Chapter 7
Analysis and Design: Towards Large-Scale Reuse and Integration of Web User Interface Components

Hao Han, Peng Gao, Yinxing Xue, Chuanqi Tao, and Keizo Oyama

Abstract With the trend for Web information/functionality integration, application integration at the presentation and logic layers is becoming a popular issue. In the absence of open Web service application programming interfaces, the integration of conventional Web applications is usually based on the reuse of user interface (UI) components, which partially represent the interactive functionalities of applications.

In this paper, we describe some common problems of the current Web-UI-component-based reuse and integration and propose a solution: a security-enhanced "component retrieval and integration description" method. We also discuss the related technologies such as testing, maintenance and copyright. Our purpose is to construct a reliable large-scale reuse and integration system for Web applications.

H. Han (✉)
Kanagawa University, Kanagawa, Japan
e-mail: han@kanagawa-u.ac.jp

P. Gao
Tokyo Institute of Technology, Tokyo, Japan
e-mail: gao@tt.cs.titech.ac.jp

Y. Xue
National University of Singapore, Singapore
e-mail: yinxing@comp.nus.edu.sg

C. Tao
Southeast University, Nanjing, Jiangsu, China
e-mail: taochuanqi@seu.edu.cn

K. Oyama
National Institute of Informatics, Tokyo, Japan

The Graduate University for Advanced Studies (SOKENDAI), Tokyo, Japan
e-mail: oyama@nii.ac.jp

Introduction

More and more information/data is available on the World Wide Web as a result of the development of the Internet, but it is not always in forms that support end-users' needs. Mashup implies easy and fast integration of information to enable users to view diverse Web content in an integrated manner. However, there is no uniform interface used to access the data, computations (application logic) and user interfaces provided by different kinds of Web content. Although there are widely popular and successful Web services such as the Google Maps API [15], most existing websites unfortunately do not provide Web services. Web applications are still the main information distribution methods. Compared with Web services and Web feeds, it is difficult to integrate Web applications with other Web content because open APIs are not provided. For example, the famous global news site CNN[1] provides an online news search function at the site side for general users. However, this news search function cannot be integrated into other systems because CNN has not opened its search function as a Web service. Similarly, BBC Country Profiles[2] does not provide the Web service APIs, so it is difficult for developers to integrate it with other Web services.

Beyond the limitation of open APIs, the integration of Web application content and functionalities at the presentation and logic layers based on the reuse of UI components is becoming an important issue. Here, the Web application functionality indicates part of a Web application/page, and it works as a mechanism for dynamically generating content in response to an end-user request. These functionalities are usually achieved by server-side logic processing or client-side scripting, as shown in Fig. 7.1, and the end-users use the UI components (e.g., the drop-down list in the form in Fig. 7.2 or text input field in Fig. 7.3) to complete the request/response message exchange. Some approaches and systems have been proposed for integrating general Web applications by reusing UI components or emulating the functionality (and content extraction).

However, all of these current technologies and approaches face some common problems such as UI component retrieval in a large-scale Web resource. They are usually limited to a small-scale Web resource, such as frequently used Web applications and websites, or only applicable to some specific domains considering the possible security or copyright issues. The developed mashup applications are weak and have a short lifespan because there are no corresponding systematic testing and maintenance methods.

Moreover, with the development of Rich Internet Application (RIA) technologies, more and more Web content is being generated dynamically by client-side scripts or plug-ins. Originally, the Web browser's role was just to render and display static content. But now to support interacting components (data and code) from

