we can extract information from general Web applications and use it to build mashup applications.

However, with the development of the Web technology, more and more Web sites generate Web pages containing client-side scripts such as JavaScript and Flash instead of ordinary static HTML pages. These scripts create dynamic HTML pages and provide modern interfaces to Web applications such as AJAX applications. We divide the contents on Web pages into two types, static Web contents and dynamic Web contents. As we mentioned, the dynamic Web contents are created by client-side scripts dynamically, which is shown on the Web page but cannot be found directly in the HTML source file. Thus it is difficult to extract and reuse the dynamic Web contents as mashup components. The existing partial information extraction methods can only

extract static web contents. To address this problem, we need an approach that can extract any parts from a Web page of general Web applications for reuse.

Furthermore, the widely used mashup technology has become very close to the users without programming experience. But, it is not easy to build mashup applications with existing Web applications, especially for the users without programming experience. Because the fact that most of the current Web sources are not aimed towards the users without programming experience, many researches towards personal-use-oriented mashup for users without programming experience have occurred.

The existing personal-use-oriented mashup tools for users without programming experience cannot address the following problems.

- They cannot use dynamic Web content of Web pages as mashup components.
- It is not easy to use general Web applications as Web sources of mashup applications without programming. In addition, the existing researches do not support the cooperation between the Web application which can be reflected in the mashup components generated from Web applications.
- Most of the existing researches have the limitation when they use the Web services as the Web source by using a predefined library.

2.2 Related Work

Handling of entire Web pages is not convenient for integration of necessary applications or collection of desired information. If we can automatically extract parts of Web pages, then we can realize such integration and collection easily. To address this problem, currently there are many existing methods for partial information extraction. Unfortunately these existing methods can only extract static contents.

WIKE (32) can extract information from Web sites conveniently to generate Web services or integrate it into an XML file, but the type of information is limited to static Web contents. It cannot deal with the dynamic Web contents created by client-side scripts. In addition, it is towards the automatic information integration of one Web application instead of multiple Web applications.

Dapp Factory (16) can wrap information extracted from Web pages or RSS (54) into other data format, such as Dapp XML, RSS feed, Google Gadget and so on. But

it cannot extract dynamic Web content from Web pages, for example the clock part of Localtimes.info (46).

Crunch (29) is an HTML tag filter to remove the extraneous elements of a Web page. It is implemented as a proxy to filter Web pages before they are loaded into end-users' browsers. PSO (59) is an approach to extract the parts of Web pages. ANDES(50) is an XML-based methodology to use the manually created XSLT processors to realize the data extraction. However, these three methods above cannot treat dynamic Web contents and need end-users to find out the paths from the HTML document of Web page manually.

At the same time, there are many methods for combining data from more than one Web sources into a single and integrated Web page automatically or semi-automatically. We study these methods from the following two facts. What kind of Web sources can be used and whether it has any limitation on the usage of Web sources. The process of integration is automatically or need programming.

Whip (44) provide an interface to build mashup applications with the predefined widgets. Users with programming skills can build new widgets. The data sources it can use mainly focus on the Web services. The presentation level of mashup applications can be accomplished automatically.

Internet Scrapbook (43) is a system which allows end-users to create a personal Web page by clipping parts of several Web pages and assembling them. The Web sources are limited to Web applications. The integration process is automatic. However, the extracted information is a part of HTML document and end-users can not change the layout of the extracted parts. In addition, it cannot reuse the dynamic contents.

C3W (23) can reuse text and functional parts of Web pages by clipping, connecting and cloning to build mashup applications. The Web sources it can use are limited to Web applications. The integration process is automatic. However C3W is inaccessible to many end-users because of a special browser to clip Web contents from Web pages, and a special container named DerivationPad for the clipped Web contents.

Mixup (77, 78) can quickly build complex user interfaces by using the Web service APIs available in the system. The Web sources it can use are limited to the Web services that have registered in it. The integration process is semi-automatic.

Yahoo Pipes (75) offers a platform which provides a number of user-configurable modules to build mashup applications. Users can use Web feeds, Web services and

retrieved text from Web pages by configuring modules to build mashup applications. Users can add certain kinds of Web sources, but not completely free. The integration process is automatic. However it do not deal with the dynamic contents and it is not easy to retrieve text from Web pages in Yahoo Pipes.

"Data Mashup Platform" (18) can use four kinds of data resources (database, rest-ful sensor resource, web service and local file), and the predefined mashup logics to generate "mashup services". Users can configure the resources and achieve the mashup services through a data mashup editor. The Web sources it can use do not include the information from Web application. This approach does not provide any user interface for designing mashup applications.

MashMaker (20) use the widgets created by third-parties to build mashup applications. It works as a plug-in of Web browser. The resources are limited to the widgets library and the widgets can be generated from Web applications and Web services. Although they allow the users to create their own widgets, many users do not have the ability to create the widgets. Thus we should say the integration process is semi-automatic.

mashArt (48)allows users to compose mashup applications using UI components. The Web sources can be Web applications and Web services. The integration process is automatic. However, it has a premise that component developers (IT specialists) need to componentize UI components from the Web applications and publish them on the Web.

Liu et al. (45) consider that a mashup application consists of three elements, UI Component, Service Component and Action Component. This approach allows users to create new applications by simple actions. The integration process is automatic. However, it cannot ensure that every service works well. In addition, it just supports simple actions.

iMashup (34) is a web-delivered service composition environment based on the browser middle-ware. It can build mashup applications with the predefined Web services. The integration process is automatic.

ServFace (56) aims at creating a model-driven service engineering methodology for an integrated development process for service-based applications. It focuses on the service-based mashup and is not a published tool. The Web services will be generated to ServFace Annotations before they are used to do integration.

Chapter 3

Partial Information Extraction

With the development and popularity of the World Wide Web, more and more information has become available on the Web. The Web can be considered as an infinite source of information. However, the heterogeneity and the lack of structure of Web information sources made the access to this huge collection of information be limited to browsing and searching at a time. Web information extraction is a kind of helpful method for filtering information from large volumes of Web information to address this problem. Information extraction extracts data from a collection of relevant information to a particular extraction task. For instance, the meta-search engines extract the search results from multiple search engines. It could produce structured data ready for post-processing, which is crucial to many applications of Web mining, searching tools and information integration.

In fact, the idea of reducing the information in a document to a tabular structure is not new. Its feasibility for sub-language texts was suggested in the 1950's, and an early implementation for medical texts was done at New York University (28). Information extraction has been the focus of the Message Understanding Conference (MUC). The proliferation of the Web, however, intensified the need for developing information extraction systems that help people to cope with the enormous amount of data that is available online (66). Recent years, many approaches about information extraction have been proposed aiming to retrieve data from Web pages.

According to the different purpose, the effective approaches might not be the same. The goal of our partial information extraction method is to support the integration that uses general Web applications. As a result, the data source of our information

extraction task is the accessible Web pages. The extraction target could be any part on these Web pages. In this chapter, what kind of information extraction technology is fitting to our purpose will be discussed. It is important to note that the low cost, flexibility in development and easy adaptation to various domains are necessary to any kinds of information extraction task.

3.1 Information Extraction Technologies

As Grishman said that a mature information extraction technology would allow us to rapidly create extraction systems for new tasks whose performance was on a par with human performance. In information extraction systems, the user interaction has evolved from writing extraction rules to labeling target extraction data. Extraction rules are the main part of the information extraction technology, which includes the rule of labeling target extraction data. Some of information extraction systems use path-expressions of the HTML parse tree path (e.g. `html.head.title`, and `html->table[0]`) as the features in extraction rules; some use syntactic or semantic constraints, such as POS-tags and Word-Net semantic class; while others use delimiter-based constraints, such as HTML tags or literal words, in the extraction rules (12).

Usually, in order to search for target information, Web applications provide request-submit functionalities for users. For example, search engine applications provide a text input field on the Web page for users to input the searching keywords. Most of the Web applications use the same or similar layout-generator file to show the resulting pages of the same request-submit functionality. This kind of Web pages are so called template-based Web pages, are the data source of our information extraction task. Since the data to be extracted are often co-located in the same path of the DOM tree, it is a good selection to use DOM tree paths to denote a specific piece of information in a Web page.

Some approaches use HTML DOM tree and pattern expression as the features in information extraction rules, for example WIKI system. However, for our purpose, the methods using pattern expression have following two demerits.

- For the Web pages that are generated dynamically according to the data and layout definition, the WIKI system is not applicable. Although the pattern

mining method(13) can be used in this case, the accuracy is not high enough and the adaptation to domains except search engine type is unverified.

- It takes time to generate pattern information and search target content using pattern detection method. The time that uses WIKE system to generate the pattern information of BBC Country Profiles Web pages is 16 minutes ¹. Because that it needs to analyze all Web pages before generating the pattern information.

For integration, we need a fast method with good extraction accuracy and adaptation to various domains.

3.2 Feature-Expression of Our Extraction Method

The nested structure of HTML document tags in a Web page naturally forms a tree structure. Each path represents a root node of subtree and each subtree represents a part of the Web page. The Xpath (70) of HTML document is widely used to describe the HTML nodes. The id attribute of tag in HTML document usually is thought as an unique value. Hence, the path format that begin with the last ancestor node which has id attribute is widely used. This kind of path is short, but the experiments we have done shows that the id attribute of tag in HTML document is not always unique, for example, the Web pages in Infoplease (38) Web site.

Through the experiments, we found that although the class attribute of tags in HTML documents usually is not unique, the class attribute is widely used. According to the experiments that we have done, to ensure the extracting accuracy, we use HTML document's Xpath and the id, name, class attributes of tag in the following format to record the location information of Web contents.

/ HTML/ BODY/ node1[index1] (id1, name1, class1)/ node2[index2] (id2, name2, class2)/ .../ node(N-1)[index(N-1)] (id(N-1), name(N-1), class(N-1))/ nodeN[index] (idN, nameN, classN)

Here, *nodeN* is the tag name of the *N-th* node, *[indexN]* is the order of the *N-th* node among the sibling nodes that have the same tag name as *nodeN*, *idN* is the ID attribute's value of the *N-th* node, *nameN* is the NAME attribute's value of the *N-th*

¹The experimental computer is under Microsoft Windows XP Professional SP3, with Genuine Intel(R) CPU U1400 @ 1.20GHz, 1.99GB RAM.

node, $classN$ is the CLASS attribute's value of the N -th node, and the $node(N-1)$ is the parent node of $nodeN$.

The experimental data shown in 8.2 shows that the extraction accuracy is more than 90% with one path expression. We use multiple path expressions to improve the extraction accuracy. The extraction time it takes, for example extracting five segments (country name, location map, facts, photo of leader and name of leader) from a page of BBC Country Profiles with one path expression, is 3 second, while WIKE system takes about 6 second ¹. We use the same computer that we survey the time of generating the pattern information of BBC Country Profiles. Figure 3.1 shows the process how we fix the location of a HTML node according to the given XPath based expression.

There are many kinds of information on Web pages such as a piece of text, a piece of map or a group of photos. We divided them into tow kinds, dynamic contents and static contents, according to their features as shown in Figure 3.2 and Figure 3.3. We explain the approach for extracting the static Web contents in the following Section and the approach for extracting the dynamic Web contents in the next Chapter.

3.3 Extraction of Static Web 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. Here we use the term *text* to represent the text string between two tags; *object* to represent the instance of the photo, video and other multimedia file; *link* to represent the reference in a hypertext document; *single* to represent a node without similar sibling nodes; *continuous* to represent a list of nodes with similar paths such as result list of a search engine.

¹Because that the WIKE system collects information of a Web site continuously, the time here we have gotten is the average time that the total time it takes divided the number of pages it dealt with.

3.3 Extraction of Static Web Contents

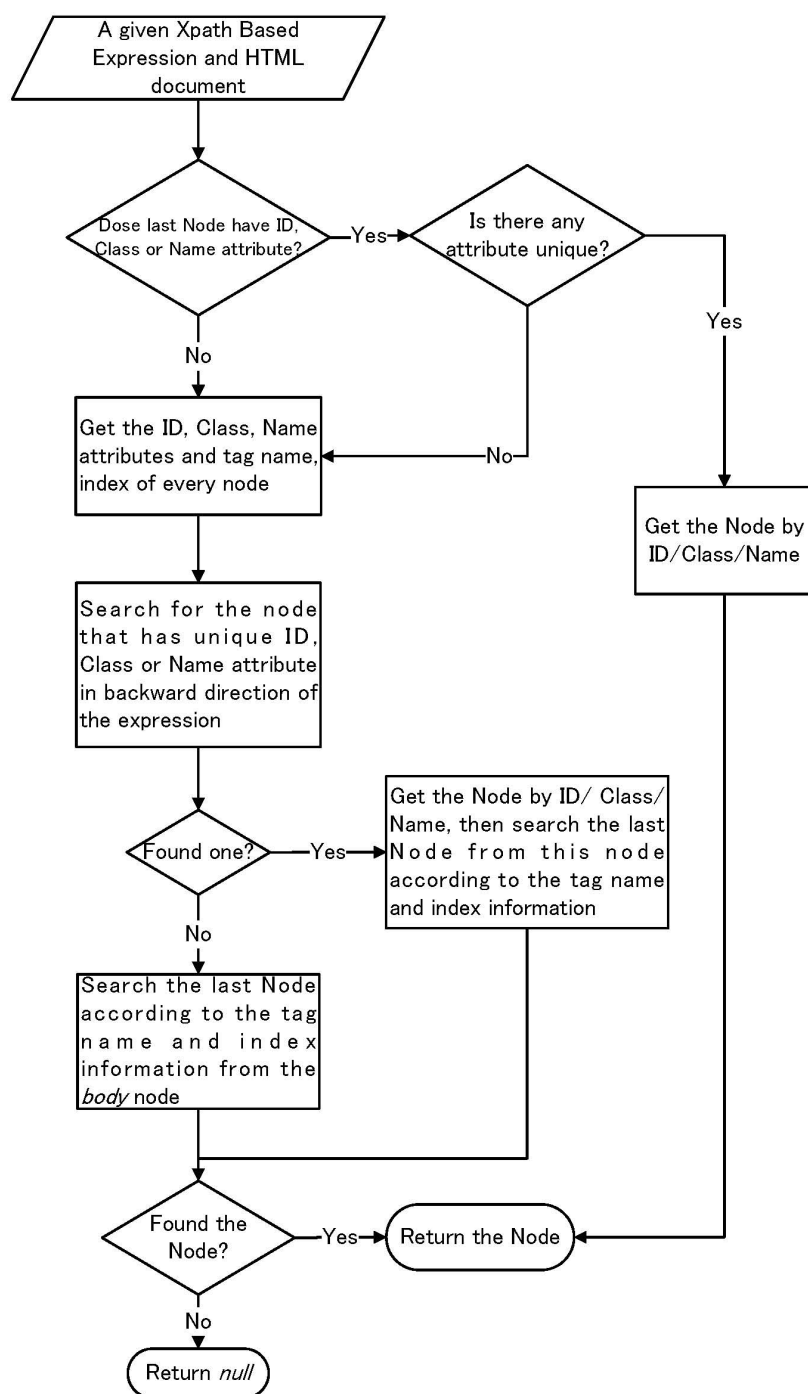


Figure 3.1: Process of Fixing the Location of a Web Content - Flow Chart

3.3 Extraction of Static Web Contents

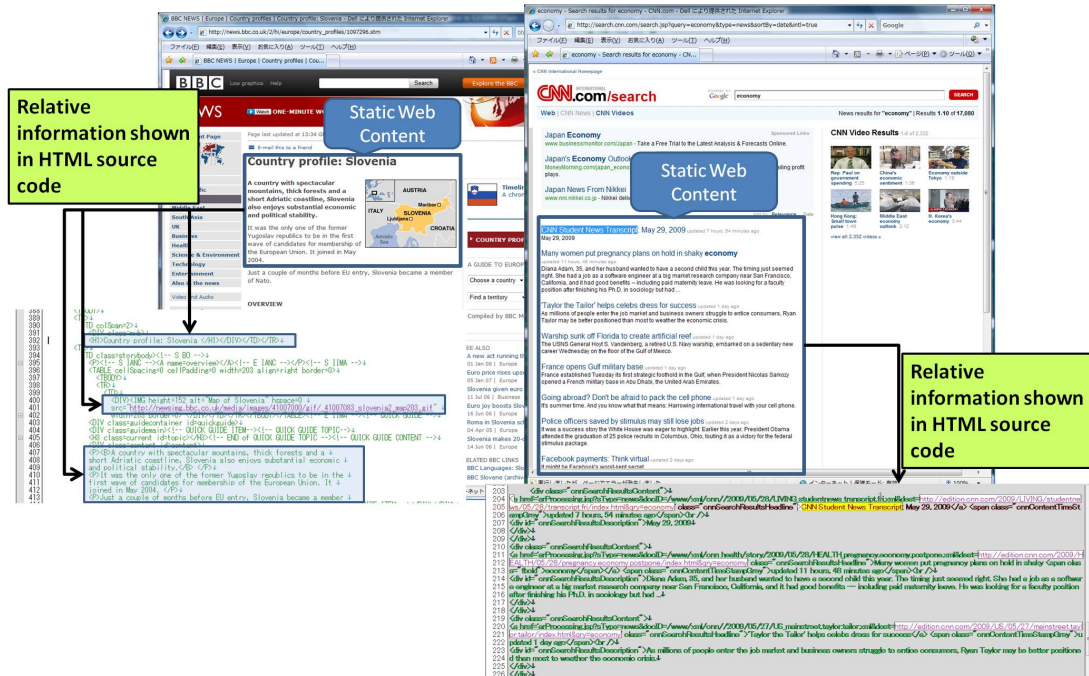


Figure 3.2: Static Web Content - Example

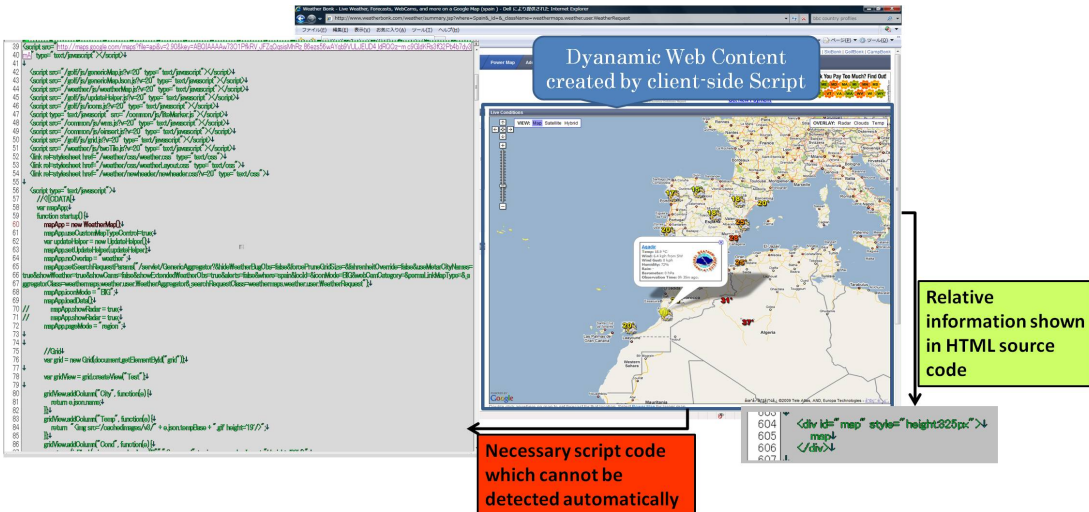


Figure 3.3: Dynamic Web Content - Example

Text Extraction

When extracting the text type of information, we extract text string (the node value) of corresponding node as shown in Figure 3.4.

3.3 Extraction of Static Web Contents



Figure 3.4: Text Extraction - Example

Object Extraction

When extracting the object type of information, we extract the attribute value of corresponding `<object>` node. For example, the extracted information of a photo is the value of "src" attribute as shown in Figure 3.5.

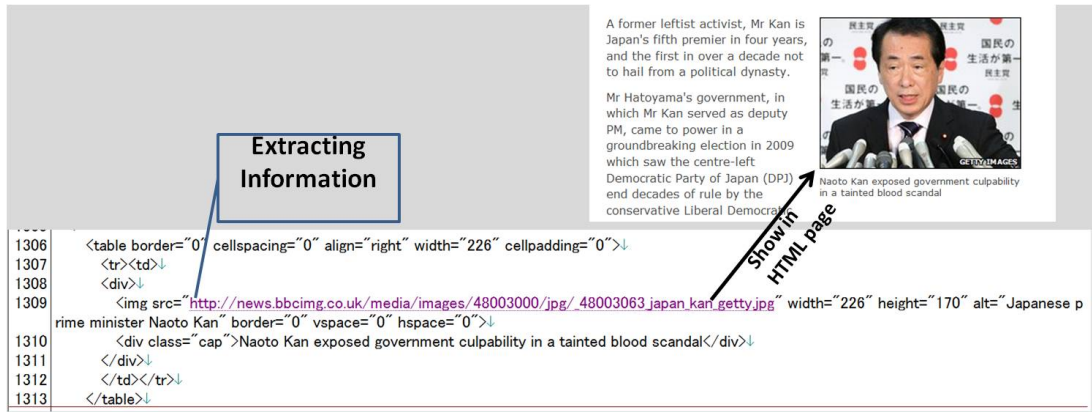


Figure 3.5: Object Extraction - Example

Link Extraction

When extracting the link type of information, we extract the embedded link value of corresponding `<a>` node as shown in Figure 3.6.

