A method for facilitating end-user mashup based on description

Junxia Guo*

College of Information Science and Technology,
Beijing University of Chemical Technology,
4# 15 Beisanhuan East Road, Chaoyang District,
Beijing, 100029, China
and
Department of Computer Science,
Tokyo Institute of Technology,
Ookayama 2-12-1-W8-71, Meguro, Tokyo 152-8552, Japan
E-mail: gixia@mail.buct.edu.cn
E-mail: guo@tt.cs.titech.ac.jp
*Corresponding author

Hao Han

Department of Information Science, Faculty of Science,
Kanagawa University,
Hiratsuka, Kanagawa 259-1293, Japan
E-mail: han@kanagawa-u.ac.jp

Abstract: The development of mashup technology has allowed people to more efficiently create web applications with existing information drawn from the web. However, not every mashup developer can use any given web source in a mashup. There are two technological problems. First, it is not always possible to reuse a given source. There are web applications and web services, two kinds of sources. The traditional way of reusing web services is to write a program. Relevant and applicable approaches are needed in order to reuse web applications. Second, it must be possible for users without programming knowledge to reuse an existing web source. In this paper, we propose an effective approach that allows end-users to build mashup applications with almost any part(s) of web applications and REST web services without programming. Our approach supports the more flexible transfer of information between components, which can be achieved via a web application or REST web service.

Keywords: end-user mashup; web applications; REST web services; information transfer.


Biographical notes: Junxia Guo is currently a Lecturer at College of Information Science and Technology, Beijing University of Chemical Technology, China. Her current research interests include web information extraction, web service, mashup and web mining. She received her PhD in
1 Introduction

Mashup technology has become central to users without programming experience. Unfortunately, most current web sources are not aimed toward these users. The web sources that people use to build mashup applications mainly include web applications and web services. Integrating published web services through end-user programming is a flexible and widely used method of building a mashup application. However, such integration is limited based on the information resource because many websites publish information in a web application format rather than in a format compatible with web services. For example, BBC Country Profiles provide information on over 200 countries and regions. This data cannot easily be reused in other mashup applications, however, because it is not published as a web service API. In addition, research on end-user programming has found that programming is difficult for novices for a number of reasons (Ko et al., 2004). Our previous work (Guo et al., 2010a, 2010b) allowed users to build mashup applications with almost any part(s) of general web applications, with no programming knowledge. Due to the format of the information, however, use of the above approaches remains limited. If we use the services to reformat information, the above approaches will be more widely applicable. At the same time, if users can freely access any part(s) of the information that service APIs return without having to undertake any programming, building a mashup application will be an easier and more efficient process.

The purpose of this paper is to present an approach that allows users to build mashup applications conveniently, with information taken from existing applications and REST web services and without requiring any original or additional programming. In addition, this approach supports more flexible information transfers between different web resources, which can increase the functionality of mashup applications. Web services that are used in mashup applications are mainly sorted into two styles, SOAP and REST. According to the ProgrammableWeb, 68% of the protocol usages by APIs are REST while 23% are SOAP. Thus, we have made web applications and REST services the web sources of our approach.

We use two description files to record information on the basic segments; these files are then used to generate the mashup components and the mashup scenario. Our approach can generate mashup components from both general web applications and REST services. According to the mashup scenario that end-users have designed, we first generate basic segments using the information from web applications and REST services. For the components that use information from web applications, we use our partial information
A method for facilitating end-user mashup based on description  

A method for facilitating end-user mashup based on description extraction and functionality emulation methods. For the components that use information from REST services, we convert the result of the services to HTML format by using style files; we use the same technique to extract the necessary information before displaying the components. Users can merge several basic segments (excluding segments generated by dynamic web content) into one component or choose to not display certain basic segments. Here we use static web content to represent the information shown on the web page, which can be found directly in the HTML source file – such as the text between the HTML code’s span tags. Dynamic web content represents the information that is shown on the web page but cannot be found in the HTML source file – such as the HTML code that is dynamically generated by client-side scripts.

The rest of this paper is structured as follows. Section 2 provides an overview of related works. In Section 3, we introduce an example built using our approach. In Section 4, we explain the details of our approach. In Section 5, we evaluate our approach. Finally, we conclude and discuss avenues for future work.

2 Related works

Although we can access a great deal of web sources to build mashup applications, most are not aimed at users with no programming background. Hence, many research works and industrial tools have appeared that are designed to ease the creating of mashup applications. These are sorted into two main styles. One is based on the spreadsheet, such as spreadsheet-based web mashups (Kongdenfa et al., 2009), vegemite (Lin et al., 2009), marmite (Wong and Hong, 2007) and so on. The other is based on the web browser and is the approach we have used here. Following are some well-known approaches that can support the building of mashup applications either automatically or semi-automatically. We study these methods from two points of view: What kind of web sources can this method use? And, is there any limitation on its usage? The process of integration can occur either automatically or not.