[1] http://www.cnn.com

[2] http://news.bbc.co.uk/2/hi/country_profiles/

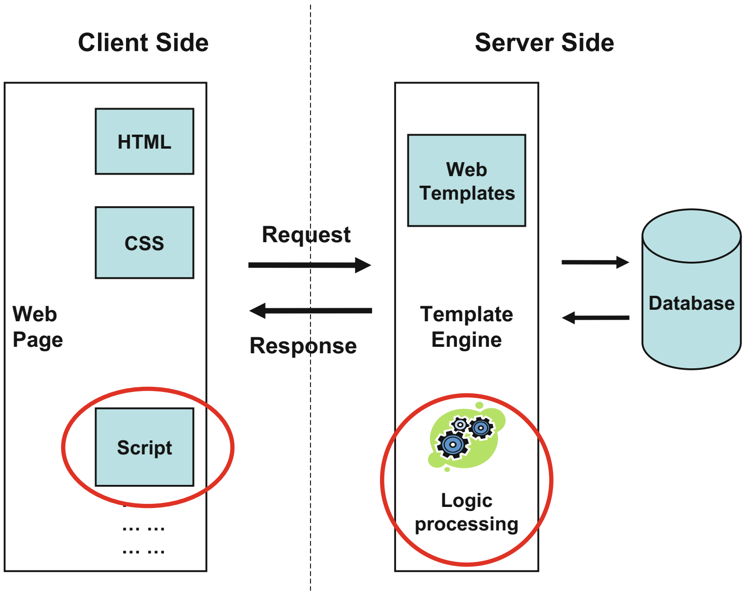
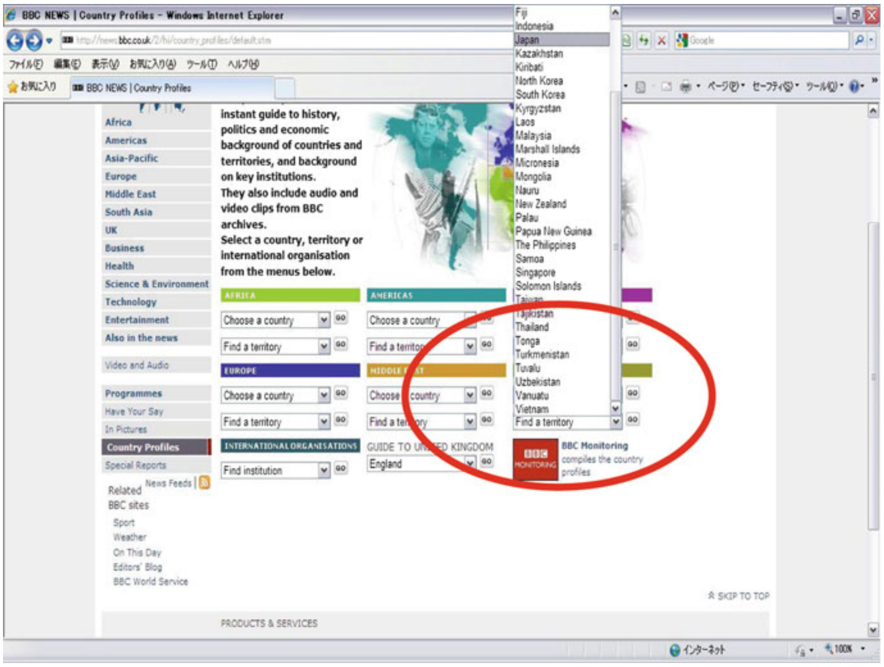
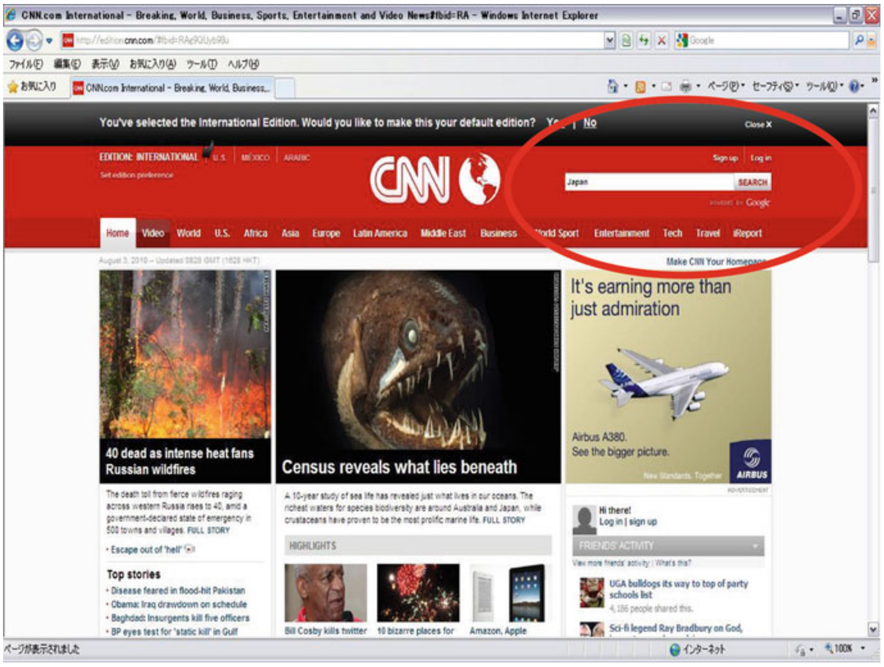