3.3 Extraction of Static Web Contents

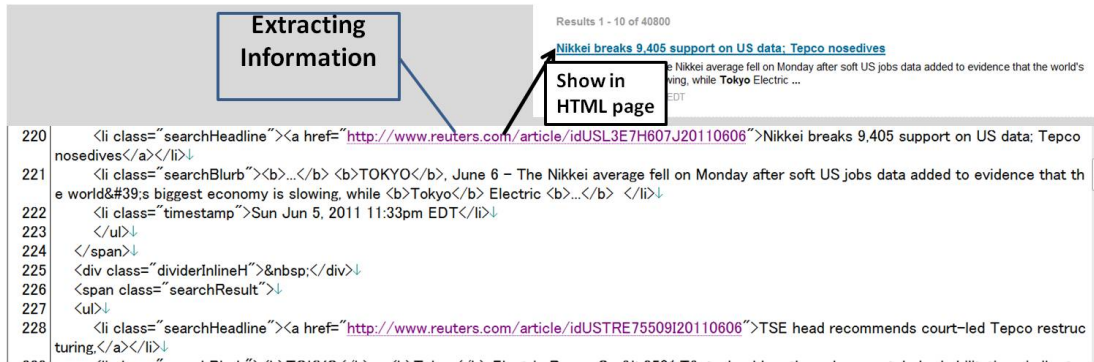


Figure 3.6: Link Extraction - Example

Continuous Type Extraction

In HTML tree structure, the data type of a part sometimes occurred continuously, as shown in Figure 3.7. In this case, we first find the node A according to the given path expression. Then, get the parent node P. Thirdly, get the P's children nodes that have the same node name with the node A. Finally, extract the information of all these nodes.

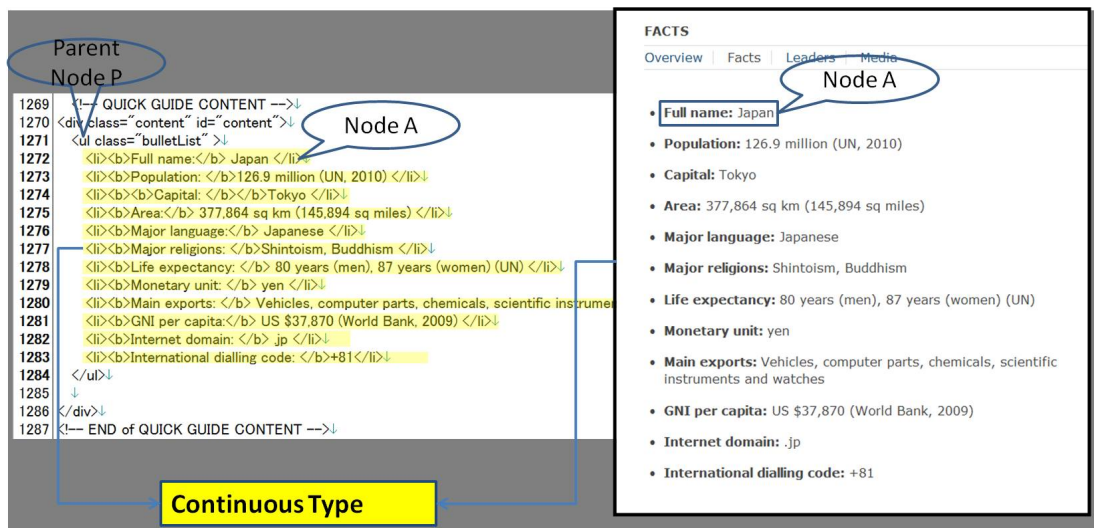


Figure 3.7: Continuous Type - Example

Chapter 4

Hide-and-Display Method

In this chapter, we propose a partial information extraction approach that is applicable to both dynamic and static Web contents. We call it hide-and-display method. We can use the approach that extracts the data from the HTML document according to the matching result and reorganize the data to implement the extraction of static Web contents. But for dynamic Web contents, if we only extract the data according to the matching result, it does not work. Because these contents are created by the scripts and do not work without the executing environment of scripts. Our partial information extraction approach is a path matching based method which can keep the original executing environment of scripts. Thus it is applicable to both dynamic and static Web contents.

4.1 Overview

The primary condition of our partial information extraction approach is that it is applicable to both static and dynamic Web contents. In addition, the functionalities of Web applications are useful and widely used now. By including certain kind of functionality (user defined, for example search functionality and so on) of the Web application, our partial information extraction method may look like a kind of Web service from users' point.

An overview of the hide-and-display method is visualized in Figure 4.1. The process of the method is as follows.

Step 1: Analyze the structure of the Web application and record the functionality that the user what to use through the application we supplied, named Path-Reader, which is shown in Figure 4.3. Then go to a random example Web page by using this functionality.

Step 2: Select the extracting contents on the above Web page with the same application.

Step 3: Extract the specified Web contents according to the given type of the Web contents by using path matching method to fix the location.

For dynamic Web contents, we prepare two kinds of methods to do the extraction. First method is extracting all the script tags, meta tags and the HTML tags of the specified Web contents. The second method is extracting the whole HTML page and then hiding the HTML tags except the specified Web contents. This is because the scripts may use DOM operation to control the dynamic parts of Web pages and sometimes access the elements outside the target parts such as the hidden values. Thus the first method may do not work in some cases.

For static Web contents, 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. 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.

The details of the extraction will be explained later in this Chapter.

Step 4: Output the extracting result. For the Web contents that is extracted as dynamic contents, the extracting result is in HTML format. For the Web contents that is extracted as static contents, the extracting result is in XML format, which can be returned in HTML format if a usable style file is given in advance.

4.2 Structural Analysis

Usually, in order to search for target information, Web applications provide request-submit functionalities for users. For example, search engine applications provide a text input field on the Web page for users to input the searching keywords. Most of the

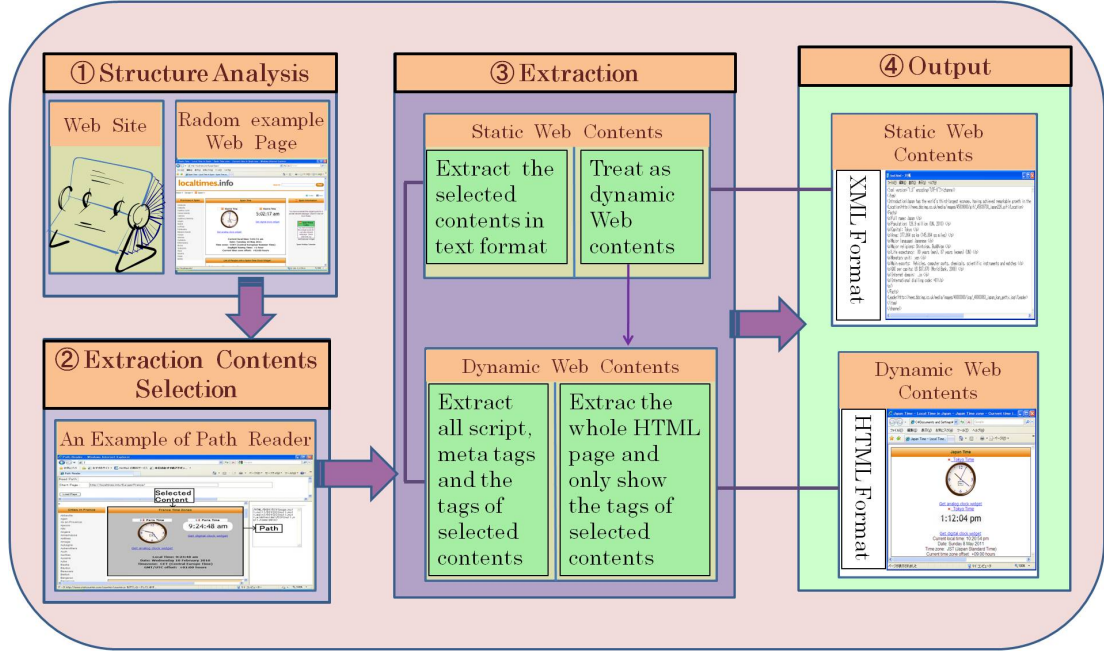


Figure 4.1: The Process of Hide-and-Display Method - Overview

Web applications use the same or similar layout file to generate the resulting pages of the same request-submit functionality. Based on this fact, we include certain kind of functionality (user defined) of the Web application in the hide-and-display method.

Three basic types of methods are widely used as shown in Figure 4.2. One is entering a keyword into the input field by keyboard and clicking the submit button by mouse to get the target Web page. We call it *InputBox* type. Another one is clicking an option in drop-down list in a Web page by mouse to view the target Web page. We call it *OptionList* type. The third one is clicking a link in link list in a Web page by mouse to go to the target Web page. We call it *LinkList* type.

There may be more than one request-submit functionality on the same Web page and they usually are put into different containers (for example form tags). So we need to record the the type of functionality and the container information here, including the URL of the Web page that the request-submit functionality can works. Users can select the type of functionality from { *InputBox*, *OptionList*, *LinkList* and *None (do not include any functionality)* }. The container information can be gotten by clicking the area in the application we supplied as the example shown in Figure 4.3. The container information is recorded as an XPath-like expression. The format of this expression will

4.3 Selection of the Extracting Web Contents

be explained in Section 4.3 in detail.

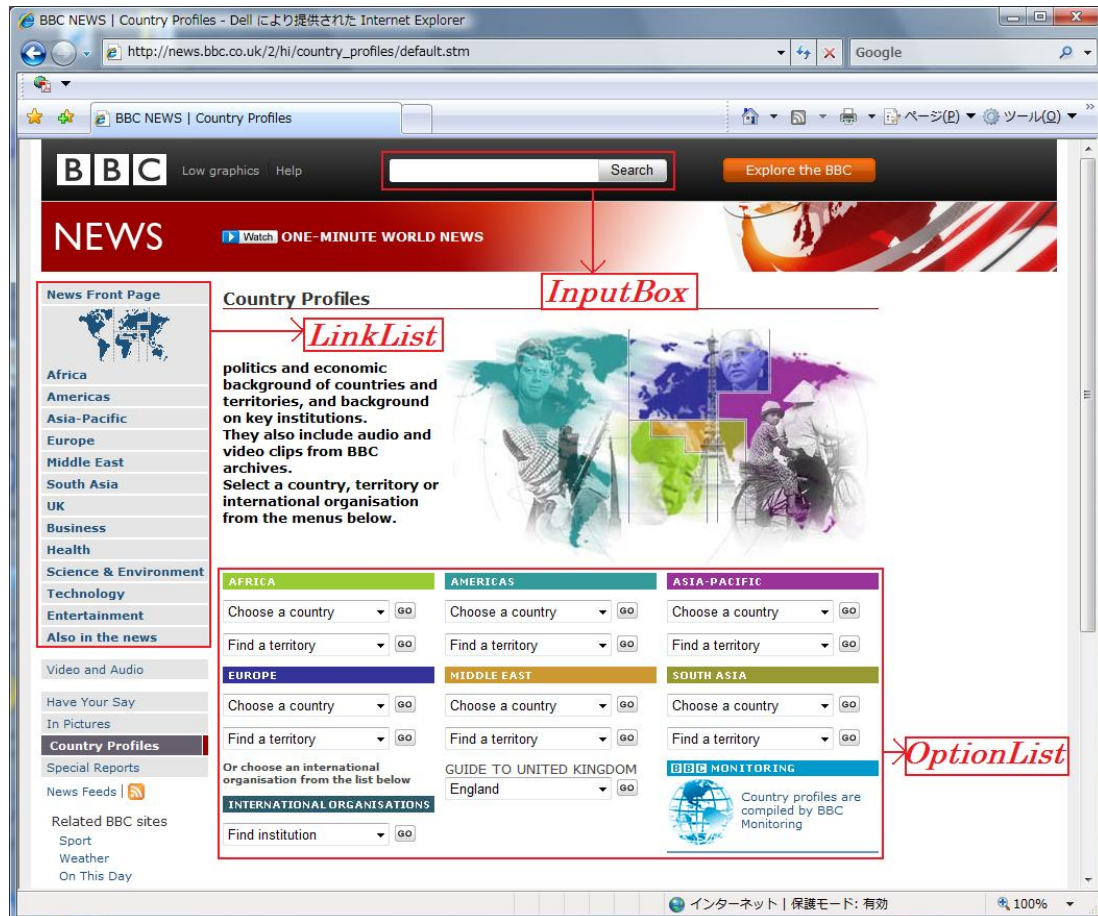


Figure 4.2: Examples of Request-Submit Functionalites - Three Basic Types

4.3 Selection of the Extracting Web Contents

We developed an application, Path Reader, to analyze Web pages and show the location information of Web contents easily, as shown in Figure 4.3. We use path matching method to fix the location of Web contents. Thus, the location information of Web contents is shown and recorded as an Xpath-like expression.

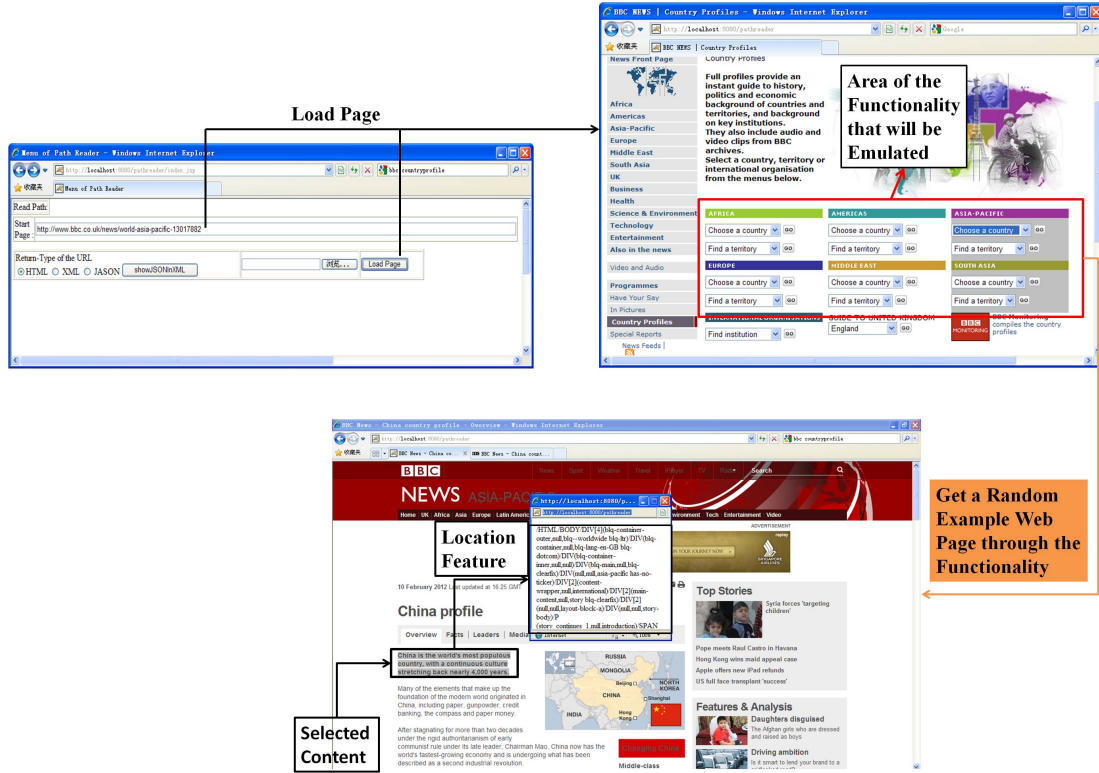


Figure 4.3: Path Reader - An Example

4.4 Information Extraction

If the user specified any functionality in the step one, we need to search the target Web page by using the functionality before information extraction. In this case, the searching keyword using in the functionality is necessary in advance. With the URL, functionality type and location information gotten in the step one, according to the given keyword we search the target Web page by emulating the submittal.

To submit a request, there are POST method and GET method, also encrypted codes or randomly generated codes are used by some Web sites. In order to get the target Web pages from all kinds of Web sites, we use HtmlUnit (33) to emulate the submitting operation. According the type of the functionality, the emulating process is shown as following.

1. In the case of an InputBox, after finding the input box node according to the path string of input area, setting the value of input box, then we emulate the

click function of the search button to get the target Web page, as described in Listing 4.1.

2. In the case of a LinkList, we compare the text of all children links of the specified functionality location node to look for the keyword, then we emulate the click function of the link to get the target Web page.
3. In the case of an OptionList, we compare the text of all select options of the specified functionality location node to find the keyword, then we emulate the setting function of the select option to get the target Web page.

Listing 4.1: Example Codes of Emulating InputBox-type-submission

```

1 // initial a web client object
2 WebClient objWebClient = new WebClient();
3 objWebClient.setRefreshHandler(new ThreadedRefreshHandler()
4     );
5 // enable JavaScript
6 objWebClient.setJavaScriptEnabled(true);
7 // enable cookies
8 objWebClient.setCookiesEnabled(true);
9 objWebClient.setThrowExceptionOnScriptError(false);
10 objWebClient.setPrintContentOnFailingStatusCode(false);
11 // load start page
12 HtmlPage StartPage = (HtmlPage) objWebClient.getPage(
13     StartPageURL);
14 // get form
15 HtmlForm objForm = getSubmitForm(StartPage);
16 // get text input box
17 HtmlTextInput objTextInput =
18     (HtmlTextInput)getTextInputSubmit(objForm).
19     elementAt(0);
20 // get submit button
21 HtmlInput objSubmitInput =
22     (HtmlInput)getTextInputSubmit(objForm).
23     elementAt(0);
24 // input the value as keyword

```

```

21 objTextInput.setValueAttribute(Keyword);
22 // submit request to get the returned target result webpage
23 HtmlPage ResultPage = (HtmlPage) objSubmitInput.click();

```

After obtaining the Web page, we extract the specified Web contents according the type and location information gotten in the step two.

4.4.1 Dynamic Web Contents

Dynamic Web contents are generated by the client side script codes. Thus, to extract a dynamic content, first we think about locating the relative script codes. The following three kinds of situations are frequently occurred.

- Script codes or the codes which can illustrate the relationship are nested in the relative HTML tag. Figure 4.4 shows two examples of this kind of situation. In the case shown in A, the content can work correctly if we extract the <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 case, the automatic detection and extraction are difficult.
- Script codes locate at a part which has no hint to illustrate the relationship with the relative HTML tag. Figure 3.3 in Chapter 3 shows an example of this situation. For this kind of situation, the automatic detection and extraction are difficult.
- Script codes are located at a script file on the server of the Web site. This situation usually occurs with situation one or two. An example is shown in Figure 4.5, in which part of the script codes are nested in the relative HTML tag while the other part are in a script file on the server. In this example, the location of the script file is given in absolute path, thus the content can work correctly with the extracted HTML tag. If the location is given in relative path, it cannot work correctly.

```

685 -->↓
686 ↓
687 <IFRAME WIDTH="160" HEIGHT="600" SCROLLING="No" FRAMEBORDER="0" MARGINHEIGHT="0" MARGINWIDTH="0" SRC="http://ad.dc2.adte
ch.de/adiframe/3.0/1099/2752636/0/0/ADTECH.target=blank;key=China">↓
688 <script language="javascript" src="http://ad.dc2.adtech.de/addyn/3.0/1099/2752636/0/0/ADTECH.loc=700;target=blank;key=China"></script>↓
689 </IFRAME>↓
690 ↓
691 <!-- End of IFRAME Tag -->↓

```

A: Script codes are embedded in the HTML tag

```

1007 <div class="countryTitle" onclick="SwitchMenu('sub1')">↓
1008  ↓
1009 <font size="2">Asia</a> has 6876 people with a Clock widget</font>↓
1010 </div>↓
-----
30 function SwitchMenu(obj){↓
31 if(document.getElementById){↓
32   var el = document.getElementById(obj);↓
33   var ar = document.getElementById("masterdiv").getElementsByName("span");
34   if(el.style.display != "block"){↓
35     for (var i=0; i<ar.length; i++){↓
36       if (ar[i].className=="countrySubmenu") ↓
37         ar[i].style.display = "none";↓
38     }↓
39     el.style.display = "block";↓
40   }↓
41   else↓
42   {↓
43     el.style.display = "none";↓
44   }↓
45 }↓
46 }↓
47 </script>↓

```

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

Figure 4.4: Script codes in HTML Tag - Situation 1

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

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

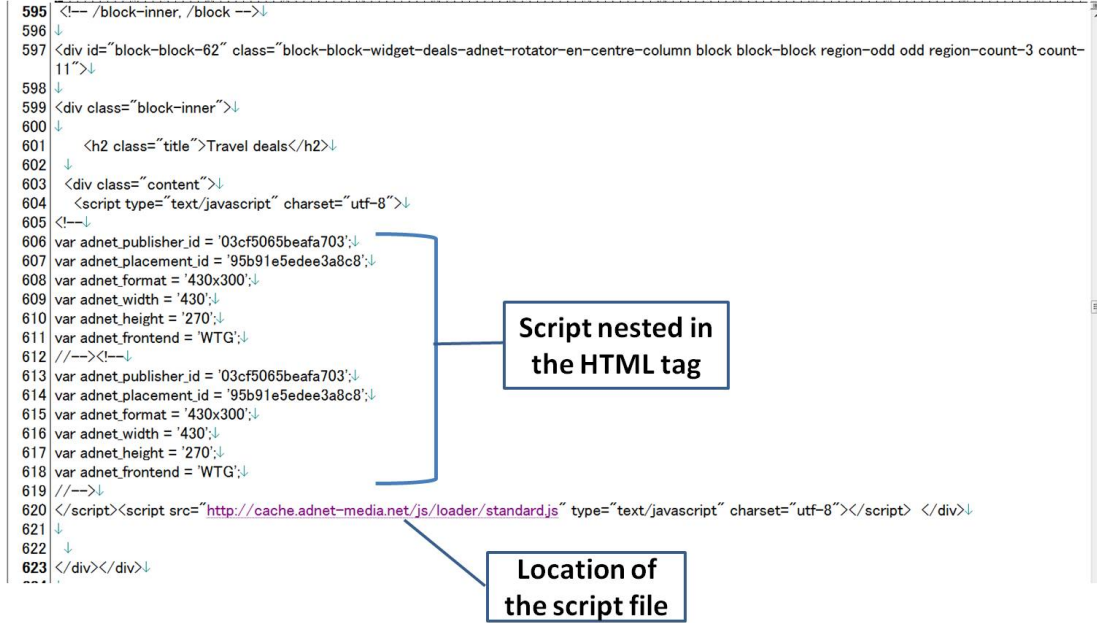


Figure 4.5: Script codes in Script File on Web Server - Situation 3

4.4.1.1 Compression Method

The Compression Method is implemented as following steps.