Whip (Latih et al., 2011) provides an interface for the building of mashup applications that uses predefined widgets. Users with programming skills can build new widgets. Its data sources mainly focus on web services. The presenting of mashup applications can be accomplished automatically.

Internet scrapbook (Sugiura and Koseki, 1998) is a system that allows users to create a personal web page by clipping parts of other 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 a HTML document and users cannot alter the layout. In addition, it cannot reuse dynamic content.

C3W (Fujima et al., 2004) can reuse text and functional elements 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 users because of its need for a specific browser to clip web content and a unique container, DerivationPad, for the clipped content.

Mixup (Yu et al., 2007a, 2007b) can quickly build complex user interfaces using the web services available in the system. The web sources it can use are limited to those it has registered. The integration process is semi-automatic.

Yahoo Pipes provides a number of user-configurable modules for use in building mashup applications. Users can use web feeds, services and retrieved text from web
pages. Users can add certain kinds of web sources but cannot do so in a completely free fashion. The integration process is automatic. However, it does not deal with dynamic content and it is not easy to retrieve text from web pages.

‘Data Mashup Platform’ (Du et al., 2011) can use four kinds of data sources (database, restful sensor resource, web service and local file) and predefined mashup logics to generate ‘mashup services’. Users can configure the resources and create the mashup services through a data mashup editor. The web sources it can use do not include information from web applications. This approach does not provide any user interface for designing mashup applications.

MashMaker (Ennals and Garofalakis, 2007) uses the widgets created by third-parties to build mashup applications. It works as a web browser plug-in. The resources it can use are limited to the widgets library, but the widgets can be generated from web applications and services. Although they allow users to create their own widgets, many users do not have the knowledge required to do so, which leads us to consider the integration process in this case to be semi-automatic.

mashArt (Daniel et al., 2009) allows users to compose mashup applications using UI components. The sources can be either web applications or services. The integration process is automatic. However, mashArt is centred on the premise that component developers (IT specialists) have componentised UI components from web applications and published these on the web.

Liu et al. (2007) consider a mashup application as consisting of three elements: UI component, service component and action component. This approach allows users to create new applications through simple actions. The integration process is automatic. However, it cannot ensure that every service works well. In addition, it supports only simple actions.

iMashup (Huang et al., 2009) is a web-based service composition environment that uses a web browser as middle-ware. It can build mashup applications using predefined web services. The integration process is automatic.

ServFace (Nestler et al., 2010) aims to create a model-driven service engineering methodology for the integration of service-based applications. ServFace is not a published tool. The web services thus created are generated to ServFace Annotations before they are used in subsequent integrations.

Based on the above analysis, we propose a new approach to the automatic building of mashup applications with general web applications and REST services. Compared with the works discussed above, this new approach offers the following features:

1. our approach can use almost any part of a general web application or of the returned results of a REST service
2. our approach can easily transfer information between mashup components extracted from general web applications or from the returned results of rest services, respectively, without the need for any programming knowledge on the part of the end-user, which allows for more flexible information transfers
3. our approach uses similar descriptions to record information from both general web applications and REST services, which makes that information easier to use.
3 An example based on our approach

To illustrate our approach, we provide a sample mashup application that includes information from both web applications and REST services. Let us suppose that User-A wants to compare the price of several books in different countries or areas. User-A can check the price at Amazon’s website, but the price is displayed in local currency – meaning User-A must then convert the price into local currency. To build a mashup application to accomplish this task, users can use the web service that Amazon supplies to get the price information and then use a currency conversion web service to determine the price in local currency, which will require some programming knowledge; users with no programming background will struggle with this task.

Figure 1 A sample mashup application (see online version for colours)

Our approach will allow users to build mashup applications easily and efficiently. Users can extract the price information and use a currency conversion web service by performing multiple copy-and-paste actions in the user-friendly interface. We illustrate the experiment in detail below, in Section 5. As is shown in Figure 1, however, this web application can provide the user with the price for a given book, identified by its ISBN code, in both local currency and US dollars. After inputting an ISBN code and clicking the ‘Search’ button, users are taken to a web page – the upper portion of Figure 1 – which has the following elements:

a the cover image, title, author and date of publication of the book, from amazon.com

b the logo of amazon.co.jp, the price of the book at Amazon Japan and the price difference compared to the price in US dollars

c the logo of amazon.ca, the price of the book at Amazon Canada and the price difference compared to the price in US dollars

d the logo of amazon.cn, the price of the book at Amazon China and the price difference compared to the price in US dollars

e The logo of amazon.com and the price of the book at Amazon USA.
Figure 2  Image of using web service API (see online version for colours)
A method for facilitating end-user mashup based on description

Users can adjust the layout by dragging and dropping the web content containers, as is shown in the lower portion of Figure 1. Figure 2 illustrates how it is possible to then use a specific web service to perform the currency exchange. User-A would need to complete the following steps in order to build this application using our approach:

1. create the segment-information description (SID) file
2. create the mashup-scenario description (MSD) file based on the SID file
3. setup the location and file name of the MSD file in the template mashup application.