- We extract the whole HTML document of the target Web page.
- Analyze the obtained target Web page and mark the extracting contents and all of its child-nodes.
- Keep the *html*, *meta*, *script*, *body* tags and the marked tags and delete the other tags in the HTML document.

Finishing the above steps, we can get a compressed HTML file from the original source file of the target Web page. Consider of the fact that many Web sites use relative path to give the path of the meta files and other 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>`. 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.

If the DOM operation or hidden values of the HTML document are used, the dynamic content that is extracted using this approach can not work correctly. For example, the script codes in Figure 4.4 B use the following DOM operations.

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

4.4.1.2 Visibility Control Method

The process of Visibility Control Method is in following four steps.

- We extract the whole HTML document of the target Web page.
- We hide all HTML nodes in the extracted HTML document of the target Web page by setting the display property of style attribute to "none".
- We show the Web contents that are specified as extracting contents. Listing 4.2 is the code for showing the target Web contents. In the *ShowData* function we delete the display property of the style attribute of the target Web content node and all its ancestor nodes. Then we delete the display property of style attribute of all its children nodes in the *ShowChildren* function.

Listing 4.2: Show Target Web Contents

```

1 showAccordingPath(Html HTMLNode, String[]
   targetContents){
2     BodyTag body = GetBodyTag(HTMLNode);
3     for (int i = 0; i < targetContents.length ; i ++){
4         String strPath = targetContents[i];
5         Node targetNode = getNodeByPath(body, strPath);
6         ShowData(targetNode);
7         ShowChildren(targetNode.getChildren());
8     }
9 }
```

- We 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.

In order to accelerate the loading procedure, we modify the HTML source. For example, we set the "src" attribute of `<img>` to null to avoid downloading the image file.

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

The reason that we need Visibility Control Method even we have the Compression Method is that some Web content is used in Script, such as hidden values, when the Web content is not within the target Web contents. Using this Visibility Control Method, we can get an HTML document that only shows the specified Web contents.

4.4.2 Static Web Contents

We supply two kinds of extracting methods to extract static Web contents. One method is using traditional method to extract the target part in text format excluding the tags of the HTML document. Another method is to treat them as dynamic Web contents. For the first method, we support six kinds of data types. They are single text, continuous text, single object, continuous object, single link and continuous link.

For the text type of contents, we extract the node value of corresponding node. For the object type of contents, we extract the value of the attribute *src* of the corresponding node. For the link type of contents, we extract the embedded link value of the corresponding node. For the continuous type of contents, we extract the related value of the corresponding node and the sibling nodes.

4.5 Extracted Result

For the Web contents that is extracted as dynamic contents, the extracting result is in HTML format. For the Web contents that is extracted as static contents, the extracting result is in XML format, which can be returned in HTML format if a usable style file is given in advance. An example of the extracting result is shown in Figure 4.6.

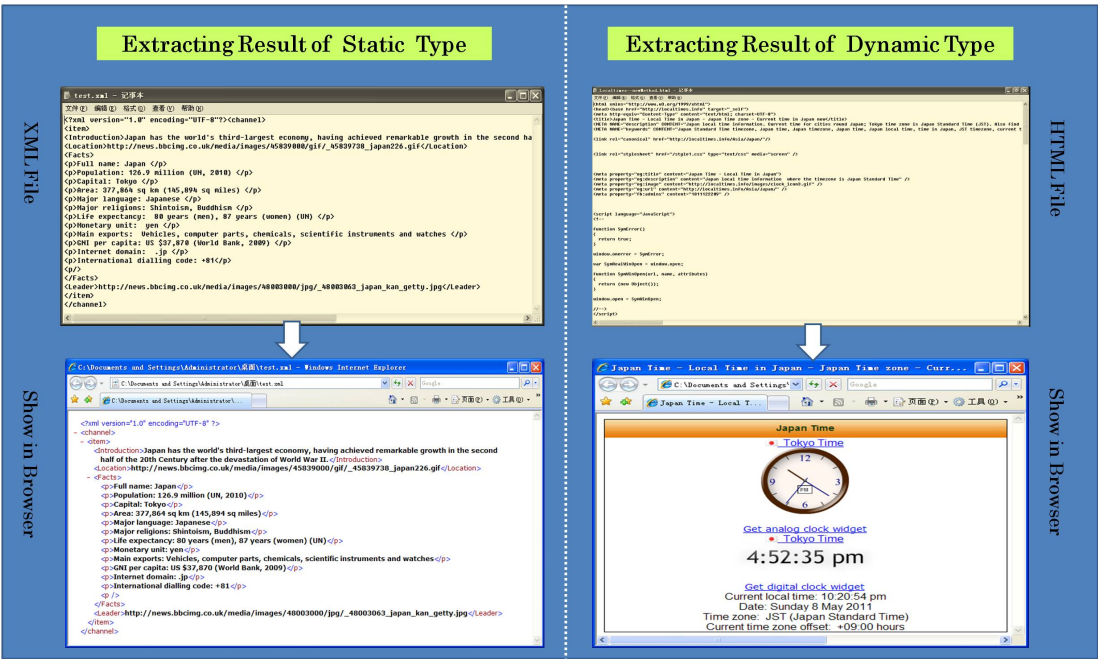


Figure 4.6: Extracting Result - An Example

Chapter 5

Description Based Mashup of Web Applications

With the output of the hide-and-display method, users can build mashup applications with any parts of general Web applications by programming. However, for the users without programming experience, it is still difficult to use the hide-and-display method. Research on end-user programming has found that programming is difficult for novices for a number of reasons (42). They must: enter syntactically correct code, find appropriate operations, and understand any programming errors that may arise (41). A variety of solutions to these problems have been proposed. We can find a review in (40). In this chapter, we present a description based approach for building mashup applications with general Web applications without programming. Based on the hide-and-display method, we design a display-template for information collection which allows users to build information collection mashup applications without programming. In addition, this approach supports the cooperation of Web applications by transfer information between them.

5.1 Overview of the Approach

The overview of Description Based Mashup of Web Applications (DBMWA) method is shown in Figure 7.1. To address the problem that users can use the hide-and-display method without programming, we design a Mashup Application Template. It is a HTML file that generated by HTML Java-Script codes. The purpose of the Mashup

Application Template is to define the display layout of the mashup application, call the hide-and-display method, receive and show the extracting result.

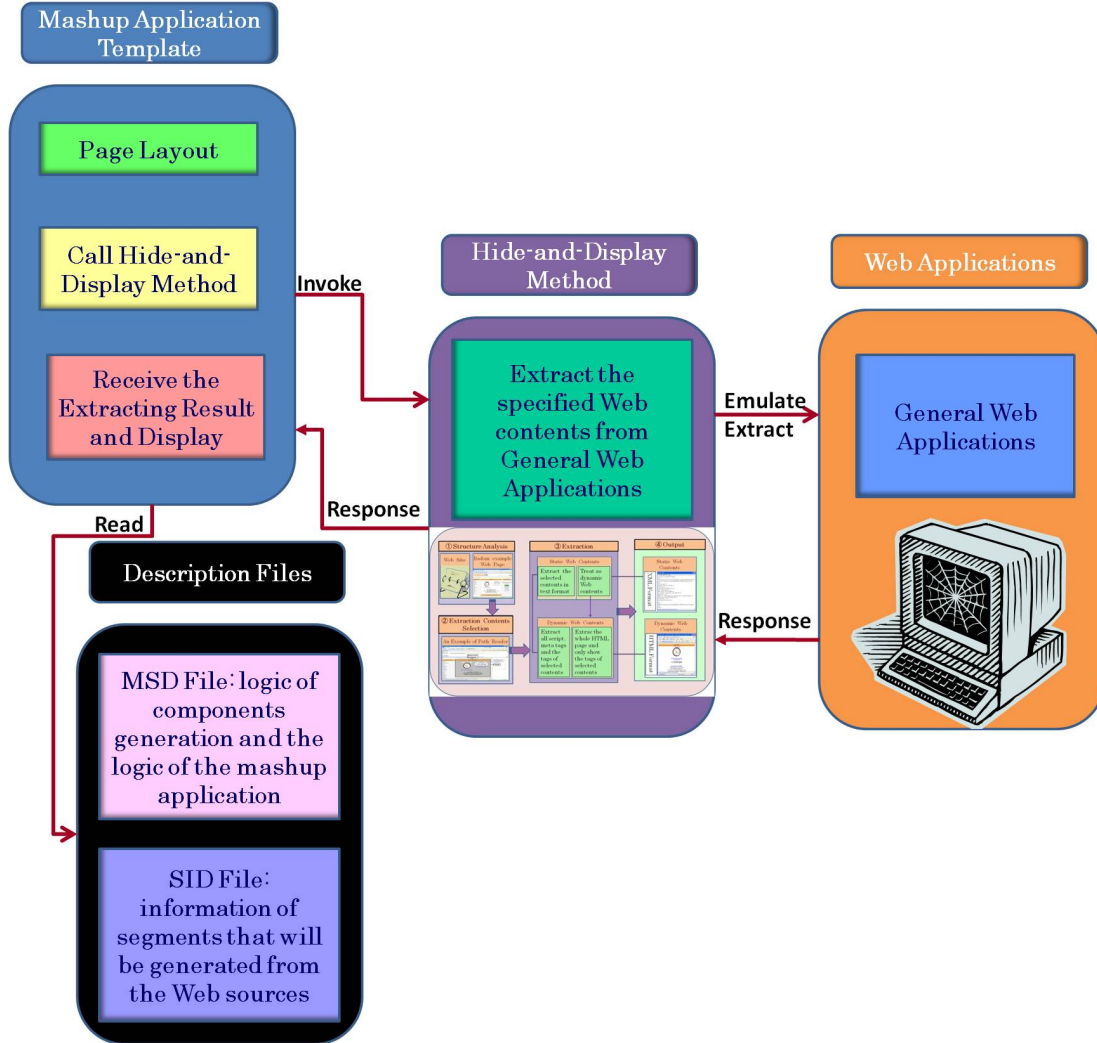


Figure 5.1: Description Based Mashup of Web Applications - Overview

As we explained in Chapter 4, the execution of hide-and-display method need several parameters users specified. We record this information into an XML-based file. We call it description file. Furthermore, the logic of the mashup application is also recorded in an XML-based description file, so that users can define and modify the information easily. In another word, we use two description files in the DBMWA method.

We use an XML-based file, Segment-Information Description (SID) file, to describe

the information of segments that will be generated from the Web applications. The segments will be used to generate the components that will be used to build the mashup applications. We use an XML-based file, Mashup-Scenario Description (MSD) file to describe the logic of components generation and the logic of the mashup application. Our approach allows that one mashup component is generated from one or multiple segments. The detail of the description files will be explained in Section 5.3.

5.2 Mashup Application Template

The layout of the mashup application template is simple, as shown in Figure 5.2.

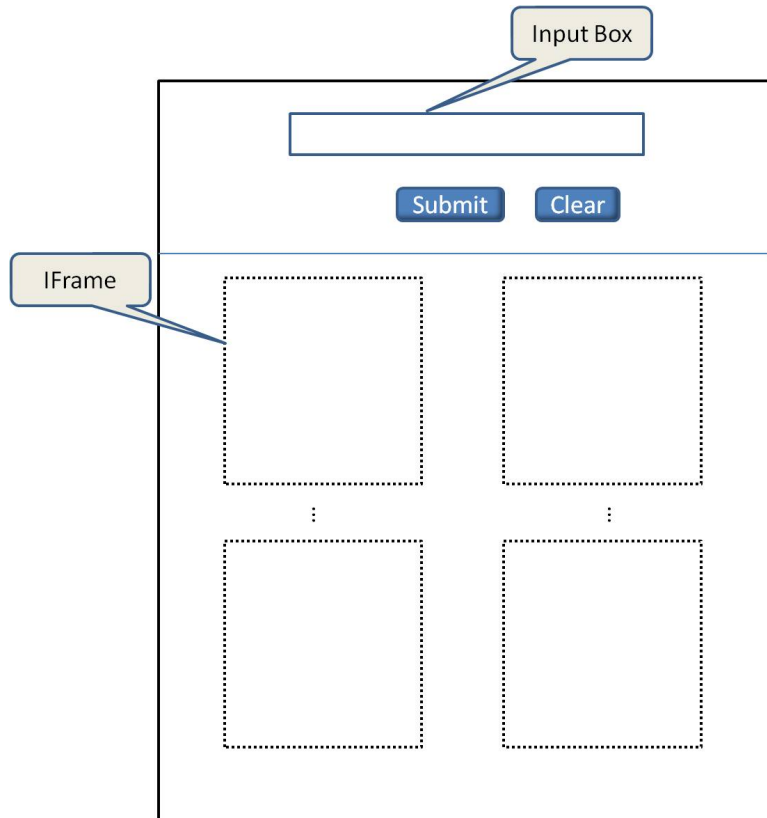


Figure 5.2: Layout of Mashup Application Template - Demonstration

Considering the usage of certain request-submit functionalities of Web applications, we put an input box on the Web page which is used to input the searching keyword. The containers of the extracting results are IFrames (36). IFrame (Inline Frame) is an HTML element which makes it possible to embed an HTML document inside another

HTML document. While regular frames are typically used to logically subdivide the Web content of one Web page, IFrames are more commonly used to insert Web content from another web site into the current Web page. Also, IFrames are supported by all major browsers. IFrames are created dynamically according to the number of the mashup components. After the resulting page is shown completely, users can move IFrames by dragging and dropping to adjust the location.

To call the hide-and-display method, we need to read the two description files and get data before sending request. We supply an application to create the description files, as shown in Figure 5.3. Thus, the structure and the tag names of the XML files are fixed. So we can get the data easily. To show the extracting result, just write it to the related IFrame.

5.3 Definition of Description Files

Definition of SID File

A segment can be part(s) of a Web source. For example, a segment generated from a Web application type of Web source can be a paragraph of text on a Web page. The SID file is constituted by several separated segments. The information that is necessary for the generation of the segments is recorded in the SID file. Each segment has following items.

- id: We specify the unique name of each segment.
- startPageURL: We specify the URL of a Web source where we can submit the request. For the Application type of Web source, it should be the URL of a web page where we can submit the searching request to get the target Web page.
- inputType: We specify the type of searching method on the start page. It should be one of {LinkList (anchor list), OptionList (drop-down option list in selectbox) or InputBox (text input field) and None}. We just support these four basic input types by now. We would like to do experiments about other input types, for example radio button, as future work.
- inputArea: We specify the path of the input Web content on the start page. If there is other content with the same "inputType" in the start page, we need to

specify which one is going to be used. Users can give a value of "null" instead of the path, which will be treated as the path of *body* node by default. The path that we use here and other items are XPath-like expressions. The value of path can be gotten easily by using the tool we supply.

- input: We specify how to get the searching keyword of emulation. It could be the input value from user interface, or be the value transferred from other segment. When the searching keyword is from other segment, we use the "id" of the segment to describe it.
- extractionContents: We specify the Web content to be extracted by path. Here we support multiple Web contents extraction.
- contentsType: We specify the data type of the target Web content. It could be dynamic content (mashup, flash or function part) or static content (body, bullet, bullet list, definition list bullet, definition list, div, form, heading, input, label, text, image, link, object, select, paragraph, span, table or text area). For static Web contents, another two sub-items of information is necessary. They are "returnNodeName" and "extractionType". The "returnNodeName" is any word that used to describe the image of the extracted information. This word can be used in the style file. The "extractionType" is a selection from {single text, single link, single object, continue text, continue link and continue object} to describe the type of extracting action. For dynamic Web contents, a child node "extractionMethod" is necessary to specify the extracting method from {Compression Method and Visibility Control Method}, which we explained in Section 4.4.1. Default value is *Compression Method*. Because that the loading speed of the extracting result is faster.
- returnType: We specify the format of the information that we should wrap into. It should be one of {HTML and XML}. We support the mergence of segments returned in the XML format into a mashup component. The "returnType" of dynamic Web content must be HTML.
- output: We specify the information that will be transferred to other Web sources. This information will be used in the generation of segments. For example, it will

5.3 Definition of Description Files

be used as input value for the Web application. This information is recorded in path format.

- memo: We record the attribute of the segment is user's words. When users create the MSD file based on the SID file, users can check the memo that they wrote.

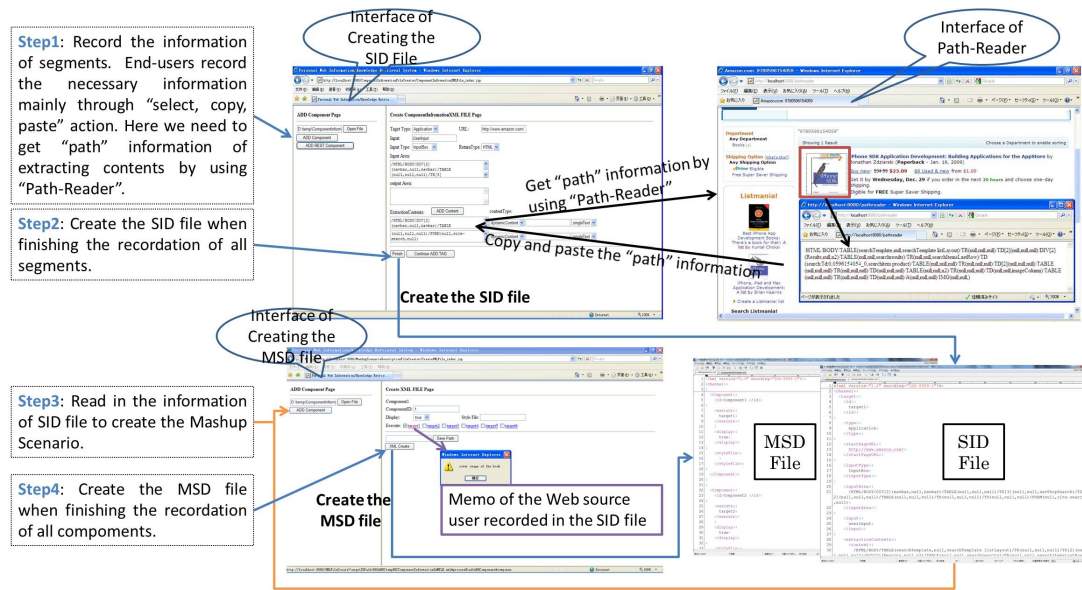


Figure 5.3: Generator of Description Files - An Example Process of Generating Description Files

Definition of MSD File

The MSD file is constituted by several components. A component is generated from one or multiple segments. The creation of a MSD file must be based on a SID file. The location and the name of the SID file will be recorded in the MSD file. Each component has following items.

- id: An index number is used to specify the unique name of each mashup component. This index number also decide the proceeding order of the mashup components.
- execute: We specify the segment(s) from which the component will be generated. This item is recorded by one or several "id" values of the segments that exist in the SID file. If it is recorded by several "id" values, the "returnType" of these segments must be "XML" and they will be merged into one XML file.

- display: We specify whether the component will be displayed or not. It should be one of {True and False}.
- styleFile: We specify the style information of component which is generated from the segment(s) that "returnType" is XML.

We supply an Web application called Description-Generator, which can generate the two kinds of description files based on the user's *selection* , *copy* and several input actions, as shown in Figure 5.3. We suppose the users of our approach have the basic knowledge of HTML and XML.

5.4 Process of DBMWA Method

As shown in Figure 5.4, the whole process of the DBMWA method includes the following steps.

- First, the user sets up the location of MSD file and send the request. The mashup application template that we supplied will access the MSD file and get the information of mashup components. According to the information, the mashup application template will access the SID file to get the necessary information of each segment.
- Then, based on the information recorded in the SID file, the "Extractor" will access the Web sources and emulate the requests to get the original Web pages.
- Thirdly, the "Extractor" will extract the target part(s) according to the record items in the SID file and generate the segments, which will be used to generate the mashup components according to the record items in the MSD file.
- Finally with the display and layout information from MSD file, the mashup application template will show the result.

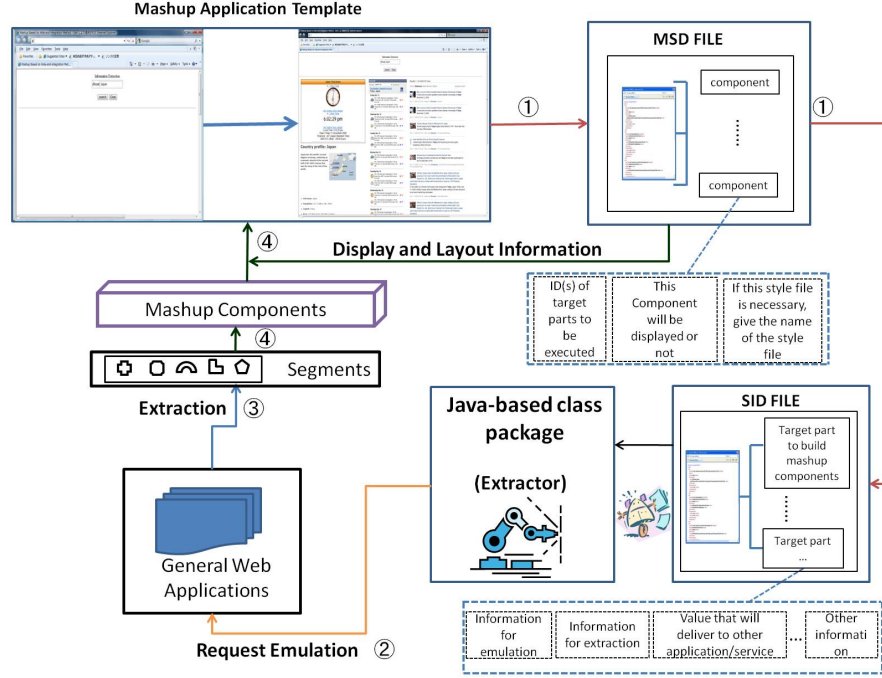


Figure 5.4: Description Based Mashup of Web Applications - Process

5.4.1 Generation of Mashup Components

The mashup components are generated from the segments recorded in the SID file. One component can be generated from one or multiple segments. If the component is generated from multiple segments, the "returnType" of the segments must be "XML" type. We will merge all the segments into one XML file and show the result with the relative style file recorded in the MSD file.