We use an XML-based file, known as a SID file, to describe the information of segments that will be generated from web sources. The segments will be used to generate the components, which in turn will be used to build the mashup application. We use another XML-based file, known as a MSD file, to describe the logic of the generation of the components and of the mashup application. We supply a web-based application that can ease the creating of SID and MSD files, which we explain in detail below, in Section 4. The data, which is part of the SID and MSD files used in this example, is accessible at http://www.tt.cs.titech.ac.jp/~guo/SIDfile.xml and http://www.tt.cs.titech.ac.jp/~guo/MSDfile.xml.

When a mashup application receives a request from a user, it first will read the MSD file and then it will access the web sources according to the information specified therein in order to generate the required components. Finally, it will integrate the components and present them to the user as a web page.

Figure 3 Architecture of M-WA&RS method (see online version for colours)
4 Integration of web applications and REST web services without programming

We call our approach the mashup with web applications and REST services (M-WA&RS) Method. A visual overview of the M-WA&RS Method is provided in Figure 3. It consists of three parts: mashup application template, description files and hide-and-display method. The purpose of the ‘mashup application template’ is to define the display layout of mashup applications, invoke the hide-and-display method, receive and show the results of extracting data. The ‘hide-and-display method’ is a partial information extraction method we designed to extract information from web applications and REST services. The ‘description files’ are two XML-based files, as we noted above, in Section 3.

4.1 Mashup application template

There are two web pages associated with the template for building mashup applications. The first is used to establish the location and the name of the MSD file; the second is used to input keywords, send requests and display the results (Figure 4).

Figure 4 Layout of mashup application template (see online version for colours)

On the second web page, considering the usage of request-submit functionalities of web applications, we placed an input box in which the user can enter keywords. The containers of the extracting results are IFrames. IFrames are commonly used to insert web content from another website into the current web page. Also, IFrames are supported by all major browsers. In our mashup application template, IFrames are created
A method for facilitating end-user mashup based on description dynamically according to the number of the mashup components. After the resulting page is completely rendered, users can move IFrames by dragging and dropping (Figure 1).

To invoke the hide-and-display method, we first must read the two description files and obtain the data before we can then send the request. We have supplied an application that can create the description files (Figure 5). Thus, the structure and the tag names of the XML files are fixed and so we can access the needed data with ease. To show an extracting result, it only needs to be written to the related IFrame.

Figure 5 Generating the description files (see online version for colours)

4.2 Segment-information description file and mashup-scenario description file

The two description files are central to the automatic generation and processing of mashup applications in the M-WA&RS Method. As noted, we use the SID file to describe the information of segments that will be generated from the web sources. These segments will then be used to generate the components used to build the mashup applications. Our approach allows that one mashup component is generated from one or multiple segments; a segment can be part(s) of a web source. For example, a segment generated from a web application can be a paragraph of text on a web page.

4.2.1 Definition of segment-information description file

The SID file contains several separate segments. The information needed to generate each segment is recorded in the SID file. Each segment includes the following items:

- **id**: the unique name of each segment.
- **type**: the type of web source from which the segment was generated. It should be one of either {Application} or {REST}.
- **startPageURL**: the URL of a web source where we can submit the request. For the Application, it should be the URL of a web page where we can submit the search
request to get the target web page. For REST web services, it should be the basic part of request URL.

- **inputType**: necessary only for the Application type of web source, this specifies the type of search method on the start page. It should be one of {LinkList (anchor list)}, {OptionList (drop-down list in a selection box)}, {InputBox (text input field)} or {None}. The M-WA&RS Method supports only these four basic input types for the time being, although we plan to conduct experiments with other input types, such as radio buttons, in future work.

- **inputArea**: necessary only for the Application type of web source, this specifies the path of the input web content on the start page. If there is other content with the same ‘inputType’ on the start page, we must specify which is to be used. Users can give a value of ‘null’ instead of the path expression; ‘null’ is treated as the path of body node. The path we use here, as well as other, similar items, is XPath-like expression. The value of the path can be easily determined using the tool we supply.

- **input**: necessary only for the Application type of web source, this specifies how to emulate the keyword search. It could be the input value from the user interface or the value transferred from another segment. When the search keyword is taken from another segment we use the ‘id’ of the segment to describe it.

- **styleFileForShowInHTML**: necessary for the REST service type of web source, this specifies the location and name of the style file that will be used to convert the returned result of REST services into HTML format.

- **extractionContents**: this specifies the web content to be extracted by path. Here we support the extraction of multiple contents.

- **contentsType**: this specifies the data type of the target web content and could refer to either dynamic or static content. For static content, an additional two sub-items are necessary—the ‘returnNodeName’ and ‘extractionType’. ‘ReturnNodeName’ is any word used to describe the image of the extracted information; this can be included in the style file. ‘ExtractionType’ is a selection from {single text, single link, single object, continue text, continue link and continue object} that describes the type of extraction action. For dynamic web content, a child node, ‘extractionMethod’, is necessary to specify the extraction method from among {Compression Method} and {Visibility Control Method}, which we will explain below, in Section 4.3. The default value is Compression Method because it offers the fastest loading speed from among the available extraction methods.

- **returnType**: this specifies the format of the information that we should wrap into and should be one of either {HTML} or {XML}. We support the merging of segments returned in XML format into a single component. The ‘returnType’ for dynamic web content must be HTML.

- **output**: this specifies the information that will be transferred to other web sources. This information could be used in the generation of segments. For example, it could be used as an input value for a web application type of web source and as a parameter for the REST type of web source. This information is recorded in path expression.
parameters: necessary for the REST service type of web source, this will be added to the base part of the request URL recorded in 'startPageURL'. There are two types of parameters: the information from the 'output' of other segments and a string from user interface of the mashup application.

serviceResultType: necessary for the REST service type of web source, this specifies the format of information returned by a REST service. We currently support three formats: XML, Plain Text and JSON.

memo: this records the attributes of the segment in the user’s words. When users create the MSD file based on the SID file, they have the opportunity to check any associated memos.

4.2.2 Definition of mashup-scenario description file
The MSD file contains several components, each generated from one or multiple segments. The creation of a MSD file must be based on a SID file. The location and name of the SID file will be recorded in the MSD file. Each component includes the following items:

• id: an index number is used to specify the unique name of each mashup component. This index number also decides the order of procession for each of the mashup components.

• execute: this specifies the segment(s) from which the component will be generated. This item is recorded by one or several ‘id’ values of the segments that already 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.

• display: this specifies whether the component will be displayed or not. It should be one of either {True} or {False}.

• styleFile: this specifies the display-style of the component that is generated from the segment(s) that ‘returnType’ is XML. The style file is an XSL file, which is a Style Sheet for XML [XSL transformations (XSLT)]. Here, the user must give the location and name of the style file.

We also supply a web application known as description-generator, which can generate the two description files based on the user’s selection, copy and input actions, as shown in Figure 5. We assume that users will have basic knowledge of HTML and XML.

4.3 Hide-and-display method
In recent years, many approaches to information extraction have been proposed. As each was motivated by a different purpose, not all of the various methodologies are equally effective toward all ends. The goal of our partial information extraction method is to support the type of integration that uses general web applications. As a result, the data source of our information extraction task is the group of accessible web pages. The extraction target could be any part of a given web page, including the static part(s) and the dynamic part(s).
First, we must access the web page using a web application. In order to search for target information, web applications typically provide request-submit functionalities for users. For example, search engine applications provide a text input field for users to input search keywords. Most web applications use the same or a similar layout generator file to show the resulting pages of the same request-submit functionality. These kinds of web pages are called template-based web pages and are the primary data source for our information extraction task. We get the web pages that includes the information we want, which we call the target page, by emulating web applications’ request-submit functionalities.

Second, extraction rules are the main part of information extraction technology, which includes the rule of labelling target extraction data. There are three widely used rules applied to fix the location of content: String Matching rule, Path Matching rule and Pattern Detection rule. The methods that use only either the String Matching rule or the Path Matching rule require well-written HTML pages, whereas those that use Pattern Detection rule are time-consuming. For example, the method that uses the WIKE system (Han and Tokuda, 2008) to generate the pattern information of the BBC Country Profiles web pages requires 16 minutes to run (the computer used for the experiment was running Microsoft Windows XP Professional SP3, with an Intel(R) CPU U1400 @ 1.20GHz, 1.99GB RAM). The lengthy time-to-complete is a result of the method’s need to analyse all web pages before generating the pattern information. We have designed a rule that combines the String Matching and Path Matching rules in order to fix the location of web page content. We use the Xpath feature of HTML documents and the id, name and class attributes, in the following format, to record the location of web content:

- /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 Nth node, [indexN] is the order of the Nth node among the sibling nodes that have the same tag name as nodeN, idN is the ID attribute’s value of the Nth node, nameN is the NAME attribute’s value of the Nth node, classN is the CLASS attribute’s value of the Nth node, and the node(N-1) is the parent node of nodeN.