The process that generates a segment from general Web applications is as follows.

- Step 1.** We go to the start Web page of the Web application that is specified in the SID file.
- Step 2.** According to the value of "inputType" and "input", we emulate the searching function of the application to get the target Web page.
- Step 3.** We search for the target Web contents by paths specified in "extractionContents" of the SID file by using hide-and-display method. For static Web contents, if the users want to keep the same layout of target parts with the original Web

page, they can be treated as dynamic Web contents by setting the value of "contentsType" into *dynamic*. Furthermore, to ensure the extracting accuracy of the contents, we support that multiple pathes are used to describe one content. In this case, we will search the target Web content according to the specified pathes one by one.

Step 4. We get the value that will be transferred to other Web sources if the "output" item is not "null". The value is also extracted by using partial information extraction method. We store the extracted value into a storage space indexed by the segment id.

5.4.1.1 Cooperation of Web Applications

The cooperation of Web applications is reflected by the information transfer between them in the DBMWA method. The process of information transfer is shown in Figure 5.5. The whole process includes the following steps.

Step 1: We use the same algorithm as searching the target Web content to find the HTML node that is specified as "output" in the SID file.

Step 2: We extract the text excluding the HTML tags of the node.

Step 3: We store the data into a storage space indexed by the segment id.

Step 4: If certain component's value of "input" is the id of a segment, we use the data stored in the storage space to do the emulation and obtain the target Web page.

Step 5: We transfer the extracted target part(s) to the client-side mashup application as response result through the "Extractor".

We classify the information transfer between the Web applications that are used to generate mashup components into following two models.

- **Parallel Model.** If the information transfer between the Web applications used in a mashup application is from one application to other all applications, we call it Parallel Model. For example, information is transferred from Web application A to application B, C, ..., and N.

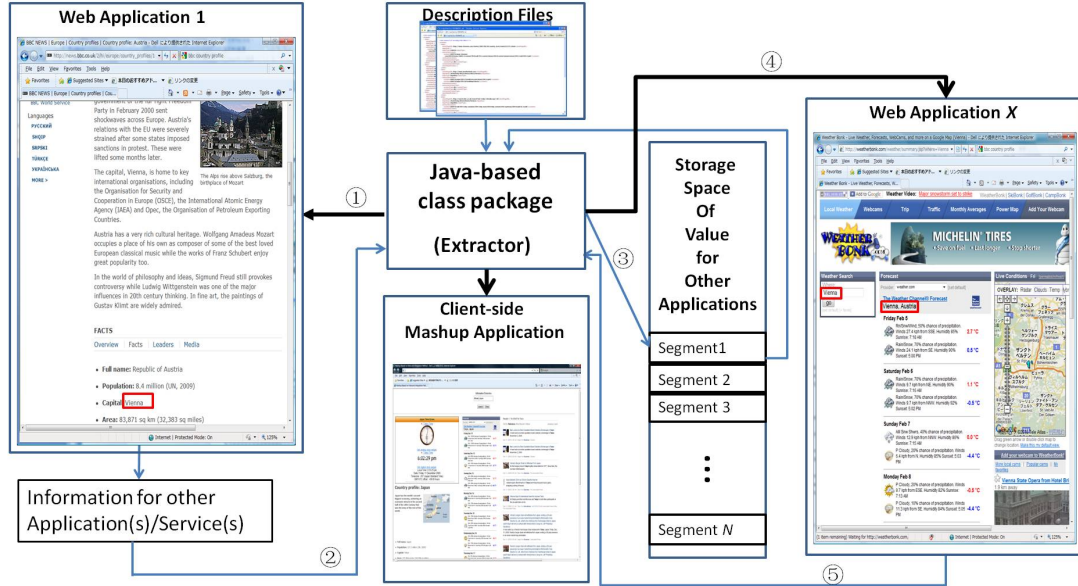


Figure 5.5: Process of Information Transfer - Cooperation of Web Applications

- **Sequential Model.** If the information transfer between the Web applications used in a mashup application is from one to one sequentially, we call it Sequential Model. For example, information is transferred from Web application A to Web application B, then from Web application B to Web application C, and so on.

We create the mashup components one by one according to the recorded order in the MSD file. As a result, the segment's output can only be used to the generation of other segments which are executed after it. We support the parallel model and the sequential model information transfer with the above restriction. We also support the mix of parallel model and sequential model information transfer in one mashup application.

5.4.1.2 Component Consisted of Multiple Segments

If the "execute" item of a component is consisted by more than one segment, the extracting result of every segment will be merged into one as the result of the component. The "returnType" of these segments must be "XML" and they will be merged into one XML file by using following codes.

Listing 5.1: Merge the XML Type of Extracting Results

```

1 public class MergeXmlFile {
2     public Document merge(String expression, String[]
        strXMLFiles){
3         Document result = null;
4         try{
5             XPathFactory xPathFactory = XPathFactory.newInstance();
6             XPath xpath = xPathFactory.newXPath();
7             XPathExpression compiledExpression = xpath.compile(
                expression);
8             result = merge(compiledExpression, strXMLFiles);
9         }catch (Exception e) {
10             e.printStackTrace();
11         }
12         return result;
13     }
14     public Document merge(XPathExpression expression, String[]
        strXMLFiles) {
15         Document base = null;
16         try {
17             DocumentBuilderFactory docBuilderFactory =
                DocumentBuilderFactory.newInstance();
18             docBuilderFactory.setIgnoringElementContentWhitespace(
                true);
19             DocumentBuilder docBuilder = docBuilderFactory.
                newDocumentBuilder();
20             base = docBuilder.parse(new InputSource(new
                StringReader(strXMLFiles[0])));
21             Node results = (Node) expression.evaluate(base,
                XPathConstants.NODE);
22             if (results == null) {
23                 throw new IOException(strXMLFiles[0] + ":␣expression␣
                    does␣not␣evaluate␣to␣node");
24             }
25             for (int i = 1; i < strXMLFiles.length; i++) {
26                 Document merge = docBuilder.parse(new InputSource(new

```

```

27         StringReader(strXMLFiles[i])));
28     Node nextResults = (Node) expression.evaluate(merge,
29         XPathConstants.NODE);
30     while (nextResults.hasChildNodes()) {
31         Node kid = nextResults.getFirstChild();
32         nextResults.removeChild(kid);
33         kid = base.importNode(kid, true);
34         results.appendChild(kid);
35     }
36 } catch (Exception e) {
37     e.printStackTrace();
38 }
39 return base;
40 }

```

5.5 Implementation of DBMWA Method

We build several mashup applications to verify that DBMWA method is applicable to general Web applications under Windows XP SP3, Windows Vista SP1 and Windows 7 with Internet Explorer7/8 and JDK 1.6. The DBMWA method can work well on platforms that fully support Java and JavaScript. Figure 5.6 shows a mashup application that integrates information using extracted Web contents from five Web sites: CBS News Country Facts(9), Weatherbonk.com (64) ¹, BBC News (7), BBC Country Profiles (6) and Trippermap.com (62). Our integrated Web page has the same searching facilities of these Web sites. This personal mashup Web page can show various related information for a given country name.

After giving a country name, for example "Japan", and clicking the "Search" button, we can get a resulting Web page on the left side of Figure 5.6, where following mashup components can be displayed with the same functionalities.

¹The Web site has been shut down permanently from Jan,2011.

²The local clock information is not supplied now.

5.5 Implementation of DBMWA Method

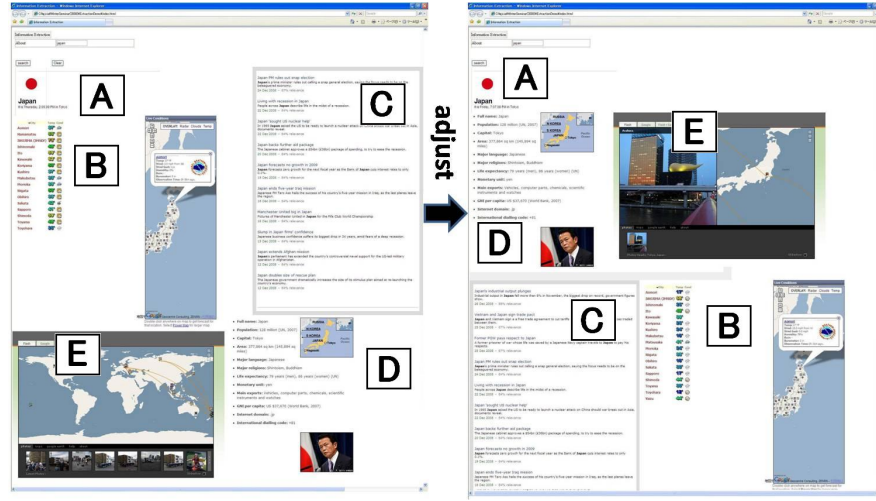


Figure 5.6: Mashup Application Example 1 - Collecting Information about a Country

Table 5.1: Contents of the Mashup Application Example 1

Component	Description of the Contents
A	Country name, country flag and local clock of capital ² from country facts of CBS News Country Facts. The local clock of capital is dynamic text created by client-side script, which changes every second.
B	Weather information for given country name from Weather-bonk.com. The weather information shown here is mashup Web content created by client-side scripts, which can respond to click and span events.
C	The latest news articles from BBC News about the given country name.
D	The country's location, basic information and leader picture from BBC Country Profiles.
E	Pictures from Trippermap.com with map, which can respond to click event and show the relevant pictures.

Users can adjust the layout by dragging and dropping the Web content containers, as shown on the right side of Figure 5.6.

5.5 Implementation of DBMWA Method

Figure 5.7 shows a mashup application that integrates four components which are extracted from four Web applications: Localtimes.info, BBC Country Profiles, Weatherbonk.com and abc News(1).

After giving a country name, for example "Japan", and clicking the "Search" button, we can get a resulting Web page as shown in Figure 5.7, where following components can be found.

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

Component 2. Given a country's name, introduction, location map and basic information from BBC Country Profiles. The capital city's name extracted here will deliver to other components as searching keyword.

Component 3. The capital city's weather information from Weatherbonk.com. The capital city's name is from "Component 2".

Component 4. The latest news articles from abc News about the capital city. The capital city's name is also from "Component 2".

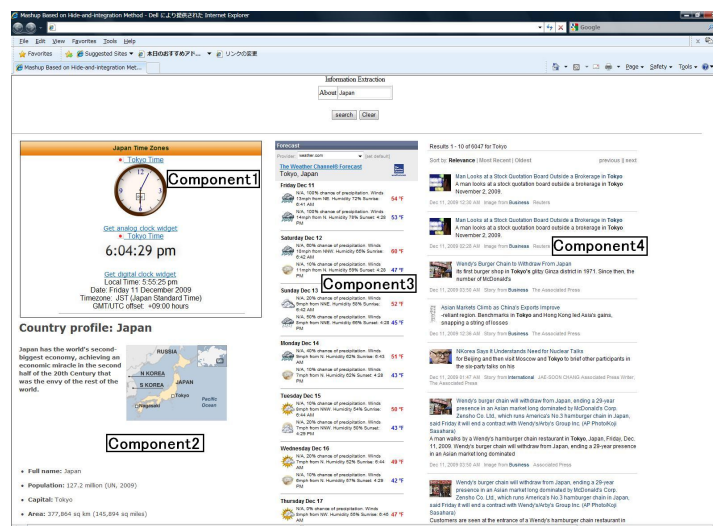


Figure 5.7: Mashup Application Example 2 - Including Cooperation between Web Applications

Chapter 6

Integration of Web Applications and REST Web Services Without Programming

A Web service is a software system designed to support inter-operable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.(63)

The "Web API" is sometimes considered a synonym for web service. Web service APIs that are used in mashup applications are mainly two styles, SOAP and REST. The existing approaches that can use Web service APIs to build mashup applications automatically or semi-automatically usually have some kind of limitation. Some approaches limit the usable Web service APIs to the pre-defined APIs in the approaches. Yahoo Pipes is a typical example. Some approaches need special process before using the APIs, for example, re-generation or registration. ServFace and Mixup are well-known examples.

In fact, integrating the SOAP type Web services automatically without any pre-process is difficult because that we need the client side programm to send request. However, for the REST services, it is possible. Because that REST services do not require XML, SOAP, or WSDL service-API definitions and is easy to send request.

Furthermore, now the Web 2.0 applications typically have moved away from SOAP based web services towards more direct REST style communication (17). The fact that 68% of the protocol usage by APIs are REST while 23% are SOAP also can prove the above viewpoint¹.

Although it is easy to send request to REST Web services, it is not easy to extract information from the returned results² without programming. Thus, in this chapter, we propose an approach that allows users to build mashup applications with general Web applications and REST Web services without programming. We call it Mashup with Web Applications and REST Services (M-WA&RS) method. The M-WA&RS method supports the information transfer between the mashup components, which can be the part(s) extracted from a Web application or the information from a REST Web Service. In addition, our approach uses the similar technic to process the Web application and REST Web service type of Web sources. It will be easy for users to use.

6.1 Architecture of M-WA&RS Method

The overview of the M-WA&RS method is shown in Figure 7.1. It is an approach based on the DBMWA method by extending the SID file, hide-and-display method and the Mashup Application Template. To extract information from REST Web services automatically, we need to mark and record the relative information first. By converting the result that gotten from the REST service server to HTML format, we can use the same method explained in Chapter 5 to use REST Web services to do integration.

Because that our purpose of sending request to REST services is to get data, this approach just supports the GET method of REST services. Usually, the Request URL format of REST service APIs is consisted of a base part and several parameters. To treat the REST service APIs automatically, the return type is also very important. We extending the SID file to record the necessary information of REST service APIs, such as parameters, return type and so on.

We extending the hide-and-display method with the following functions.

1. Convert the result from REST server to HTML format.

¹The data is provided by programmableweb.com, 10/06/13

²The data formats that are widely used for the result of REST Web services are XML, JSON and Plain Text.

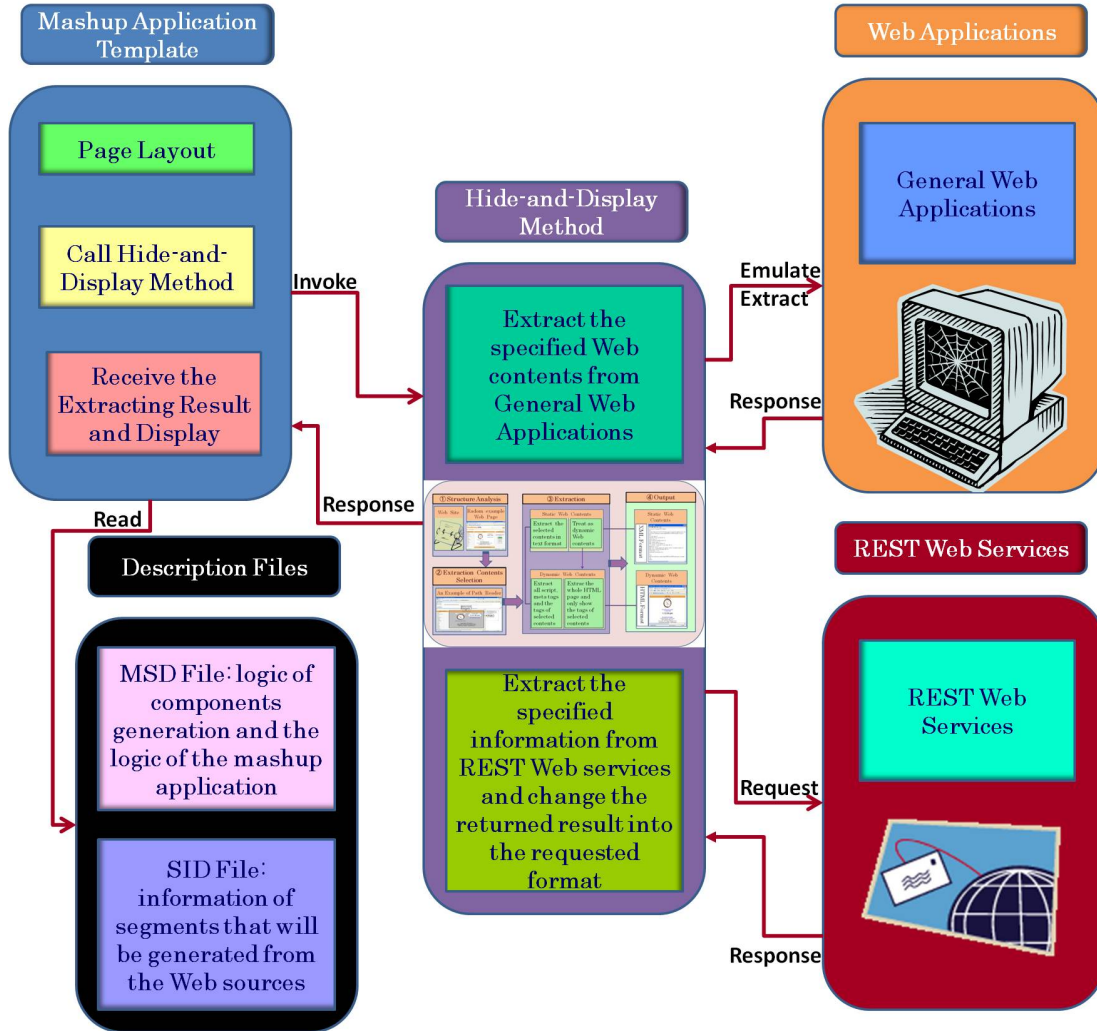


Figure 6.1: Architecture of the M-WA&RS Method - Overview

2. Generate segments from REST Web services.

The Mashup Application Template is extended to handling the new version of the Hide-and-Display method. The coding of the Mashup Application Template is shown in Appendix A.

6.1.1 Extension of the Segment Information Description File

To record the necessary information for generation segments from REST services, four items are added in SID files. They are *type*, *parameters*, *serviceResultType* and *style*-

FileForShowInHTML. Following is the completed item-definition of the SID file.

- **id**: We specify the unique name of each segment.
- **type**: We specify the type of the Web source which the segment is generated from. It should be one of {Application and REST}.
- **startPageURL**: We specify the URL of a Web source where we can submit the request. For the Application type of Web source, it should be the URL of a web page where we can submit the searching request to get the target Web page. For the REST type of Web source, it should be the basic part of the request URL.
- **inputType**: This is just necessary for the Application type of Web source. We specify the type of searching method on the start page. It should be one of {LinkList (anchor list), OptionList (drop-down option list in selectbox) or Input-Box (text input field) and None}. We just support these four basic input types by now. We would like to do experiments about other input types, for example radio button, as future work.
- **inputArea**: This is just necessary for the Application type of Web source. We specify the path of the input Web content on the start page. If there is other content with the same "inputType" in the start page, we need to specify which one is going to be used. Users can give a value of "null" instead of the path, which will be treated as the path of *body* node by default. The path that we use here and other items are XPath-like expressions. The value of path can be gotten easily by using the tool we supply.
- **input**: This is just necessary for the Application type of Web source. We specify how to get the searching keyword of emulation. It could be the input value from user interface, or be the value transferred from other segment. When the searching keyword is from other segment, we use the "id" of the segment to describe it.
- **styleFileForShowInHTML**: This is just necessary for the REST type of Web source. We specify the location and name of the style file that will be used to convert the returned result of REST Web services into HTML format. If the "serviceResultType" of the REST Web service is "Plain Text", the value of this item should be "null".

- **extractionContents:** We specify the Web content to be extracted by path. Here we support multiple Web contents extraction.
- **contentsType:** We specify the data type of the target Web content. It could be dynamic content (mashup, flash or function part) or static content (body, bullet, bullet list, definition list bullet, definition list, div, form, heading, input, label, text, image, link, object, select, paragraph, span, table or text area). For the static content, another two sub-items of information is necessary. They are "returnNodeName" and "extractionType". The "returnNodeName" is any word that used to describe the image of the extracted information. This word can be used in the style file. The "extractionType" is a selection from {single text, single link, single object, continue text, continue link and continue object} to describe the type of extracting action. For dynamic Web contents, a child node "extractionMethod" is necessary to specify the extracting method from {Compression Method and Visibility Control Method}, which we explained in Section 4.4.1. Default value is *Compression Method*. Because that the loading speed of the extracting result is faster.
- **returnType:** We specify the format of the information that we should wrap into. It should be one of {HTML and XML}. We support the mergence of segments returned in the XML format into a mashup component. The "returnType" of dynamic Web content must be HTML.
- **output:** We specify the information that will be transferred to other Web sources. This information will be used in the generation of segments. For example, it will be used as input value for the Web application type of Web source and as parameter for the REST type of Web source. This information is recorded in path format.
- **parameters:** This is just necessary for the REST type of Web source. The parameters will be added to the base part of the request URL recorded in "start-PageURL" orderly. There are three types of parameters, the information from the "output" of other segment, from the "userInput", or a string.

- **serviceResultType**: This is just necessary for the REST type of Web source. We specify the format of the information returned by the REST service by this item. We support three kinds of format now, XML, Plain Text and JSON.
- **memo**: We record the attribute of the segment is user's words. When users create the MSD file based on the SID file, users can check the memo that they wrote.

6.1.2 Extension of the Hide-and-Display Method

6.1.2.1 Converting the Result of REST Service to HTML Format

We support three kinds of formats of the result from REST server, XML, JSON and Plain Text.

For the Plain Text format of result, the text can be directly shown as HTML format. It does not need any conversion.

For the XML format of result, we convert it to HTML format by using the style file¹ that prepared by users. As an example, Figure 6.2 shows the process of how we get the value of "extractionContents" of a REST service which return the result in XML format. The figure can also explain the image of how we convert the returned result of services into HTML format.

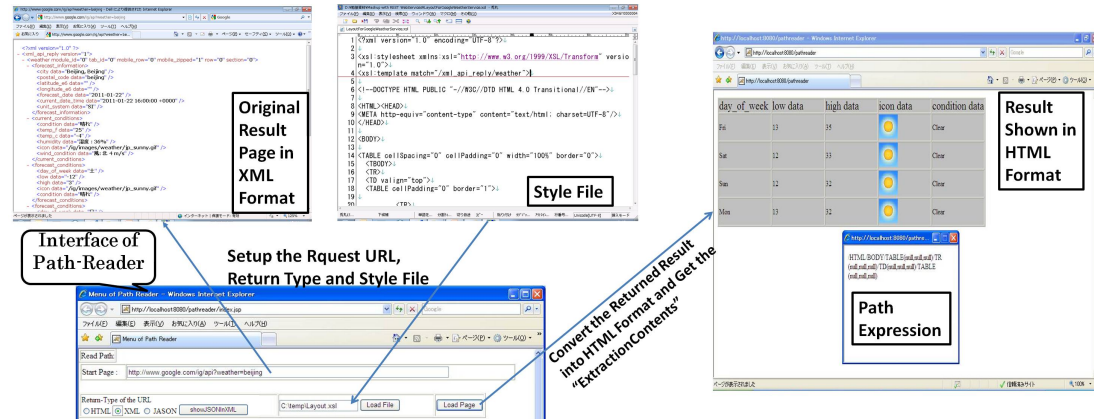


Figure 6.2: Converting the Returned Result of Services into HTML Format - Process in Path-Reader

For the JSON format of result, we first convert it to XML format automatically

¹Extensible Stylesheet Language (XSL) file

by using the Json-lib (39) library, as shown in Listing 6.1. Then we convert the XML format to HTML format with the style file uses prepared.

Listing 6.1: Convert JSON Format to XML Format

```
1 public String getJSONToXML(String StartPage)
2 {
3     String strServiceResult = "";
4     String strXMLFormat = "";
5     try
6     {
7         InfoAbstract objInfoAbstract3 = new InfoAbstract();
8         strServiceResult = objInfoAbstract3.getRESTAPIResult(
9             StartPage);
10        XMLSerializer serializer = new XMLSerializer();
11        JSON json = JsonSerializer.toJSON(strServiceResult);
12        strXMLFormat = serializer.write(json);
13    } catch (Exception e) {
14        e.printStackTrace();
15    }
16    return strXMLFormat;
17 }
```

6.1.2.2 Generating Segment from REST Web Service

To generate a segment from the REST type of Web source, the process is as follows.

Step 1. We merge the basic part of the request URL and the parameters that are specified in the SID file to generate the request URL of the REST Web service.

Step 2. We emulate the request and get the result from the service server. Then we convert the returned result of the service into HTML format with the information of "styleFileForShowInHTML" which is specified in the SID file if it is necessary.

Step 3. According to the "extractionContents" specified in the SID file, we extract the information as what we do towards the Web application type of Web source, and then generate the segment. If the result format of the REST service is plain text, we will extract the whole text.

Step 4. We extract the value that will be transferred to other Web sources if the "output" item is not "null".

6.2 Implementation of M-WA&RS Method

We build several mashup applications to verify that M-WA&RS method is applicable to general Web applications and REST Web services.

Figure 6.3 shows an example mashup application that extracts the latest news of given keyword and translate the title of the first news into Japanese. We design the keyword to be a city name. In addition, this mashup application gets weather information of the given city. The detail of the Web resources that are used in this mashup application is listed in Table 6.1. The MSD file and SID file of this example are listed in Appendix B.

The screenshot shows a web browser window titled "Mashup With Web Applications and REST Web Services - Windows Internet Explorer". The address bar shows "Google". The page content includes a search bar with the text "About:tokyo" and buttons for "search" and "Clear". Below the search bar, there are two main components. Component A is a list of news items with headlines like "Nikkei tumbles, Mideast unrest spurs profit-taking" and "Japan's Aeon to take 12.31 percent stake in Parco". Component B is a box titled "Title of first news shown in Japanese" containing the text "日経平均株価は中東不安が、利食いスリアス、宙返り". Component C is a table showing weather data for Tokyo, including day of week, low data, high data, icon data, and condition data.

day_of_week	low data	high data	icon data	condition data
Tue		51		Clear
Wed	37	53		Partly Sunny
Thu	42	59		Cloudy
Fri	46	60		Partly Sunny

Figure 6.3: Mashup Application Example 3 - Integration with Web Applications and REST Services

6.2 Implementation of M-WA&RS Method

Table 6.1: Contents of the Mashup Application Example 3

Component	Web Source (Type)	Description
A	reuters news (Application)	including one segment, show the latest news of given city
B	Google Translate API (26) (REST-JSON)	including one segment, translate the news title gotten from Component A from English to Japanese
C	Google Weather API (27) (REST-XML)	including one segment, show the weather information of given city

Figure 6.4 shows an example mashup application that extracts the prices of a book from different country's online book stores and convert the prices from local currency to US dollar through REST Web service. The input keyword of this mashup application is the ISBN code of a book.

Display of the components has been adjusted. The components that do not be marked are the logo images of every online book stores. The Web resource information of other components that are used in this mashup application is listed in Table 6.2. Figure 6.5 illustrates how it is possible to then use a specific Web service to perform the currency exchange. Part of the MSD file and SID file that are use in this example is listed in the Appendix C.

6.2 Implementation of M-WA&RS Method

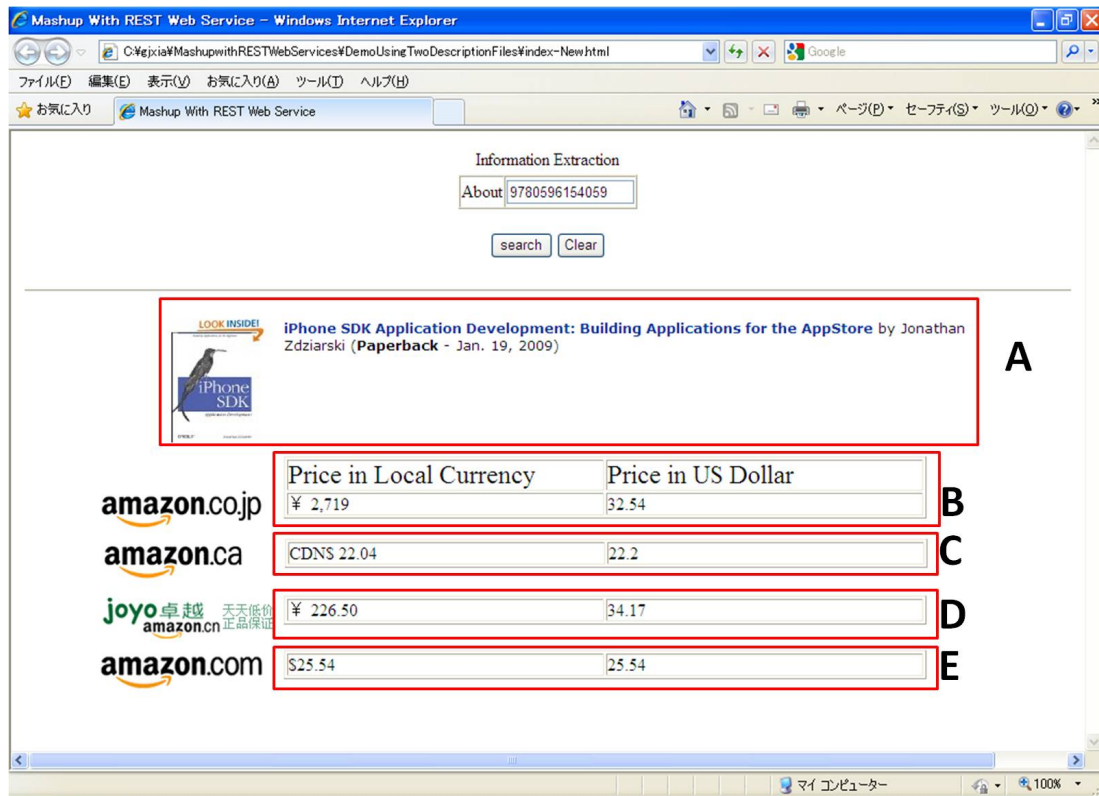


Figure 6.4: Mashup Application Example 4 - Integration with Web Applications and REST Services

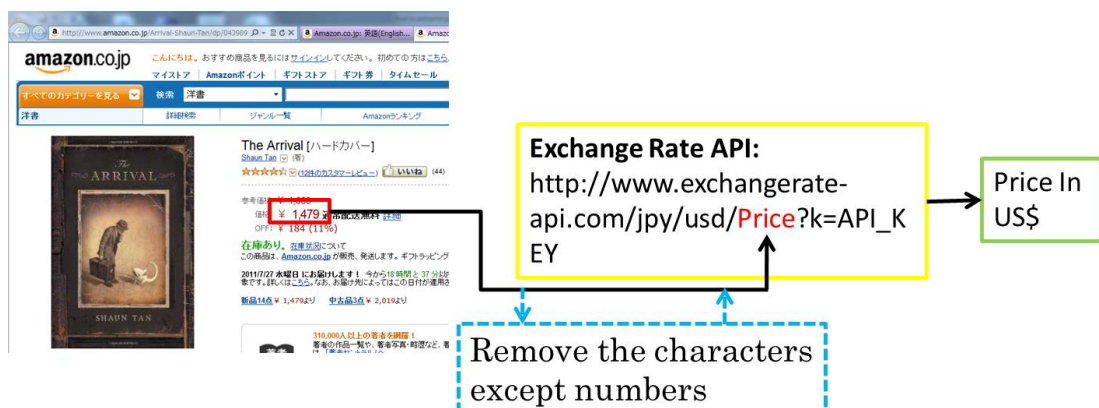


Figure 6.5: Image of Using Web Service API - Currency Exchange API

Table 6.2: Contents of the Mashup Application Example 4

Component	Web Source (Type)	Description
A	amazon.com (5) (Application)	including one segment, shows the cover and title of given book.
B	amazon.co.jp (4) (Application) exchangerate-api (21) (REST-PlainText)	including two segments, shows the price in Japanese Yen extracted from amazon.co.jp and in US dollar exchanged by REST service.
C	amazon.ca (2) (Application) exchangerate-api (REST-PlainText)	including two segments, shows the price in Canada dollar extracted from amazon.ca and in US dollar exchanged by REST service.
D	amazon.cn (3) (Application) exchangerate-api (REST-PlainText)	including two segments, shows the price in Chinese Yen extracted from amazon.cn and in US dollar exchanged by REST service.
E	amazon.com (Application)	including one segment, shows the price in US dollar extracted from amazon.com.

Chapter 7

A Method for Flexible Integration through End-user Programming

Integrating the published Web services through end-user programming is a natural and flexible method to build a new Web application based on the information from the Internet. However, this kind of integration has a limitation of the information resource. Because that many Web sites publish information in Web application format instead of Web services. For example, we can get the basic information of a selected country from the BBC Country Profiles, but this Web site does not provide this functionality as a Web service. In addition, although research(80) shows that half of the survey respondents would like to build mashup applications. However, majority of the survey respondents consider themselves non-programmers. Being distinguished from the professional programmers, end-user programmers, were defined as people who have computational needs, but do not want to become professional programmers(51). They prefer using third party technologies with simple interface and little programming to meet their needs. End-user programmers outnumber professional programmers now, about 90 million (58). The respondents' experience with Web technologies are on the low end except HTML in research (80).

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. J. Cao et al. (8) 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 for keeping the flexibility of design and saving the time of development.

(31) proposed a Web information extraction method to realize the integration of Web application and designed a Java-based class package with three functions. However, this approach can only extract static Web contents from Web applications.

Marmite (68) 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 (XUL). 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 users can choose is limited to the supplied operators.

Some approaches are proposed to construct Web services based on the Web applications for end-user programming. Pollock (47) can generate Web services from the form-based query interfaces of a Web site. It generates wrappers using XWrap, WSDL file using 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(60) generates Web services based on the page-transition of existing Web applications instead of the data structures behind them. (35) 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. (30) also provides a Web services generation method based on information extraction 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 users, and it is extremely unlikely that the constructed Web services can support all the needs of end-users in building mashup applications.

Differing from the above approaches, in this chapter, we propose an approach that can help the users with programming experience focus on the mashup scenario when they build mashup applications instead of how to get information from the Web sources. As shown in Figure 7.1, we wrapped several functions that can extract any part(s) from general Web applications according to the given search keyword into a Java-based

library. Users 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, such as merging XML files according given XPath, converting JSON format to XML format etc., are also wrapped in.

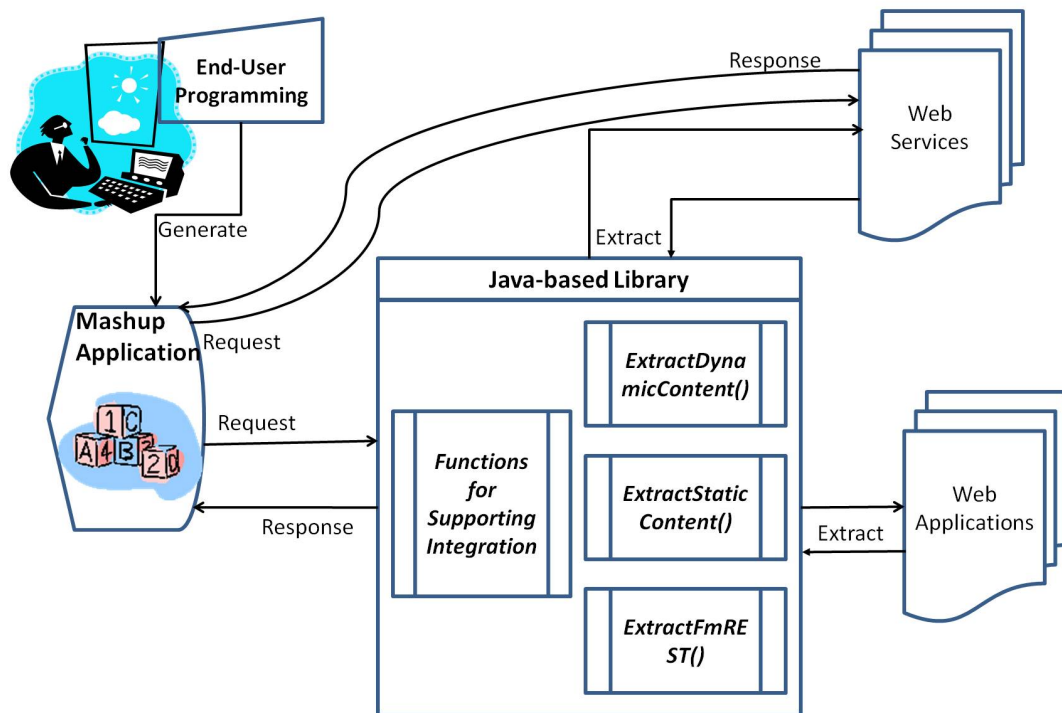


Figure 7.1: Overview of Using the Library - through End-User Programming

7.1 Constituents of the Library

The library we designed is consisted by two classes of functions. One class is the functions that are used to do extraction. Another class is the functions that can be used to help the integration.

7.1.1 Functions for Extraction

Three functions for extraction are supplied in the library to handling with different kinds of Web sources. The data type of the returned result from the functions is *String*

Array with two elements. First one is the extracting result. The second one is the information that will be transferred to other Web sources.

1. Extracting Dynamic Content from Web Application

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

This function is used to extract dynamic Web contents from general Web applications. As we mentioned, we designed two methods 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 size 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 is listed in Table 7.1.

2. 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. Users can convert this XML file into HTML format by using the *transXMLtoHTML* function which will be explained in Section 7.1.2. 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 with those listed in Table 7.1. The parameter *contentsType* is the data types and the XML's node names of the extracting target Web contents, which is recorded as *NodeName:DataType* in an array format. *NodeName* is any user 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}.

Table 7.1: Parameter Definition of *ExtractDynamicContent* Function

Parameter Name	Description
keyword	is the searching keyword for emulating the functionality of the application.
submitPage	is the URL of a Web page where we can submit the request of the Web application' functionality that is emulating.
inputType	is the type of the emulating functionality, which should be one of {LinkList, OptionList, InputBox and None}.
inputArea	is the location of the HTML <form> that includes the User-Interface part of the emulating functionality, which can be gotten by using the Path-Reader application.
extrContents	is the path expressions of the Web contents to be extracted, which can be gotten by using the Path-Reader application and recorded in array format.
methodtype	is value selected from {Compression Method and Visibility Control Method}, which specifies the extracting method.
output	is the information that will be transferred to other Web sources, which is recorded by path expression and gotten by using the Path-Reader application.

3. Extracting Information from REST Service

String[] ExtractFmREST(requestURL, servReturnType, styleFile, extrContents, returnNodeName, output)

This function is used to send request to REST services, then extract information from the result. We convert the returned result of a REST service into HTML format, then specify and extract the necessary information. The extracting result is a XML format document. Users can convert this XML file into HTML format by using the *transXMLtoHTML* function which will be explained in section 7.1.2. The parameters of the function is listed in Table 7.2.

Table 7.2: Parameter Definition of *ExtractFmREST* Function

Parameter Name	Description
requestURL	is the request URL of the REST service.
servReturnType	is the format of the information returned by the REST service, which should be one of {XML, Plain Text and JSON}.
styleFile	is the location and name of the style file that is used to convert the returned result of REST Web services into HTML format. If the "SRT" of the REST Web service is "Plain Text", the value of this parameter should be "null".
extrContents	is the path expressions of the Web contents to be extracted, which can be gotten by using the Path-Reader application and recorded in array format.
returnNodeName	is any word to describe the image of the extracting information and used as the node name of the resulting XML file.
output	is the information that will be transferred to other Web sources, which is recorded by path expression and gotten by using the Path-Reader application.

7.1.2 Functions for Supporting Integration

Four functions are designed for supporting the integration as follows.

1. Converting JSON Format to XML Format

The results that we receive from the servers of Web service APIs could be in different formats. The XML format, JSON format and Plain-Text format are widely used. It is easy to show the data of XML format or Plain-Text format in HTML page. However, for the data of JSON format, we need do large numbers of coding to show it in HTML page with our designed style. Thus, we design a function to convert data in JSON format to XML format.

After converting data into XML format, users can show the JSON format result of a Web service API in a HTML page by using the *transXMLtoHTML* function

with several lines of code. In addition, users can merge multiple Web services' results in JSON format easily with the function *CombineXMLFiles*. We explain these two functions later in this section. The function which is used to convert data from JSON format to XML format is defined as following:

$$\textit{String ConvJSONtoXML}(\textit{originalJSON})$$

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

2. Converting XML Format to HTML Format

The extracted result of static Web contents is in XML format. The results that we receive from the servers of Web service APIs also could be in XML format. Showing the information in XML format on Web pages, we need to convert the information into HTML format. The function that is used to convert information from XML format to HTML format is defined as following:

$$\textit{String ConvXMLtoHTML}(\textit{originalXML}, \textit{styleFile})$$

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 (EXtensible Stylesheet Language) file, which is a Style Sheet for XML. The result of this function is in HTML format.

3. Combing 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 following:

$$\textit{String CombineXMLFiles}(\textit{originalXMLFiles}, \textit{xPathStr})$$

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 are 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. The function that is used to emulate a functionality of a Web Application is defined as following:

String Emulation(submitPage, inputArea, inputType, keyword, returnType)

where, the parameters *submitPage*, *inputArea*, *inputType* and *keyword* have the same definition with those listed in Table 7.1. 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.

7.2 Building Mashup Applications Based on the Library

Based on this Java-based library, users can build mashup applications more easily with the information from Web applications and Web service APIs with less work by programming. We build an example mashup application, which is shown in Figure 7.2, by using the this Java-based library. This mashup application is designed to show the travel information about a country and the popular destinations in the country. It has following functions.

1. Select a country from the country list, four part of information 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 city.
 - (b) the travel guild, simple introduction and some facts of the country, while the travel guide is extracted from World Travel Guide (69) and the simple introduction and some facts are extracted from BBC Country Profiles.
 - (c) the local clock of the capital city extracted from Localtimes.info, which keeps the change of time every second.