A flowchart of the process of how we fix the location of web content is shown in Figure 6. First, we look for a unique attribute for the last node. Here, the ‘unique attribute’ means that the value of the node’s id, name or class is unique in the HTML document, or that the value of any attribute plus the tag name of this node is unique; we use the string matching method here. If it exists, we can quickly fix the location of the web content. Otherwise, we can analyse the id, class and name attributes and the tag name of each node in a backward direction of the ‘path expression’ to search for the node that has a unique id, class or name attribute (or one in which the value of any attribute plus its tag name is unique). If there is such a node, we can retrieve the target web content using the path matching method from this node to the last node in the ‘path expression’. In a worst case scenario, there is no node in the ‘path expression’ that has any unique attribute and the process becomes the same as that used in the path matching method.
Finally, we must ensure that the partial information extraction method is applicable to both dynamic and static web content. We can use the approach that extracts the data from the HTML document according to the matching result and then reorganise said data to implement the extraction of static web contents. For dynamic web content, however, we cannot extract only the data according to the matching result because these contents are
created by the scripts and do not work without the executing environment. Thus we need to retain the executing environment when extracting dynamic web content.

**Figure 7** Process of hide-and-display method (see online version for colours)

**Figure 8** Example of path-reader application, (a) page structure analyser – for applications (b) page structure analyser – for REST services (see online version for colours)
Figure 8  Example of path-reader application, (a) page structure analyser – for applications (b) page structure analyser – for REST services (continued) (see online version for colours)

An overview of the hide-and-display method is illustrated in Figure 7. The process is as follows:

Step 1  Analyse the structure of the web application or REST service and the returned data through the supplied application, Path-Reader (Figure 8).

As is shown in Figure 8(a), for the web application, we first must record the functionality users want to emulate. Then we go to a random sample web page by using this functionality to analyse the page structure. Three basic methods are widely used: we can enter a keyword into the input field using a keyboard and clicking the submit button to get the target web page (the *InputBox* method); we can click an option in a drop-down list to view the target web page (the *OptionList* method); or, we can click a link from a list to go to the target web page (the *LinkList* method).

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 must also record the type of functionality and the container information here, including the URL of the web page that the request-submit functionality can access.

As is shown in Figure 8(b), for the REST service, we must analyse and record the structure-rule of the request-URL, as well as the number and locations of the parameters. Then we can send the request according to the given parameters to get an example of the result-data. Finally, we display the returned-data in HTML format for the next step according to the given style file.

Although it is easy to send requests to REST web services, it is not easy to sift through the information from the returned-data without performing some kind of intermediary programming step. The data formats that are widely used in this context include XML, JSON and Plain Text. By converting the returned-data into HTML format, we can use a technique similar to that used to process web applications in order to process REST web services.
If the resultant format of a REST service is plain text, we do not convert the
format of the result and the style information is not necessary. If the result
format is JSON, we convert the result to XML automatically and then convert
that XML data to HTML format with the user-defined information included in
the style file.

Step 2 Record the location features of the contents extracted on the sample web page
into the SID file using the path-reader application and the description-generator
application (Figure 5).

Step 3 Extract the specified web contents according to the given type of web content.
For dynamic web contents, we have prepared two extraction methods. First, we
can extract all of the script tags, meta tags and HTML tags of the specified web
contents. We call this compression method. The second method would extract
the entire HTML page and then hide the HTML tags, excepting the specified
web contents. We call this visibility control method. There are two main
differences between these two methods:
1. the applicability of the compression method is limited when DOM
   operation or hidden values are used in the HTML document
2. if both methods are applicable, the loading time of the extracted dynamic
   contents using compression method is about half that of the visibility control
   method.

For static web contents, we supply two extraction methods. One method is to
extract the target data in a text format, excluding the document’s HTML tags.
For example, the extracted information of a picture is the value of attribute ‘src’
of node <img>. Alternatively, we can use the same method as is used for
dynamic web content, if the users want to keep the same layout of the target
components from the original web page.

Step 4 Output the extraction result. The web contents that have been extracted as
dynamic contents are output in HTML format, whereas those extracted as static
contents are output in XML format, which can be converted to HTML format if
a usable style file is provided in advance.