7.2 Building Mashup Applications Based on the Library

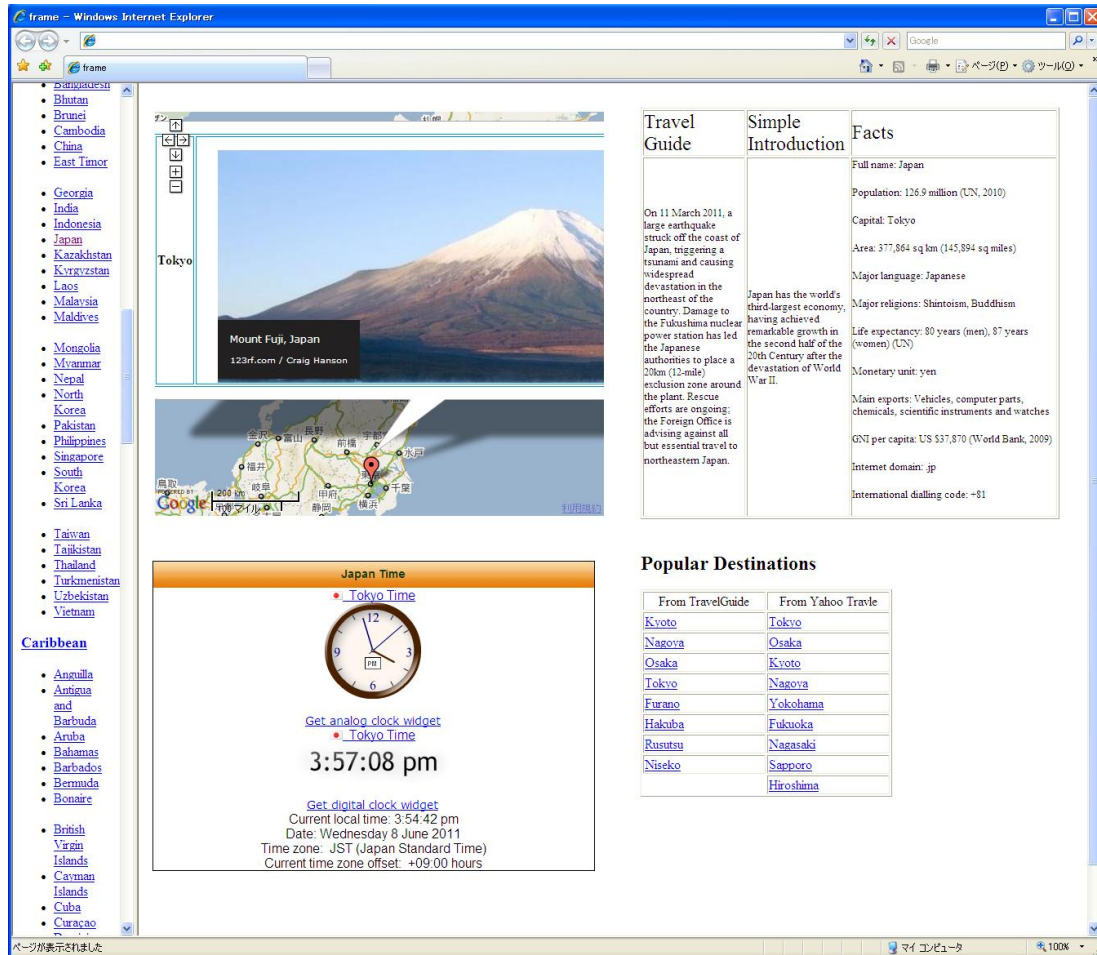


Figure 7.2: Mashup Application Using the Library - by End-User Programming

- (d) the popular destinations extracted from World Travel Guide and Yahoo Travel (76).
- Click the maker on the capital city, the photos about the country' travel spots will be shown. The photos are extracted from World Travel Guide and displayed in circular mode automatically.
 - Click any city name listed in the *Popular Destinations* list, the information about the city would be displayed in a new page, as shown in Figure 7.3. In the order of left-right and up-down, they are:
 - the map of the country with a marker on the selected city.

7.2 Building Mashup Applications Based on the Library

- (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 (61).
 - (d) the popular destinations extracted from World Travel Guide and Yahoo Travel, which is same with this part in Figure 7.2.
4. Click the maker on the city, the photos about the city' travel spots will be shown. The photos are extracted from World Travel Guide and displayed in circular mode automatically.

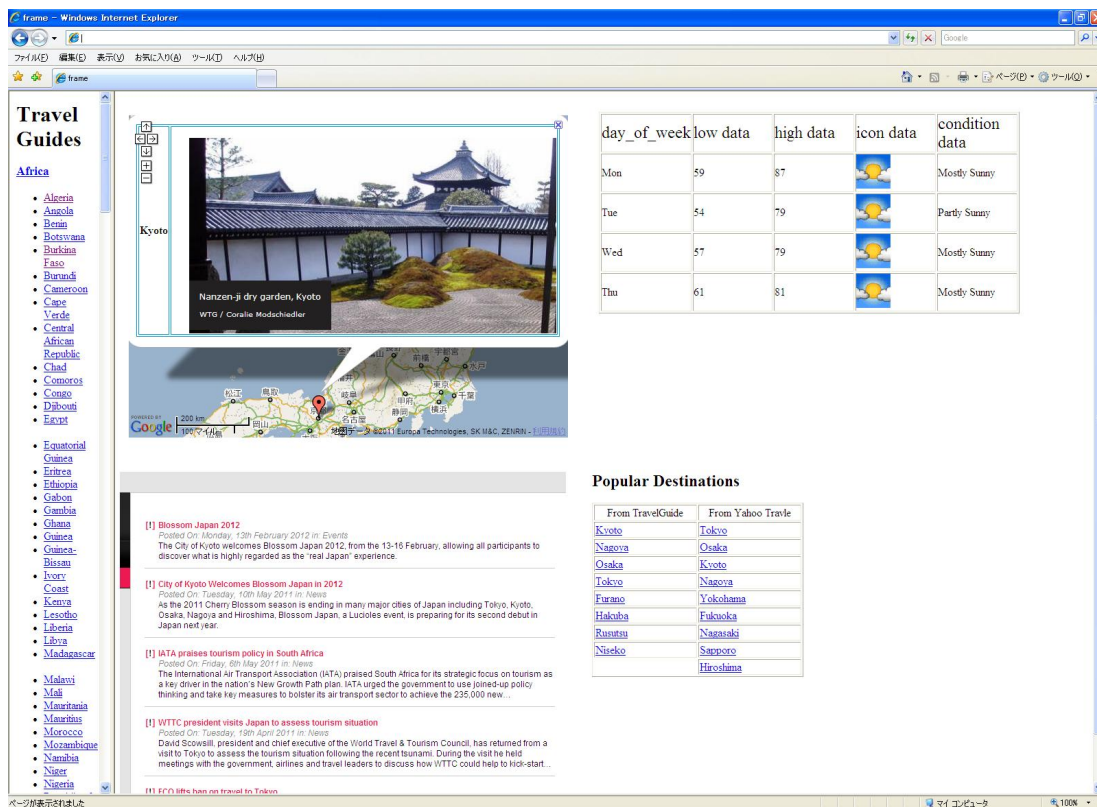


Figure 7.3: Mashup Application Using the Library - Continue

The Web sources we used in this mashup application is shown in Table 7.3. The functions we used in the Java-based library is listed in Table 7.4. We give the code sizes¹ of the actual program in Table 7.5 as a kind of user study. At the same time,

¹The code size is in the case that there is one extracting item. If the extracting item increased, the code size will increase because of the parameter numbers.

7.2 Building Mashup Applications Based on the Library

we implement the same mashup application by programming from zero. 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 could 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, List 7.1, show how to call the *ExtractDynamicContent()* by using JavaScript. The variable—"result" is used to get the extracting result. It could be a string in XML or HTML format according to the type of extracting method.

Listing 7.1: Call *ExtractDynamicContent()* by Using JavaScript

```
1 function searchTravelImg(countryNM){
2     var strInput = countryNM;
3     var strStartPageURL = "http://www.worldtravelguide.net/
4         destinations/";
5     var strInputType = "LinkList";
6     var arrExtractcontent = new Array(1);
7     arrExtractcontent[0] = "/HTML/BODY/DIV(page,null,null)/
8         DIV(page-inner,null,null)";
9     var strInputArea = "";
10    var strMethodType = "Visibility_Control_Method";
11    var strOutput = "null";
12    var result = new Array(2);
13    //get images from world travel guide
14    result = ExtractDynamicContent(strInput,
        strStartPageURL, strInputType, strInputArea,
        arrExtractcontent, strMethodType, strOutput);
    return(result[0]);
}
```

7.2 Building Mashup Applications Based on the Library

Table 7.3: Web Sources List of Travel Information Mashup Application

Web Source (Type)	Description
Google Maps (SOAP API)	Diagrammatic representation of an area.
GoogleGEO (REST API)	The response result of this API is in JSON format. We convert the result into XML format by using the function in the Java-based library. Then we extract the information of longitude and latitude and use this information to locate the marker on the Google Maps.
World Travel Guide (Web Application)	Various kinds of travel information. Here we extract the images, popular destinations, travel guide of the given country and the images of the given city, as shown in Figure 7.4.
BBC Country Profiles (Web Application)	Information of countries. Here we extract the simple introduction and facts of the given country to show. We extract the name of the capital at the same time as the "OP" item which is used as the parameter of the GoogleGEO API to get the coordinate information later.
Localtimes.info (Web Application)	Local time clock of cities. Here we extract the capital city's clock of the given country.
Yahoo Travel (Web Application)	Various kinds of travel information. Here we extract the city names from the searching result of a given country name as shown in Figure 7.5.
Google Weather (REST API)	The response result of this API is in XML format. We convert the result into HTML format by using the the function in the Java-based library.
Travel News (Web Application)	Latest travel news about the given keyword. Here we extract the top five news.

7.2 Building Mashup Applications Based on the Library

Table 7.4: Functions Used in Travel Information Mashup Application

Function	Process Description
ExtractDynamicContent()	Extract the images of the given country or city, which keep the functionality that display in circular mode automatically.
ExtractStaticContent()	Extract the travel guide text information, simple introduction and facts of the given country.
ExtractFmREST()	Extract weather information of the given city.
CombineXMLFiles()	Combine the travel guide text information extracted from World Travel Guide with the text information extracted from BBC Country Profiles.
ConvToJsonToXML()	Convert the response result of GoogleGEO into XML to get the coordinate information easily.
ConvXMLtoHTML()	Convert the combined text information into HTML format.

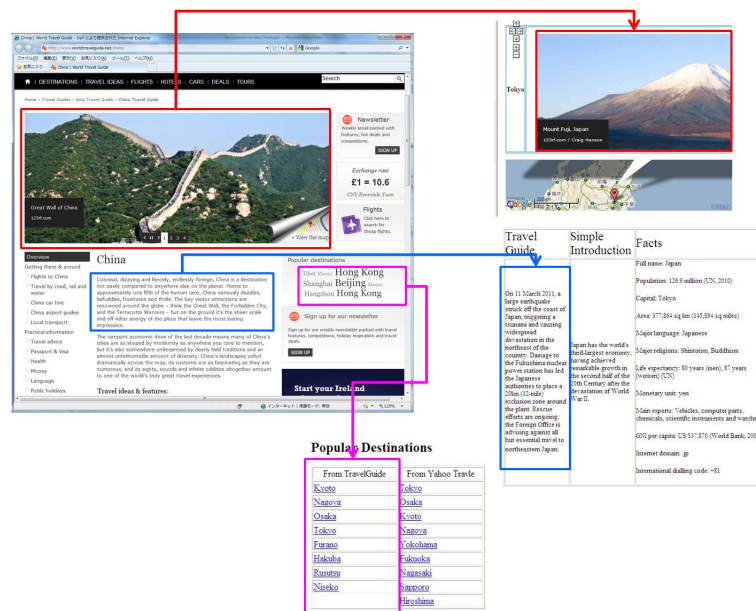


Figure 7.4: Information Extracted from World Travel Guide - Images, Text and City Names

7.2 Building Mashup Applications Based on the Library

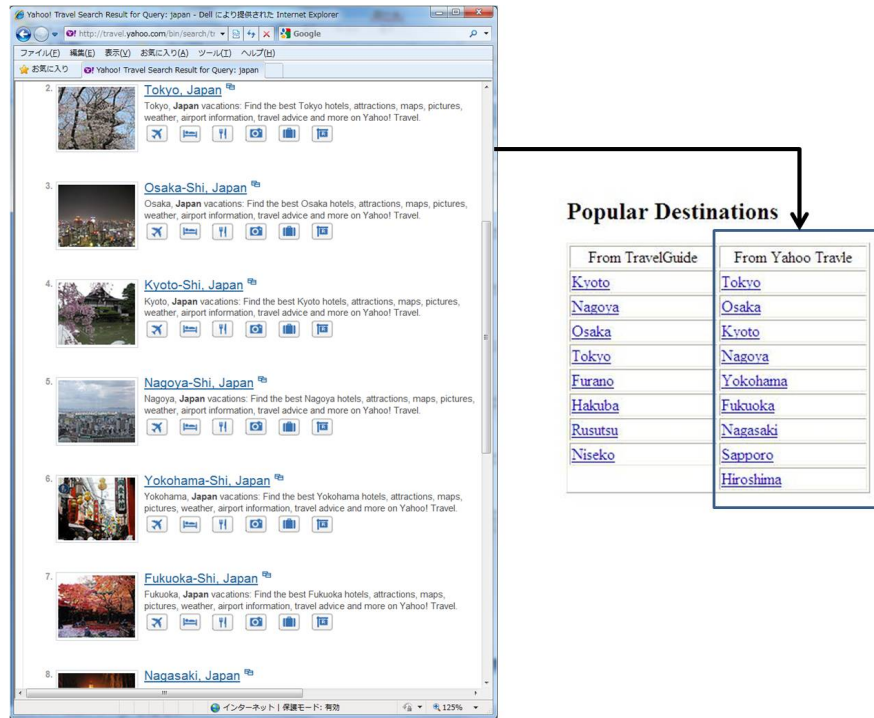


Figure 7.5: Information Extracted from Yahoo Travel - City Names

Table 7.5: Code Size of Functionalities in Travel Information Mashup Application

Object Type	Process	Code Size
Extracting Functions	Extract from Application	12
	Extract from REST Service	9
Web Services	Load Map	15
	Get Coordination Information	15
Supporting Functions	Combine XML Files	6
	Convert JSON to XML	2
	Convert XML to HTML	6
Others	Create Marker	10
	Mark City	6
	Country Selection	3
	City Selection	23

Chapter 8

Evaluation

We evaluate the methods we proposed, hide-and-display method, description based integration method and the mashup method through end-user programming in this chapter. We show the experimental results, comparison with existing approaches and the mashup security factors of our approaches.

8.1 Experimental Results

8.1.1 Applicability of Hide-and-Display Method

The experimental results in Table 8.1 could indicate that the hide-and-display method is applicable to most kinds of general Web applications and Web contents. As the results show, we can always correctly get target Web pages and extract the specified Web contents having the same functionality except in the case of SJR. The reason is that the Web site uses relative path as a parameter in client-side scripts' function codes. The line of source code is shown below, where the "ammap.swf" is the parameter using a relative path.

```
var so = new SWFObject("ammap.swf", "ammap", "550", "350", "8", "D5DEF4");
```

Unfortunately, in the case of the client-side scripts using relative path as parameter of a function, the functionality of the extracted Web content cannot work correctly.

8.1 Experimental Results

Table 8.1: Applicability of Hide-and-Display Method

Web Site	Input Type for Searching	Input Parameter	Correct Target Page	Type of Target Web Contents	Web Contents Work Correctly
Weatherbonk	Input Box	city/ country name (China, Canada ...)	Yes	dynamic content, image	Yes
CBS News Country Facts	Link List	country/ region name (Japan, China ...)	Yes	image, text, dynamic content	Yes
CNN News	Input Box	search keyword (Panda, economy ...)	Yes	text	Yes
Trippermap	None	Not Necessary	Not Necessary	dynamic content	Yes
BBC Country Profiles	Option List	country/ region name (Japan, Italy ...)	Yes	image, bullet list	Yes
Yahoo Finance (72)	Input Box	stock name (google, toyota ...)	Yes	dynamic content, text, image	Yes
X-Rates (71)	None	Not Necessary	Not Necessary	dynamic content	Yes
Infoplease	Link List	country/ region name (England, Russia ...)	Yes	image, text	Yes
SJR (57)	None	Not Necessary	Not Necessary	dynamic content	No
CIA (11)	Link List/ Option List	country/ region name (India, Spain ...)	Yes	image, text	Yes
Wikipedia (67)	Input Box	search keyword (tokyo, sasimi ...)	Yes	text, image	Yes

8.1.2 Extraction Accuracy of Hide-and-Display Method

We chose several Web applications to generate components and build mashup applications to implement our approaches. The experimental results that can indicate the extracting accuracy of the hide-and-display method are given in Table 8.2.

Table 8.2: Experimental Results about Extracting Accuracy

Web Application	Input Type for Search	Input Parameter	Num. of Checked 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	Input Box	stock name (AAPL, GOOG ...)	36	1 (dynamic content)	36	100%
Infoplease	Link List	country name (Canada, Russia ...)	204	5 (image, text)	186	91%
Yahoo News (74)	Input Box	city name (Beijing, Tokyo ...)	189	1 (text, link)	189	100%
Total	—	—	1090	—	1060	97%

As the experimental results show, the average extracting accuracy is more than 90% when we use one path expression to record the Web content. 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 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 that 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 name are the same as their capital city names. But in the Weatherbonk Web application, when we input the city's name, it was treated as a country name and we get a different page.

8.1.3 Applicability of M-WA&RS Method

To evaluate the applicability of the M-WA&RS method, we first needed to know whether our target users would be able to use it smoothly. We recruited 15 people to build mashup applications according to the mashup novels we had designed. The mashup novel and the Web sources used are listed in Table 8.3. These three mashups are build in the order of "City Information", "Book Price in Different Country" and "Country Information". We followed this order of creation so that we might observe any changes to the time a user consumed in building a mashup as their familiarity with the tool increased. The first and second mashup applications are the examples we showed in Section 6.2.

The users who participated in the experiment are all familiar with HTML and XML technology. We had two types of users.

Type 1: Users with programming experience; and

Type 2: Users with little or no programming experience.

¹The number of segments and components given here is the data that we implemented when building the mashup applications.

8.1 Experimental Results

Table 8.3: Mashup Novels for User Study of M-WA&RS Method

Mashup Novel	City Information	Book Price in Different Country	Country Information
Input	City name	ISBN Code of a book	County name
Process	Extract the latest news list and translate the title of the first news into Japanese. Get the weather information from Web service	Extract the prices of a book from different country's book stores and convert the prices from local currency to US dollar through REST Web service	Extract basic information of a country from Web applications and transfer the extracted capital city's name to a REST service to get the coordinates information. Then transfer the coordinate information to a REST service to get the time zone information
Output	The weather information, latest news list and the title of first news in Japanese	The cover image, title, publication data of the book and the logo of book stores, prices in local currency and prices in US dollar	Some basic facts of the country and the coordinates, time zone information
Web Sources (type)	reuters news (Application) Google Translate API (REST-JSON) googleWeather API (REST-XML)	amazon.com (Application) amazon.co.jp (Application) amazon.cn (Application) amazon.ca (Application) exchangerate-api (REST-PlainText)	Localtimes (Application) BBC Country Profiles (Application) googleGEO (24) (REST-JSON) earthtools-timezone (19) (REST-XML)
Number of Segments ¹	3	13	5
Number of Components	3	9	5

As part of the experiment, although we showed users the information included in Table 1 and the images of our result pages, we did not require that users make the same mashup applications. Users only needed to keep the input, process and output of the mashup scenario. The number of segments and the style of the display were

left up to them. We also had users record the time consumed in building each mashup application, which included three items to be measured: the time needed to build the SID file (SID_file_time), the time needed to build the MSD file (MSD_file_time) and the time consumed from the start to the mashup application working correctly (Total_time). Because the time users would need to design the style files could differ significantly, we asked the users to record this time separately. Some users had only limited time to devote to the experiment, and so they were able to complete only two of the mashup applications. We arranged for these eight users to complete "City Information" and "Country Information".

Before the experiment, we introduced the M-WA&RS Method, the Path-Reader application and the application for building the description files, which took approximately 40 minutes in total. We also prepared the user-guides for these applications. We did not give any instructions when users began to build the mashup applications. In some instances, the mashup application did not work correctly once a user had completed the two description files. If they were unable to resolve the problem independently, we assisted in diagnosing the issue.

The average time needed to create the information for one segment is listed in Figure 8.1 and is measured as $(\text{Total_time} - \text{MSD_file_time}) / (\text{Number of Segments})$. The average time for creating a MSD file was approximately four minutes. Users 1, 2, 3, 4, 8, 9, 11, 13 and 15 are considered Type 1 users; Users 5, 6, 7, 10, 12 and 14 are Type 2 users. In addition, Users 1, 2 and 15 were proficient in programming. Users 3 and 9 were both proficient in at least one programming language and each had experience building Web applications. Although User 5 and User 6 did not have much programming experience both were familiar with HTML and XML.

In the course of the experiment, User 10's first mashup application did not initially work correctly. Although the user read the error log file several times they were unable to identify the problem. We assisted, and found that he had inadvertently inserted a character in the "startPageURL" field after copying the URL address. This process consumed approximately 15 minutes. Mistakes also occurred when Users 12 and 14 built their "Country Information" mashup application. User 12 selected an incorrect value for the "input" field and User 14 made a mistake when configuring the parameters of the "earthtools-timezone" service. Both users were also unable to resolve the problems themselves. With our help, User 12 took 18 minutes to identify and resolve the problem

and User 14 took 16 minutes. From the time data the users recorded, we found that those users with programming experience consumed more Total_time than would be implied solely from summing the SID_file_time and MSD_file_time. However, all such users were able to resolve problems unassisted. Thus, a more effective algorithm for detecting errors and suggesting solutions is necessary in subsequent research.

All users were able to build the applications successfully. The longest average time to build a segment was less than 25 minutes (Figure 8.1). Considering the fact that the second mashup application scenario used multiple similar segments, we can say that the average time consumed in the building of segments declines with increased familiarity with our approach. Our implementation and tests show that our approach allows users without programming experience to build mashup applications with Web applications and REST Web services, and that they are able to do so quickly. This can make them feel more at ease with building mashup applications.