4.4 Process of M-WA&RS method

The process of the M-WA&RS method is as follows. First, the user sets the location of
the MSD file and sends the request. The mashup application template we supply will then
access the MSD file and retrieve the component information, according to which the
mashup application template will then access the SID file to retrieve the necessary
information for each segment. Second, based on the information recorded in the SID file,
the ‘Hide-and-Display method’ will access the web sources and emulate the requests to
get the original web pages or web service results. Third, the ‘Hide-and-Display method’
will extract the target element(s) according to the record items in the SID file and
generate the segments, which will be used to generate the components according to the
records in the MSD file. To ease the rearrangement of the extracted information, we
allow users to combine the XML files into a single file. As mentioned in Section 4.2, one
mashup component could be generated from one or multiple segments. When the
A method for facilitating end-user mashup based on description

The cooperation of all segments throughout this process is achieved through the collaboration of the ‘hide-and-display method’ and the ‘mashup application template’. The information recorded in the ‘input’ and ‘parameters’ fields (according to the type of web sources) can be the value entered from the user-interface of the mashup application or the output of another segment. These can be considered ‘variables’. The ‘hide-and-display method’ will use the value of such variables to retrieve the corresponding web contents (target contents). The ‘mashup application template’ then helps with the information transmission. Users are able to compose their mashup applications’ scenarios easily, by configuring the ‘input’ and ‘parameters’ fields.

5 Evaluation

5.1 Adaptability

Our approach aims to allow users with only basic knowledge of XML and HTML to easily and efficiently build mashup applications. To evaluate the adaptability and efficiency of our approach, we attempted to build the same type of mashup applications as are shown in Section 3 using several existing approaches. The approaches and their performance are described below.

- **Yahoo pipes**. Because there is no Amazon service supplied in Yahoo Pipes, we instead attempted to use the ‘Fetch Page’ module to retrieve the price of a book. Unfortunately, the content ‘price information’ in the HTML source of amazon.com has only the ‘class’ attribute and does not have any ‘name’ or ‘id’ attribute. Hence we could not set up the parameters of the ‘Fetch Page’ module to retrieve the necessary content. As a result, we could not finish the example mashup application using Yahoo Pipes. There are two restrictions on using the ‘Fetch Page’ module to get specific content in Yahoo Pipes: One is that the content users want to extract must have certain kinds of unique attribute(s), such as ‘id’ or ‘name’; the other is that the size of HTML pages must be under 200k bytes. In addition, users must find the unique attribute manually in order to set the parameters of the ‘Fetch Page’.

- **Dapp Factory**. Although we were able to successfully extract price information from amazon.com using Dapp Factory, the web pages of amazon.co.jp and amazon.cn did not display correctly and therefore we were not able to extract the price information in another currency. In addition, Dapp Factory does not use other types of web sources aside from web applications and RSS services. It is also not possible to conduct currency conversions using Dapp Factory, although the information, once extracted, can be output as a XML file, to an RSS feed, to Google Gadget, etc.

- **C3W** (Fujima et al., 2004). We were able to successfully extract price information from Amazon using C3W but, because it uses only web applications as a web source, we chose to use a web application to convert local currencies to US dollars. C3W is not designed for users to build mashup applications that can be published. Hence,
there are two main problems in building mashup applications using C3W. First, if there is no usable existing application, users have no choice. Second, users cannot change the design of the style in which the extracted information is presented. Although C3W can work with Microsoft Windows Office 2003, it is difficult to build a publishable web application.

Table 1  Mashup novels for user study of MWA&RS method

<table>
<thead>
<tr>
<th>Mashup novel</th>
<th>City information</th>
<th>Book price in different country</th>
<th>Country information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>City name</td>
<td>ISBN code for a specific book</td>
<td>Country name</td>
</tr>
<tr>
<td>Process</td>
<td>Extract the latest news, list and translate the title of the first news item into Japanese. Get weather information from web service</td>
<td>Extract the prices of the book from different country’s bookstores and convert said prices from local currency to US dollars through a REST web service</td>
<td>Extract basic information about a country from web applications and transfer the extracted name of the capital city to a REST service to get the coordinates information. Then, transfer the coordinate information to a REST service to get the time zone information</td>
</tr>
<tr>
<td>Output</td>
<td>Weather information, the latest news list and the title of the first news item in Japanese</td>
<td>The cover image, title and publication data of the book and the logo of the book stores, the price in local currency and the price in US dollars</td>
<td>Some basic facts about the country, as well as the capital city’s coordinates and time zone information</td>
</tr>
<tr>
<td>Web sources</td>
<td>reuters news (application)</td>
<td>amazon.com (application)</td>
<td>Localtimes (application)</td>
</tr>
<tr>
<td></td>
<td>googleTranslate API (REST-JSON)</td>
<td>amazon.co.jp (application)</td>
<td>BBC country profiles (application)</td>
</tr>
<tr>
<td></td>
<td>googleWeather API (REST-XMOL)</td>
<td>amazon.cn (application)</td>
<td>googleGEO API (REST-JSON)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>amazon.ca (application)</td>
<td>earthtools-timezone (REST-XML)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>exchangerate-api (REST-PlainText)</td>
<td></td>
</tr>
<tr>
<td>Number of segments</td>
<td>3</td>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>Number of components</td>
<td>3</td>
<td>9</td>
<td>5</td>
</tr>
</tbody>
</table>