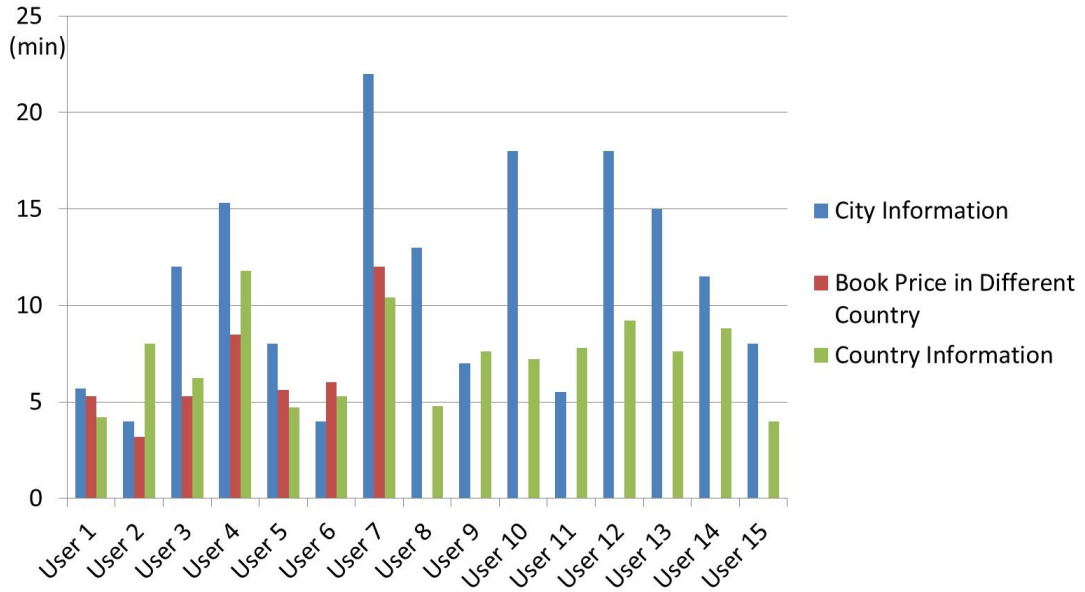


Figure 8.1: Average time of Create Information of One Segment - User Experimental Results

8.1.4 Applicability of the Java Library for End-user Programming

We got the help of six people to build mashup applications in which use the Java-based library we designed. Four of the participants in this experiment are the first

year students of Graduate Schools, and two of them are 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 themselves as expert programmers. None of them has 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 the basic knowledge about HTML language.

They built six mashup applications and reused the information from twenty applications. In addition, some participants did tests with the extracting functions which are not used in their mashup applications. The basic information about these applications are listed in Table 8.4. According to the questionnaires, a problem occurred once in using Visibility Control Method to do dynamic extraction. There is unwanted blank space in the extracted content. This is because that 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 about this fact to decide whether or not we should do change in this method. The users' comments show that functions in the Java-based library we proposed could meet most of their requests in the process of reusing the information of general Web applications in mashup applications' development.

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. For reusing the functionalities of general Web applications, when the submission type of the functionality is Request-Submit, our approach could meet the request, excluding the submission which response the clicking action on Flash. According to the questionnaires, none of the users got problem at this point. For reusing the extracted information of Web applications, our approach could wrap both static and dynamic contents except the case that relative path is used as a parameter in the script codes when extracting dynamic contents. According to the questionnaires, the total extracting times of all uses are forty, in which a user got wrong contents once. A pop-up advertisement which the user did not select is shown in the extracting result. This means we still need to improve the extracting algorithm.

Table 8.4: Basic Information of End-users' Mashup Applications

Functionality of Application	Web Application in Use (http://)	Type of Contents
Get information about the given keyword from two news Web sites, a photo Web site and wikipedia.	1)news.baidu.com 2)news.sina.com.cn 3)www.topit.me 4)en.wikipedia.org/wiki/Main_Page	text, image, flash
Get the first searching result from three different Search Engines about the given keyword.	1)www.baidu.com 2)www.soso.com 3)cn.yahoo.com	text
Get information about a given country from several Web sites.	1)localtimes.info 2)www.countryreports.org 3)travel.state.gov/travel/cis_pa_tw/cis/cis_4965.html	text, image
Get stock information from two Web site about the given keyword (stock name or code).	1)finance.sina.com.cn/stock/ 2)stock.cn.yahoo.com/	text, flash
Get the top movie list from five different Web sites, including the released movies and the coming soon movies.	1)theater.mtime.com/China.Beijing/ 2)ent.sina.com.cn/movie/top10bang.shtml 3)www.juchang.com/movie/ 4)www.moviefone.com/new-movie-releases?sort=release-date 5)movies.yahoo.com/coming-soon/	text, image
Get the titles of the ten songs which are listed in top popular list in certain Web site. Then get the words of these songs according to the titles in another Web site.	1)list.mp3.baidu.com/top/top500.html 2)www.5ilrc.com/	text

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

8.2 Comparison With Existing Approaches

Our approaches are aimed to ease the integration of Web applications and Web services in two different ways, without programming and through end-user programming. As far as our knowledge, Yahoo Pipes and C3W are the existing approaches that can use information from general Web application to build mashups without using any third-part-middleware (for example, third-part widget). However Yahoo Pipes cannot reuse dynamic contents in mashups and it is not easy to retrieve text from Web pages in Yahoo Pipes. C3W is not applicable to Web services, and cannot reuse dynamic contents in mashups. In addition, it is inaccessible to many end-users because of using the special browser and container.

The approaches that wrap information in Web pages to support users to build mashups through programming could be divided into two classes. One class uses the method that wrap the information to generate Web services, for example Pollock and H2W, which takes a great deal of time and skills to create such services. Another class wraps the information into certain data format for reuse, for example Marmite and our approach. However, our approach is the only one that could wrap the dynamic content for reuse. The experiments show that with the Java-based library we supplied, users can build mashup applications more easily with the information from Web applications and Web service APIs with less work by programming.

8.3 Security Factors

The attributes that combining multiple separate Web sources into a new application, reusing the Web contents maximizing and easy to build, have led to the fast growth of mashup applications. At the same time, Web mashups heavily depend on interaction between content from multiple origins and communication with different origins, which

need the support of strong security requirements. Currently, mashup security is based on the security policy of the web: the Same Origin Policy (SOP) (79). The SOP states that scripts from one origin should not be able to access content from other origins. This prevents scripts from stealing data, cookies or login credentials from other sites.

(55) gives the concrete definition of the security requirements for mashup applications, which is consisted of four specific categories: separation, interaction, communication and behavior control. Separation category considers about the needs of mashup components to be separated from each other, which need to ensure that the DOM tree and scripts are separated from each other. Interaction category considers about the requirements of mashup components that need to interaction with other components and the host page. At the same time, interaction category needs to ensure the safety of information, integrity of the components and the mutual authentication of interacting components. Communication category considers about the needs of mashup components to communicate with mashup provider and other parties, for example, cross-domain communication. Behavior Control category considers about the control over specific behavior of mashup components to decide the permission of the specific functionality. These security requirements could be used to examine existing security mechanisms.

Our approach, M-WA&RS, loads components from different origins in Iframes to ensure them be separated by the SOP, which offers the full separation between different origins. We include the scripts and make them to be loaded in the including pages, which could ensure the interaction between the components and the original pages. Script inclusion offers no separation, however, provides full interaction. Our approach does not allow the interaction or communication between the generated components. As the future work, we would like to discuss the security mechanism of our approach to make it support the interaction and communication between the generated components.

Chapter 9

Conclusion

This thesis presented the novel approaches for flexible integration of Web applications and REST Web services. The two kinds of integration methods are the description based approach for integration of without programming and the programmer-oriented approach for building mashup applications through end-user programming with the Java-based library we designed. These approaches are based on the partial information extraction method, hide-and-display method, we proposed.

The current problems are that the huge amount of useful information on the World Wide Web is not always aimed to the needs of the personal mashup developers, and the existing personal oriented data mashup approaches, which are designed to ease the process of building mashup application, have limitations on the using of Web application sources and the integration of Web applications and Web services without programming.

The partial information extraction method applicable to dynamic Web contents we proposed allows users to extract static and dynamic Web contents from general Web applications. The request-submit functionality' simulation of the Web application is supported. The extraction result can be reused for building mashups.

The description based approach for integration Web applications and REST Web services without programming allows the non-experience users to build mashup applications easily and efficiently. We use two description file, SID file and MSD file to record the necessary information for building the mashup applications. The format to record the information of Web applications and REST Web services are similar, and we

use the resembling method to process these two kinds of Web sources, which is easier for users to use.

The programmer-oriented approach for building mashup applications through end-user programming could help the mashup developers to 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 implement the applications with the information from general Web applications easily.

In the thesis, we addressed how to extract both dynamic and static Web contents from general Web applications, and how to build mashup applications without programming with the existing Web applications and REST Web services. As the future work, we would like to support other types of request-submit functionalities of Web applications, for example, the submission through clicking the different area of a flash object. In addition, we would like to extend the usable Web sources of our approach to SOAP type Web services.

Bibliography

- [1] abc News. <http://abcnews.go.com/>. 43
- [2] amazon.ca. <Http://www.amazon.ca/>. 54
- [3] amazon.cn. <Http://www.amazon.cn/>. 54
- [4] amazon.co.jp. <Http://www.amazon.co.jp/>. 54
- [5] amazon.com. <Http://www.amazon.com/>. 54
- [6] BBC Country Profiles. Http://news.bbc.co.uk/2/hi/country_profiles/. 41
- [7] BBC News. <Http://www.bbc.co.uk>. 41
- [8] J. Cao, Y. Riche, S. Wiedenbeck, M. Burnett and V. Grigoreanu. End-User Mashup Programming: Through the Design Lens. In *Proceeding of 28th International Conference on Human Factors in Computing* , pp.1009-1018, 2010. 55
- [9] CBS News. Http:// www.cbsnews.com/ stories /2007/ 08/ 30/ country_ facts/ main3221371.shtml. 41
- [10] Stefano Ceri, Piero Fraternali, and Maristella Matera. Conceptual modeling of data-intensive Web applications. In *IEEE Internet Computing, Volume: 6, Issue:4*, pp.20-30, 2002. 1
- [11] CIA. <Http:// www.cia.gov/ library/ publications/ the-world-factbook/ rank-order/ 2119rank.html>. 70

- [12] Chia-Hui Chang, Mohammed Kayed, Moheb Ramzy Girgis and Khaled Shaalan. A Survey of Web Information Extraction Systems. In *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 10, pp. 1411-1428, 2006. 11
- [13] Chia-Hui Chang, Shao-Chen Lui, and Yen-Chin Wu. Applying Pattern Mining to Web Information Extraction. In *the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 4-15, 2001. 12
- [14] Larry Clarkin and Josh Holmes. Enterprise Mashups. In *MSDN Architecture Journal*. <http://msdn.microsoft.com/en-us/library/bb906060.aspx>
- [15] CNN News. <http://www.cnn.com/>. 6
- [16] Dapp Factory. <http://www.dapper.net/>. 7
- [17] Benslimane Djamel, Schahram Dustdar and Amit Sheth. Services Mashups: The New Generation of Web Applications. In *IEEE Internet Computing*, vol. 12, no. 5. *Institute of Electrical and Electronics Engineers*, pp. 13-15, 2008. 45
- [18] Z.Q. Du, N. Yu, B. Cheng, J.L. Chen. Data Mashup in the Internet of Things. In *Proceeding of the 2011 International Conference on Computer Science and Network Technology*, pp.948-952, 2011. 9
- [19] earthtools-timezone. <http://www.earthtools.org/webservices.htm#timezone>. 73
- [20] R. Ennals and M. Garofalakis. MashMaker: Mashups for the Masses. In *Proceeding of the International Conference on Management of Data(ACM SIGMOD)*, pp.1116-1118, 2007. 9
- [21] exchangerate-api. <http://www.exchangerate-api.com/>. 54
- [22] Roy Thomas Fielding. Architectural Styles and the Design of Network-based Software Architectures. Ph.D. Thesis, University of California, 2000. 1
- [23] Jun Fujima, Aran Lunzer, Kasper Hornbak and Yuzuru Tanaka. C3W: Clipping, Connecting and Cloning for the Web. In *Proceeding of the 13th International Conference on World Wide Web*, pp.444-445, 2004. 8
- [24] GoogleGEO API. <http://maps.google.com/maps/geo>. 73

- [25] Google Maps. <http://maps.google.com/>. 5
- [26] Google Translate API. <http://code.google.com/intl/ja-JP/apis/language/translate/v2>. 52
- [27] Google Weather API. <http://www.google.com/ig/api?weather=cityname>. 52
- [28] Ralph Grishman. Information extraction: Techniques and challenges. In *Lecture Notes in Computer Science Volume 1299*, pp.10-27, 1997. 10
- [29] Suhit Gupta and Gail Kaiser. Extracting Content from Accessible Web Pages. In *Proceedings of the 2005 International Cross-Disciplinary Workshop on Web Accessibility (W4A)*, pp.26-30, 2005. 8
- [30] Hao Han, Yohei Kotake and Takehiro Tokuda. An Efficient Method for Quick Construction of Web Services. In *the 18th European-Japanese Conference on Information Modelling and Knowledge Bases*, pp.180-193, 2008. 56
- [31] Hao Han and Takehiro Tokuda. A Method for Integration of Web Applications Based on Information Extraction. In *the 8th International Conference on Web Engineering*, pp.189-195, 2008. 56
- [32] Hao Han and Takehiro Tokuda. WIKE: A Web Information/Knowledge Extraction System for Web Service Generation. In *the 8th International Conference on Web Engineering*, pp.354-357, 2008. 7
- [33] HtmlUnit. [Http://htmlunit.sourceforge.net](http://htmlunit.sourceforge.net). 22
- [34] G. Huang, Q. Zhao, J. Huang, X. Liu, T. Teng, Y. Zhang and H. Yuan. Towards Service Composition Middleware Embedded in Web Browser. In *Proceeding of the Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC 2009)*, pp.93-100, 2009. 9
- [35] Hoang Pham Huy, Takahiro Kawamura and Tetsuo Hasegawa. How to Make Web Sites Talk Together—Web Service Solution. In *Proceeding of the 14th International Conference on World Wide Web*, pp.850-855, 2005. 56
- [36] IFrame. [Http://en.wikipedia.org/wiki/iframe](http://en.wikipedia.org/wiki/iframe). 32

- [37] iGoogle. <http://www.google.com/ig>. 5
- [38] Infoplease. <http://www.infoplease.com/countries.html>. 12
- [39] JSON-lib Project. <http://json-lib.sourceforge.net/>. 50
- [40] Caitlin Kelleher and Randy Pausch. Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers. In *Journal ACM Computing Surveys (CSUR) Volume 37 Issue 2*, pp.83-137, 2005. 30
- [41] Andrew J. Ko and Brad A. Myers. Human Factors Affecting Dependability in End-User Programming. In *the Proceeding WEUSE I Proceedings of the first workshop on End-user software engineering*, pp.1-4, 2005. 30
- [42] Andrew J. Ko, Brad A. Myers, and Htet Htet Aung. Six Learning Barriers in End-User Programming Systems. In *IEEE Symposium on Visual Languages - Human Centric Computing (VLHCC'04)*, pp.199-206, 2004. 30
- [43] Yoshiyuki Koseki and Atsushi Sugiura. Internet scrapbook: Automating Web Browsing Tasks by Demonstration. In *ACM Symposium on User Interface Software and Technology*, pp.9-18, 1998. 8
- [44] R. Latih, A. M. Patel, A. M. Zin, T. Yiqi, S. H. Muhammad. Whip: A Framework for Mashup Development with Block-based Development Approach. In *Proceeding of the 2011 International Conference on Electrical Engineering and Informatics*, pp.1-6, 2011. 8
- [45] X. Liu, Y. Hui, W. Sun and H. Liang. Towards Service Composition Based on Mashup. In *Proceeding of the 2007 IEEE Congress on Services (SERVICES 2007)*, pp.332-339, 2007. 9
- [46] Localtimes.info. <http://localtimes.info/>. 8
- [47] YiHsuan Lu, Yoojin Hong, Jinesh Varia and Dongwon Lee. Pollock: Automatic Generation of Virtual Web Services from Web Sites. In *the Proceeding of the 2005 ACM symposium on Applied computing*, pp.1650-1655, 2005. 56
- [48] mashArt. <http://mashart.org/pages/login.jspx>. 9

- [49] Michael W. Geoghegan and Klass, Dan. Podcast Solutions: The Complete Guide to Podcasting. 2005. 5
- [50] Jussi Myllymaki. Effective Web Data Extraction with Standard XML Technologies. In *Proceeding of the 10th international conference on WWW*, pp.689-696, 2001. 8
- [51] B. Nardi. A Small Matter of Programming: Perspectives on End-User Computing. In *MIT Press, Cambridge, MA*, pp.5-6, 1993. 55
- [52] ProgrammableWeb. <http://www.programmableweb.com/>. 5
- [53] reuters news. <http://www.reuters.com/>. 6
- [54] RSS. <http://en.wikipedia.org/wiki/RSS>. 7
- [55] Philippe De Ryck, Maarten Decat, Lieven Desmet, Frank Piessens, and Wouter Joosen. Security of Web Mashups: a Survey. In *the 15th Nordic Conference in Secure IT Systems* , pp.223-238, 2010. 79
- [56] T. Nestler, M. Feldmann, G. Hbsch, A. PreuBner and U. Jugel. The ServFace Builder - A WYSIWYG Approach for Building Service-Based Applications. In *Proceeding of the 10th International Conference on Web Engineering*, pp.498-501, 2010. 9
- [57] SJR. <Http://www.scimagojr.com/countrysearch.php>. 70
- [58] K. T. Stolee, S. Elbaum and A. Sarma. End-User Programmers and their Communities: An Artifact-based Analysis. In *the Proceeding 2011 International Symposium on Empirical Software Engineering and Measurement*, pp.147-156, 2011. 55
- [59] Tetsuya Suzuki and Takehiro Tokuda. Path Set Operations for Clipping of Parts of Web Pages and Information Extraction from Web Pages. In *the Proceeding of the 15th International Conference on Software Engineering and Knowledge Engineering*, pp.547-554, 2003. 8

- [60] Michiaki Tatsubori and Kenichi Takahashi. Decomposition and Abstraction of Web Applications for Web Service Extraction and Composition. In *the Proceedings of the 2006 International Conference on Web Services*, pp.859-868, 2006. 56
- [61] Travel News. <http://www.breakingtravelnews.com/>. 64
- [62] Trippermap.com. <Http://www.trippermap.com>. 41
- [63] W3C Web Services Definition. <http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/>. 44
- [64] Weatherbonk.com. <Http://www.weatherbonk.com>. 41
- [65] Web Mashup. [http://en.wikipedia.org/wiki/Mashup_\(web_application_hybrid\)](http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)). 5
- [66] http://en.wikipedia.org/wiki/Information_extraction. 10
- [67] Wikipedia. Http://en.wikipedia.org/wiki/Main_Page. 70
- [68] Jeffrey Wong and Jason I. Hong. Making mashups with marmite: Towards end-user programming for the Web. In *the Proceeding of the SIGCHI conference on Human factors in computing systems*, pp.1435-1444, 2007. 56
- [69] World Travel Guide. <http://www.worldtravelguide.net/>. 62
- [70] XPath. <http://www.w3.org/TR/xpath20/>. 12
- [71] X-Rates. <Http://www.x-rates.com/calculator.html> 70
- [72] Yahoo Finance. <http://finance.yahoo.com>. 70
- [73] Yahoo Maps. <http://maps.yahoo.com/>. 5
- [74] Yahoo News. <http://news.yahoo.com/>. 71
- [75] Yahoo Pipes. <http://pipes.yahoo.com/pipes/>. 8
- [76] Yahoo Travel. <http://travel.yahoo.com/>. 63

- [77] Jin Yu, Boualem Benatallah, Fabio Casati, Florian Daniel. A Framework for Rapid Integration of Presentation Components. In *the proceeding of 16th International World Wide Web Conference*, pp.923-932, 2007. 8
- [78] Jin Yu, Boualem Benatallah, Fabio Casati, Florian Daniel, Maristella Matera and Regis Saint-Paul. Mixup: A Development and Runtime Environment for Integration at the Presentation Layer. In *the 7th International Conference on Web Engineering*, pp.479-484, 2007. 8
- [79] Michal Zalewski. Browser Security Handbook. In *[http:// code.google.com/ p/ browsersec/ wiki/ Part2](http://code.google.com/p/browsersec/wiki/Part2)*. 79
- [80] N. Zang and M. B. Rosson. What's in a mashup? And why? Studying the perceptions of web-active end users. In *Proceedings 2008 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp.33-38, 2008. 55

Appendix A

Mashup Application Template

```
1 <head>
2 <meta http-equiv="Content-Type" content="text/html;␣
   charset=UTF-8">
3 </head>
4 <script language="JavaScript">
5 var GscenarioDoc = new ActiveXObject("MSXML2.DOMDocument
   .3.0");
6 var GcomponentDoc = new ActiveXObject("MSXML2.DOMDocument
   .3.0");
7 var Gitems;
8 var GcomponentItems;
9 var Gexecute = new Array();
10 var Gdisplay = new Array();
11 var GstyleFile = new Array();
12 var GvalofOUTPUT = new Array();
13 var GiFrameNum = 0;
14 var Giframeids = new Array();
15 var Giframehide="yes";
16 var obj;
17 var GoffsetX;
18 var GoffsetY;
19 function init()
```

```

20 {
21     var strMSDFile = "";
22     var strSIDFile = "";
23     //Get the location and name of the MSD file from
        previous page.
24     strMSDFile = document.cookie.split(";")[0];
25     GscenarioDoc.load(strMSDFile);
26     Gitems = GscenarioDoc.selectNodes("/channel/Component")
        ;
27     GiFrameNum = Gitems.length;
28     GetScenario();
29     iFrameCreation();
30     strSIDFile = getNode(GscenarioDoc, "/channel/base-file"
        );
31     GcomponentDoc.load(strSIDFile);
32     GcomponentItems = GcomponentDoc.selectNodes("/channel/
        segment");
33 }
34 function getNode(doc, xpath) {
35     var retval = "";
36     var value = doc.selectSingleNode(xpath);
37     if (value) retval = value.text;
38     return retval;
39 }
40 function seachClear()
41 {
42     document.form_search.keyword.value = "";
43 }
44 //Get the number of components that will be shown.
45 function GetScenario()
46 {
47     Gdisplay = new Array(Gitems.length+1);
48     for (var i=0; i < Gitems.length; i++){
49         Gdisplay[i+1] = getNode(GscenarioDoc, "/channel/
            Component[" + i + "]/display");

```

```

50     }
51 }
52 //Create iframes according to the number of components.
53 function iFrameCreation()
54 {
55     if (document.readyState != "complete") return ;
56     Giframeids = new Array(GiFrameNum);
57     for (var i=0; i < GiFrameNum; i++ ){
58         var iFrameNM = "iframeRst" + (i+1);
59         var iFrameTop = 200 + Math.floor(i/2)*800;
60         var iFrameLeft = 10 + (i%2)*800;
61         var DivID = "divID" + (i+1);
62         var span = document.createElement('span');
63         span.setAttribute('id','iframe' + i);
64         span.innerHTML = '<div id=' + DivID + ' style="
            position:absolute; top:' + iFrameTop + 'px; left
            : ' + iFrameLeft + 'px; padding: 5px; border:none
            ; text-align:center; cursor: move;"><iframe id=
            ' + iFrameNM + ' width=800 height=800
            frameborder=0></iframe></DIV>';
65         document.body.appendChild(span);
66         Giframeids[i] = iFrameNM;
67     }
68 }
69 //Start the generation of components when the submitting
button is pressed.
70 function searchResult()
71 {
72     if(document.form_search.keyword.value.length == 0) {
73         return false;
74     }
75     var flag = 0;
76     if (document.readyState != "complete") return ;
77     var result ;
78     var tmpResut = new Array (2);

```

```

79  GvalofOUTPUT = new Array(GcomponentItems.length+1);
80  Gexecute = new Array(Gitems.length+1);
81  GStyleFile = new Array(Gitems.length+1);
82  for(var i=0; i < Gitems.length; i++ ){
83      Gexecute[i+1] = getNode(GscenarioDoc, "/channel/
      Component[" + i + "]/execute");
84      GStyleFile[i+1] = getNode(GscenarioDoc, "/channel/
      Component[" + i + "]/styleFile");
85      //Check the display is combined by several segments
      or just one segment.
86      if(Gexecute[i+1].indexOf(":")>0){
87          var subExecute = Gexecute[i+1].split(":");
88          var strXMLFiles = new Array(subExecute.length);
89          for(var j=0; j<subExecute.length; j++){
90              tmpResult = GetSegment(subExecute[j], GStyleFile[
              i+1]);
91              strXMLFiles[j] = tmpResult[0];
92              var tmpNM = GetTargetNM(subExecute[j]);
93              GvalofOUTPUT[tmpNM] = tmpResult[1];
94          }
95          result = CombineComponets(strXMLFiles,GStyleFile[i
              +1],"/channel/item");
96      }else{
97          tmpResult = GetSegment(Gexecute[i+1], GStyleFile[i
              +1]);
98          result = tmpResult[0];
99          var tmpNM = GetTargetNM(Gexecute[i+1]);
100          GvalofOUTPUT[tmpNM] = tmpResult[1];
101      }
102      if(Gdisplay[i+1] == "true"){
103          displayResult(i, result);
104      }
105  }
106  setTimeout('dyniframesize()',30000);
107  for (var i=0; i < 50000; i++ ){

```

```

108     if (IsEveryIframeShown())
109         break;
110     }
111     dyniframesize();
112     MouseShori();
113 }
114 //Generate a segment.
115 function GetSegment(strExecute, strStyleFile){
116     var strRESTRequest = "";
117     var strReturnNodeName = "";
118     var strExtractionType = "";
119     var strSegmentResult = new Array(2);
120     var arrExtractcontent = new Array();
121     var arrContentType = new Array();
122     var targetID = GetTargetNM(strExecute)-1;
123     var strTargetType = getNode(GcomponentDoc, "/channel/
        segment[" + targetID + "]/type");
124     var strStartPageURL = getNode(GcomponentDoc, "/channel/
        segment[" + targetID + "]/startPageURL");
125     //Get the Content Array and the Content Type Array
126     var strExtractcontents = GcomponentDoc.selectNodes("/
        channel/segment[" + targetID + "]/extractionContents
        /content");
127     arrExtractcontent = new Array(strExtractcontents.length
        );
128     arrContentType = new Array(strExtractcontents.length);
129     var tmpContentTypeCheck = getNode(GcomponentDoc, "/
        channel/segment[" + targetID + "]/contentsType/
        contentType");
130     var strContentsType = GcomponentDoc.selectNodes("/
        channel/segment[" + targetID + "]/contentsType/
        contentType");
131     for (var i=0; i < strExtractcontents.length; i++){
132         arrExtractcontent[i] = getNode(GcomponentDoc, "/
            channel/segment[" + targetID + "]/

```

```

        extractionContents/content[" + i + "]);
133     arrContentType[i] = getNode(GcomponentDoc, "/channel/
        segment[" + targetID + "]/contentsType/contentType
        [" + i + "]);
134 }
135 var strReturnType = getNode(GcomponentDoc, "/channel/
        segment[" + targetID + "]/returnType"); var
        strOutput = getNode(GcomponentDoc, "/channel/segment["
        " + targetID + "]/output");
136 strRESTRequest = strStartPageURL;
137 if (strTargetType == "REST"){
138     var strParameters = GcomponentDoc.selectNodes("/
        channel/segment[" + targetID + "]/parameters/
        parameter");
139     var arrParameter = new Array(strParameters.length);
140     var strXSLforShowInHTML = "";
141     for (var i=0; i < strParameters.length; i++) {
142         arrParameter[i] = getNode(GcomponentDoc, "/channel/
            segment[" + targetID + "]/parameters/parameter["
            + i + "]);
143         if (arrParameter[i].indexOf('\\"')>=0){
144             arrParameter[i] = arrParameter[i].substring(1,
                arrParameter[i].length-1);
            }
            else if(arrParameter[i] == "userInput"){
145                 arrParameter[i] = document.form_search.keyword.
                    value;
146             } else {
147                 var strOutputTarget = arrParameter[i].substring
                    (0, arrParameter[i].indexOf("."));
148                 var tmpNM = GetTargetNM(strOutputTarget);
149                 arrParameter[i] =GvalofOUTPUT[tmpNM];
150             }
151             strRESTRequest = strRESTRequest + arrParameter[i];
152         }
153     var strServiceResultType = getNode(GcomponentDoc, "/"

```

```

        channel/segment[" + targetID + "]/
        serviceResultType");
154     if (strServiceResultType == "PlainText"){
155         strReturnNodeName = getNode(GcomponentDoc,"/channel
            /segment[" + targetID + "]/returnNodeName");
156     }else {
157         strXSLforShowInHTML = getNode(GcomponentDoc,"/
            channel/segment[" + targetID + "]/
            styleFileForShowInHTML");
158     }
159     strSegmentResult = document.App.getRESTResult(
        document.App.getCodeBase() ,strRESTRequest,
        strServiceResultType, strXSLforShowInHTML,
        strReturnType, strReturnNodeName,
        arrExtractcontent, strOutput, strStyleFile);
160 }else if(strTargetType == "Application"){
161     var strInputType = getNode(GcomponentDoc, "/channel/
        segment[" + targetID + "]/inputType");
162     if (strInputType == "none"){
163         var strInputArea = "";
164         var strInput = "";
165     }else{
166         var strInputArea = getNode(GcomponentDoc, "/channel
            /segment[" + targetID + "]/inputArea");
167         var strInput = getNode(GcomponentDoc, "/channel/
            segment[" + targetID + "]/input");
168     }
169     if (strInput == "userInput"){
170         strInput = document.form_search.keyword.value;
171     }else if (strInput != ""){
172         var strInputTarget = strInput.substring(0,
            arrParameter[i].indexOf(".")-1);
173         var tmpNM = GetTargetNM(strInputTarget);
174         strInput = GvalofOUTPUT[tmpNM];
175     }

```

```

176      //Call the Applet to GET the extracting result.
177      strSegmentResult = document.App.getApplicationResult(
          document.App.getCodeBase() , strInput ,
          strStartPageURL , strInputType ,arrExtractcontent ,
          strInputArea ,arrContentType , strOutput ,
          strReturnType , strStyleFile);
178  }
179  return strSegmentResult;
180  }
181  //Display the components in iFrames.
182  function displayResult(intFrameNM, resultToDisplay){
183      var iframeNM = "iframeRst" + (intFrameNM+1);
184      document.getElementById(iframeNM).contentWindow.
          document.open();
185      document.getElementById(iframeNM).contentWindow.
          document.clear();
186      document.getElementById(iframeNM).contentWindow.
          document.write(resultToDisplay);
187      document.frames[iframeNM].location.reload(true);
188  }
189  //Combine the extracting results--XML files.
190  function CombineComponets(strXMLFile, strStyleFile,
          strMergeXpath){
191      var strCombineResult = document.App.CombineXMLFiles(
          document.App.getCodeBase() , strXMLFile ,
          strStyleFile , strMergeXpath);
192      return strCombineResult;
193  }
194  function GetTargetNM(strTargetID){
195      var targetIDNM = strTargetID.substring(7, strTargetID.
          length);
196      return targetIDNM;
197  }
198  function IsEveryIframeShown(){
199      for (var i=0; i < Gitems.length; i++ ){

```

```

200     var iframeNM = "iframeRst" + (i+1);
201     if (document.getElementById(iframeNM).readyState != "
        complete"){
202         return false;
203     }
204 }
205 return true;
206 }
207 function MouseShori() {
208     if (document.all) {
209         objlist = document.all.tags("div");
210     } else if (document.getElementsByTagName) {
211         objlist = document.getElementsByTagName("div");
212     } else {
213         return;
214     }
215     for (i = 0; i < objlist.length; i++) {
216         objlist[i].onmousedown = onMouseDown;
217     }
218     document.onmousemove = onMouseMove;
219     document.onmouseup = onMouseUp;
220 }
221 function onMouseDown(e) {
222     obj = this;
223     if (document.all) {
224         offsetX = event.offsetX + 2;
225         offsetY = event.offsetY + 2;
226     } else if (obj.getElementsByTagName) {
227         offsetX = e.pageX - parseInt(obj.style.left);
228         offsetY = e.pageY - parseInt(obj.style.top);
229     }
230     return false;
231 }
232 function onMouseMove(e) {
233     if (!obj) {

```

```

234     return true;
235 }
236 if (document.all) {
237     obj.style.left = event.clientX - offsetX + document.
        body.scrollLeft;
238     obj.style.top = event.clientY - offsetY + document.
        body.scrollTop;
239 } else if (obj.getElementsByTagName) {
240     obj.style.left = e.pageX - offsetX;
241     obj.style.top = e.pageY - offsetY;
242 }
243 return false;
244 }
245 function onMouseUp(e) {
246     obj = null;
247 }
248 function dyniframesize(){
249     var dyniframe=new Array()
250     var tempDivTop = 0;
251     var tempDivTop1 = 0;
252     var tempDivTop2 = 0;
253     var tempDivLeft = 0;
254     for (var i=0; i<Giframeids.length; i++){
255         if (document.getElementById){
256             dyniframe[dyniframe.length] = document.
                getElementById(Giframeids[i]);
257             var DivID = "divID" + (i+1);
258             document.getElementById(DivID).style.left = (
                parseInt(tempDivLeft) + 10);
259             document.getElementById(DivID).style.top = (
                parseInt(tempDivTop) + 150);
260             if (dyniframe[i] && !window.opera){
261                 dyniframe[i].style.display="block"
262                 if (dyniframe[i].contentDocument && dyniframe[i].
                    contentDocument.body.offsetHeight) {

```

```

        dyniframe[i].height = dyniframe[i].
        contentDocument.body.offsetHeight + 30;
263     dyniframe[i].width = dyniframe[i].
        contentDocument.body.offsetWidth + 30;
264     }else if (dyniframe[i].Document && dyniframe[i].
        Document.body.scrollHeight) {
265     dyniframe[i].height = dyniframe[i].Document.
        body.scrollHeight + 30;
266     dyniframe[i].width = dyniframe[i].Document.body
        .scrollWidth + 30;
267     }
268     }
269     if (i%2 == 0 ) {
270     tempDivLeft = parseInt(dyniframe[i].width) +
        parseInt(tempDivLeft);
271     tempDivTop1 = dyniframe[i].height
272     } else {
273     tempDivLeft = 0;
274     tempDivTop2 = dyniframe[i].height
275     if (parseInt(tempDivTop1) > parseInt(tempDivTop2)
        ) {
276     tempDivTop = parseInt(tempDivTop) + parseInt(
        tempDivTop1);
277     } else {
278     tempDivTop = parseInt(tempDivTop) + parseInt(
        tempDivTop2);
279     }
280     }
281     }
282     }
283     }
284     </script>
285     <body onload="init()">
286     <table border=0 width="100%" align="center">
287     <tr>

```

```

288     <td width="100%" valign="top" align="center">
289         <form name="form_search" method="post" action="/
290             newsSearch">
291             <table border="1" align="center" cellspacing="0"
292                 >
293                 <tr>
294                     <td height="30" align="left">About</td>
295                     <td height="30" align="right">
296                         <input type="text" name="keyword" size="14"
297                             value= " " onkeydown="if(event.keyCode==13)
298                                 {seachResult();return false}">
299                     </td>
300                 </tr>
301             </table>
302             <table border="0" align="center">
303                 <tr align="left">
304                     <td align="left">
305                         <input type="button" name="search" value="
306                             search" onclick="searchResult()">
307                     </td>
308                     <td align="right">
309                         <input type="button" name="Clear" value="
310                             Clear" onclick="seachClear()">
311                     </td>
312                 </tr>
313             </table>
314         </form>
315         <hr/>
316     </td>
317 </tr>
318 </table>
319 <applet code="JJApplet.class" archive="App.jar,commons-
320     codec-1.3.jar,commons-collections-3.2.1.jar,commons-
321     httpclient-3.1.jar,commons-io-1.4.jar,commons-lang
322     -2.4.jar,commons-logging-1.1.1.jar,cssparser-0.9.5.jar

```

```
    ,filterbuilder.jar,htmllexer.jar,htmlparser.jar,
    htmlunit-2.5.jar,htmlunit-core-js-2.5.jar,nekohtml
    -1.9.12.jar,sac-1.3.jar,serializer-2.7.1.jar,
    sitecapturer.jar,thumbelina.jar,xalan-2.7.1.jar,
    xercesImpl-2.8.1.jar,xml-apis-1.3.04.jar,json-lib-2.4-
    jdk15.jar,ezmorph-1.0.6.jar,commons-beanutils.jar,␣xom
    -1.2.6.jar" name="App" width="0" height="0">
314 </applet>
315 </body>
316 </html>
```

Appendix B

MSD and SID File of Mashup Application Example 3

MSD File of Mashup Application Example 3

```
317 <channel>
318   <base-file>
319     C:\temp\Demo1SegmentInformation.xml
320   </base-file>
321   <Component>
322     <id> Component1 </id>
323     <execute> segment1 </execute>
324     <display> true </display>
325     <styleFile> </styleFile>
326   </Component>
327   <Component>
328     <id>Component2 </id>
329     <execute> segment2 </execute>
330     <display> true </display>
331     <styleFile> </styleFile>
332   </Component>
333   <Component>
334     <id>Component3 </id>
335     <execute> segment3 </execute>
```

```
336     <display> true </display>
337     <styleFile> </styleFile>
338 </Component>
339 </channel>
```

SID File of Mashup Application Example 3

```
341 <channel>
342   <segment>
343     <id> segment1 </id>
344     <type> Application </type>
345     <startPageURL> http://www.reuters.com/ </startPageURL>
346   <inputType>   InputBox </inputType>
347   <inputArea>
348     /HTML/BODY/DIV[2](header,null,null)/DIV[5](search,
349       null,panel)/DIV(searchbox,null,null)/FORM(
350       searchForm,searchbox,null)
351   </inputArea>
352   <input>   userInput </input>
353   <extractionContents>
354     <content>
355       /HTML/BODY/DIV[2](content,null,null)/DIV[4](null,
356         null,section gridlined8)/DIV(null,null,
357         sectionContent)/DIV[2](null,null,
358         sectionColumns)/DIV(null,null,column1
359         gridPanel grid8)/DIV[3](searchCenter,null,null
360         )/DIV(null,null,module)/DIV(null,null,
361         moduleBody)
362     </content>
363   </extractionContents>
364   <contentsType>
365     <contentType>
```

```

358         dynamicContent
359     </contentType>
360 </contentsType>
361 <returnType> HTML </returnType>
362 <output>
363     /HTML/BODY/DIV[2](content,null,null)/DIV[4](null,
        null,section gridlined8)/DIV(null,null,
        sectionContent)/DIV[2](null,null,sectionColumns)
        /DIV(null,null,column1 gridPanel grid8)/DIV[3](
        searchCenter,null,null)/DIV(null,null,module)/
        DIV(null,null,moduleBody)/SPAN(null,null,
        searchResult)/UL(null,null,null)/LI(null,null,
        searchHeadline)/A(null,null,null)/SPAN(null,null
        ,null)
364 </output>
365 </segment>
366 <segment>
367     <id> segment2 </id>
368     <type> REST </type>
369     <startPageURL>
370         https://www.googleapis.com/language/translate/v2?
            key=AIzaSyDCCg1yPIdIn44fcjq7z51T_yfiUEGIkYk%26
            source=en%26target=ja%26q=
371     </startPageURL>
372     <parameters>
373         <parameter>
374             segment1.ouput
375         </parameter>
376     </parameters>
377     <serviceResultType> JSON </serviceResultType>
378     <styleFileForShowInHTML> C:\temp\
        googleTranslateLayout.xml </styleFileForShowInHTML
        >
379     <returnType> HTML </returnType>
380     <returnNodeName> </returnNodeName>

```

```

381     <extractionContents>
382         <content>
383             /HTML/BODY/TABLE(null,null,null)/TR(null,null,
384                 null)/TD(null,null,null)/TABLE(null,null,null)
385         </content>
386     </extractionContents>
387     <contentsType>
388         <contentType>
389             dynamicContent
390         </contentType>
391     </contentsType>
392     <output> null </output>
393 </segment>
394 <segment>
395     <id> segment3 </id>
396     <type> REST </type>
397     <startPageURL> http://www.google.com/ig/api?weather=
398         </startPageURL>
399     <parameters>
400         <parameter>
401             userInput
402         </parameter>
403     </parameters>
404     <serviceResultType> XML </serviceResultType>
405     <styleFileForShowInHTML> C:\temp\Layout.xsl </
406         styleFileForShowInHTML>
407     <returnType> HTML </returnType>
408     <returnNodeName> </returnNodeName>
409     <extractionContents>
410         <content>
411             /HTML/BODY/TABLE(null,null,null)/TR(null,null,
412                 null)/TD(null,null,null)/TABLE(null,null,null)
413         </content>
414     </extractionContents>
415     <contentsType>

```

```
412         <contentType>
413             dynamicContent
414         </contentType>
415     </contentsType>
416     <output> null </output>
417 </segment>
418 </channel>
```

Appendix C

Part of the MSD and SID File Used in Mashup Application Example 4

Part of the MSD File Used in Mashup Application Example 4

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <channel>
3     <base_file>
4         c:\temp\SegmentInformation.xml
5     </base_file>
6     <Component>
7         <id>Component1 </id>
8         <execute> segment4 </execute>
9         <display> true </display>
10        <styleFile> </styleFile>
11    </Component>
12    <Component>
13        <id>Component2 </id>
14        <execute> segment5:segment10 </execute>
15        <display> true </display>
16        <styleFile> c:\temp\PriceLayout.xsl </styleFile>
17    </Component>
```

```

18     ...
19 </channel>

```

Part of the SID File Used in Mashup Application Example 4

```

420 <channel>
421   <segment>
422     <id> segment1 </id>
423     <type> Application </type>
424     <startPageURL> http://www.amazon.com/ </
      startPageURL>
425     <inputType> InputBox </inputType>
426     <inputArea>
427       /HTML/BODY/DIV[2](navbar,null,navbar)/TABLE(
        null,null,null)/
428       TR[3](null,null,navShopSearch)/TD[2](null,
        null,null)/
429       TABLE(null,null,null)/TR(null,null,null)/TD(
        null,null,null)/
430       FORM(null,site-search,null)
431     </inputArea>
432     <input> userInput </input>
433     <extractionContents>
434       <content>
435         /HTML/BODY/TABLE(searchTemplate,null,
          searchTemplate listLayout)/...
436       </content>
437       <content>
438         /HTML/BODY/TABLE(searchTemplate,null,
          searchTemplate listLayout)/...
439       </content>
440     </extractionContents>
441     <contentsType>

```

```

442         <contentType> dynamicContent </contentType>
443         <contentType> dynamicContent </contentType>
444     </contentsType>
445     <returnType> HTML </returnType>
446     <output> null </output>
447     <memo> The cover, title, author and publish
        information of book. </memo>
448 </segment>
449 <segment>
450     <id> segment3 </id>
451     <type> Application </type>
452     <startPageURL> http://www.amazon.com/ </
        startPageURL>
453     <inputType> InputBox </inputType>
454     <inputArea>
455         /HTML/BODY/DIV[2](navbar,null,navbar)/TABLE(
            null,null,null)/...
456     </inputArea>
457     <input> userInput </input>
458     <extractionContents>
459         <content>
460             /HTML/BODY/TABLE(searchTemplate,null,
                searchTemplatelistLayout)/...
461         </content>
462     </extractionContents>
463     <contentsType>
464         <contentType> staticContent </contentType>
465         <returnNodeName> Price </returnNodeName>
466         <extractionType> single text </extractionType>
467     </contentsType>
468     <returnType> XML </returnType>
469     <output>
470         /HTML/BODY/TABLE(searchTemplate,null,
            searchTemplate listLayout)/...

```

```

471         </output>
472         <memo> The price of book in amazon.com and output
           the same information. </memo>
473     </segment>
474 <segment>
475     <id> segment10 </id>
476     <type> REST </type>
477     <startPageURL> http://www.exchangerate-api.com/
           JPY/USD/ </startPageURL>
478     <parameters>
479         <parameter> segment5.ouput </parameter>
480         <parameter> "?k=m6eL6-M9zDu-9jE6k" </
           parameter>
481     </parameters>
482     <styleFileForShowInHTML> null </
           styleFileForShowInHTML>
483     <serviceResultType> PlainText </serviceResultType>
           >
484     <returnType> XML </returnType>
485     <returnNodeName> PriceInUSDollar </returnNodeName>
           >
486     <extractionContents>
487         <content> all </content>
488     </extractionContents>
489     <output> null </output>
490     <memo> Convert the price from JPY to USD. </memo>
491 </segment>
492     ...
493 </channel>

```