Including the sample mashup application we showed in Section 3, we built several kinds of mashup applications using the M-WA&RS method (screen shots of the results pages of the other two demos can be found at http://www.tt.cs.titech.ac.jp/~guo/Demo1.jpg and http://www.tt.cs.titech.ac.jp/~guo/Demo3.jpg, respectively). The information of these mashup applications are listed in Table 1 (the number of segments and components given here is the data that we implemented when building the applications). The analysis of other approaches along with the experiments we conducted show that our approach is more applicable for building mashup applications that use the web application-type web
A method for facilitating end-user mashup based on description

5.2 Availability

To evaluate the validity and 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 1. These three mashup applications were built 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 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.

Type 2 Users with little or no programming experience.

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 \( (\text{SID file time}) \), the time needed to build the MSD file \( (\text{MSD file time}) \) and the time consumed from the start to the mashup application working correctly \( (\text{Total time}) \). 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’.

All users conducted the experiment in the same environment (on the same computer and network). 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 9 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 sources. Although the M-WA&RS method is applicable to many kinds of mashup applications, the template limits the possibility of the layout. However, users could get well performance mashup application.
web applications. Although User 5 and User 6 did not have much programming experience both were familiar with HTML and XML.

Figure 9  Average time of create information of one segment – user study results (see online version for colours)

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 9). 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.
Table 2  Applicability of the hide-and-display method

<table>
<thead>
<tr>
<th>Website</th>
<th>Input type for searching</th>
<th>Input parameter</th>
<th>Correct target page</th>
<th>Type of target web contents</th>
<th>Web contents work correctly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weatherbonk</td>
<td>Input box</td>
<td>city/country name</td>
<td>Yes</td>
<td>Dynamic content, image</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(China, Canada ...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBS Country Fast Facts</td>
<td>Link list</td>
<td>country/region name</td>
<td>Yes</td>
<td>image, text, dynamic content</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Japan, China ...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNN News</td>
<td>Input box</td>
<td>search keyword</td>
<td>Yes</td>
<td>text</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Panda, economy ...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trippermap</td>
<td>None</td>
<td>n/a</td>
<td>n/a</td>
<td>dynamic content</td>
<td>Yes</td>
</tr>
<tr>
<td>BBC Country Profiles</td>
<td>Option list</td>
<td>country/region name</td>
<td>Yes</td>
<td>image, bullet list</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Japan, Italy ...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yahoo Finance</td>
<td>Input box</td>
<td>stock name</td>
<td>Yes</td>
<td>dynamic content, text, image</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(google, Toyota ...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X-Rates</td>
<td>None</td>
<td>n/a</td>
<td>n/a</td>
<td>Dynamic content</td>
<td>Yes</td>
</tr>
<tr>
<td>Infoplease</td>
<td>Link list</td>
<td>country/region name</td>
<td>Yes</td>
<td>image, text</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(England, Russia ...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SJR</td>
<td>None</td>
<td>n/a</td>
<td>n/a</td>
<td>Dynamic content</td>
<td>No</td>
</tr>
<tr>
<td>CIA</td>
<td>Option list</td>
<td>country/region name</td>
<td>Yes</td>
<td>image, text</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(India, Spain ...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wikipedia (Wikipedia)</td>
<td>Input box</td>
<td>search keyword</td>
<td>Yes</td>
<td>text, image</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Tokyo, sashimi ...)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Second, we wanted to evaluate the availability of the hide-and-display method because it is the main supporting method of the M-WA&RS method. We conducted experiments using different web applications, as shown in Table 2. The results 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 retrieve the correct target web pages and extract the specific web contents, except in the case of SJR – the reason being that the website uses a 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 that use relative path as a parameter of a function, the functionality of the extracted web content does not work correctly.

In addition, we conducted a survey regarding extraction stability. We chose several web applications to generate components and build mashup applications to implement our approaches. The accuracy of the extractions achieved via the hide-and-display method is given in Table 3.
As the results show, the average extracting accuracy exceeds 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 was based on the assumption that the search result pages for the same request function would be the same or similar within a web application. Some search result pages can differ markedly from other pages, however, so that among all 201 countries and regions that we checked in the BBC Country Profiles, we retrieved correct results from 194 web pages. We supported multiple paths to describe the location of ‘targetContent’ and allowed users to decide how many paths they would like to use according to the accuracy they were seeking.

### Table 3  Experimental results about extracting accuracy

<table>
<thead>
<tr>
<th>Web application</th>
<th>Input type for search</th>
<th>Input parameter</th>
<th>Num. of checked pages</th>
<th>Num. of extraction parts</th>
<th>Num. of correct pages</th>
<th>Percent of correct pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC News</td>
<td>Input box</td>
<td>city name (Beijing, Tokyo ...)</td>
<td>194</td>
<td>1 (text, link)</td>
<td>194</td>
<td>100%</td>
</tr>
<tr>
<td>BBC Country Profiles</td>
<td>Option list</td>
<td>country name (Japan, Italy ...)</td>
<td>201</td>
<td>4 (text, image, bullet list)</td>
<td>194</td>
<td>97%</td>
</tr>
<tr>
<td>LocalTimes</td>
<td>Link list</td>
<td>country name (Canada, Russia...)</td>
<td>72</td>
<td>1 (dynamic content)</td>
<td>72</td>
<td>100%</td>
</tr>
<tr>
<td>Weatherbonk</td>
<td>Input box</td>
<td>city name (Beijing, Tokyo ...)</td>
<td>194</td>
<td>1 (dynamic content)</td>
<td>189</td>
<td>97%</td>
</tr>
<tr>
<td>Yahoo Finance</td>
<td>Input box</td>
<td>stock name (AAPL, GOOG...)</td>
<td>36</td>
<td>1 (dynamic content)</td>
<td>36</td>
<td>100%</td>
</tr>
<tr>
<td>Infoplease</td>
<td>Link list</td>
<td>country name (Canada, Russia...)</td>
<td>204</td>
<td>5 (image, text)</td>
<td>186</td>
<td>91%</td>
</tr>
<tr>
<td>Yahoo News</td>
<td>Input box</td>
<td>city name (Beijing, Tokyo ...)</td>
<td>189</td>
<td>1 (text, link)</td>
<td>189</td>
<td>100%</td>
</tr>
<tr>
<td>Total</td>
<td>----</td>
<td>----</td>
<td>1,090</td>
<td>----</td>
<td>1,060</td>
<td>97%</td>
</tr>
</tbody>
</table>

We delivered the name of the capital city, extracted from BBC Country Profiles, to ABC News and Weatherbonk. We retrieved 194 (100%) correct results from ABC News, and 189 (97%) correct results from Weatherbonk. The cause of error can be attributed to the fact that the names of some countries/regions are the same as their capital city – so that, in the Weatherbonk web application, when we input the city’s name, it was treated as a country name and we landed on a different page.

### 5.3 Other aspects

Ryck et al. (2010) gave a concrete definition of the security requirements for mashup applications, which consists of four specific categories: separation, interaction, communication and behaviour control. Our M-WA&RS approach loads components from different origins in IFrames to ensure they are separated by the same origin policy, which offers full separation between different origins. We include the scripts and cause them to be loaded in the included pages, thereby ensuring interaction between the components.
and the original pages. Script inclusion offers no separation, however, as it provides full interaction. Our approach does not allow for the interaction of or communication between the generated components. In future work we will discuss the security mechanism of our approach to make it support the interaction of and communication between the generated components.

Including the security factors, we have the following tasks still to address:

a. The hide-and-display method does not respond well to certain HTTP requests, such as requests that use the Ajax technique. We must design a method to track this kind of request.

b. The return types of REST web service we currently support are limited to the XML, Plain Text and JSON types. We should aim to support all possible types of returned results.

c. We supply only a single template mashup application. We need to allow users without programming experience to design the layout of mashup applications by ourselves adopting a more flexible approach.

6 Conclusions and future work

In this paper, we have presented an approach that allows users with little or no programming experience to build mashup applications with web applications and REST web services without requiring any additional programming. We use two description files, a SID file and a MSD file, to record the information needed for the building of the mashup application. Our approach supports the more flexible transfer of information between the mashup components. The formats used to record the information retrieved from web applications and REST web services are similar. In addition, we use a similar method to process the web applications and REST web services to that which will be used to build the mashup application.

The target end-users of our approach currently are individual users. However, company users can also use our M-WA&RS approach in the prototype design of their own mashup applications. Our approach gives end-users the advantage of easily reusing almost all existing UI components of a given web application.

In future work, we would like to extend the usable web sources of our approach to SOAP-type web services. Furthermore, we will seek to improve the range of the hide-and-display method’s applicability, including the emulation of HTTP requests using Ajax technique. Also, we would like to supply an environment that allows users to design the layout of mashup applications to work together with our approach.

Based on past research into the ‘time notion’ of the Semantic web, our approach touches on the problem of auto-detection of web sources used in mashup applications. This could help the end-user exercise caution if any web source is out-of-date. Moreover, the web form services method (Anjomshoaa et al., 2010) is proposed as a way to help end-users benefit from the simple service composition of mashups and well-controlled BPEL process engines. We would like to find out whether the proposed approach for converting mashups to BPEL processes is applicable to our approach in future research.
References

CBS Country Fast Facts [online]
A method for facilitating end-user mashup based on description