Fig. 7.1 Functionality of Web application



instant guide to history, politics and economic background of countries and territories, and background on key institutions.
They also include audio and video clips from BBC archives.
Select a country, territory or international organisation from the menus below.

Fig. 7.2 Example: option list


CNN.com International - Breaking, World, Business, Sports, Entertainment and Video News#fbid=RA - Windows Internet Explorer
ファイル(F) 編集(E) 表示(V) お気に入り(A) ツール(T) ヘルプ(H)
お気に入り
CNN.com International - Breaking, World, Business...
ページ(P) セーフティ(S) ツール(O)
Close X
EDITION: INTERNATIONAL U.S. MÉXICO ARABIC
Set edition preference
Sign up Log in
Japan
SEARCH
Google
Home Video World U.S. Africa Asia Europe Latin America Middle East Business World Sport Entertainment Tech Travel iReport
Make CNN Your Homepage
It's earning more than just admiration
Airbus A380. See the bigger picture.
New Standards. Together
AIRBUS
ADVERTISEMENT
HIGHLIGHTS
Bill Cosby kills twitter
10 bizarre places for
Amazon, Apple
Hi there!
Log in | sign up
FRIENDS' ACTIVITY
View more friends' activity | What's this?
ページが表示されました
インターネット
100%

You've selected the International Edition. Would you like to make this your default edition? Yes | No

August 3, 2010 – Updated 0828 GMT (1628 HKT)

40 dead as intense heat fans Russian wildfires

The death toll from fierce wildfires raging across western Russia rises to 40, amid a government-declared state of emergency in 500 towns and villages. FULL STORY

• Escape out of 'hell'

Top stories

• Disease feared in flood-hit Pakistan
• Obama: Iraq drawdown on schedule
• Baghdad: Insurgents kill five officers
• BP eyes test for 'static kill' in Gulf

Census reveals what lies beneath

A 10-year study of sea life has revealed just what lives in our oceans. The richest waters for species biodiversity are around Australia and Japan, while crustaceans have proven to be the most prolific marine life. FULL STORY

UGA bulldogs its way to top of party schools list
4,186 people shared this.

Sci-fi legend Ray Bradbury on God,

Fig. 7.3 Example: text input field

multiple parties together and execute them within the Web user's browser, it is clear that at least we need certain security mechanism to be enforced.

In this paper, we describe the main problems with current component reuse in detail and propose our solutions: component retrieval, integration description, systematic testing, and maintenance. We also state the current reliability and security issues, and present our solutions, which involve leveraging some standardized mechanisms, and also present an access control policy (here, the issue of privacy leakage risk of the server-side composition [55] is out of our scope).

Our purpose is to construct a reliable system and develop the corresponding testing and maintenance technologies for large-scale reuse and integration of Web applications. *Our research emphasis is on the reuse and integration of existing Web applications by general Web users at the client side instead of the component or portlet-based development by application developers*.

The rest of this paper is organized as follows. In section "Current Problems and Research", we describe the current problems and research about UI component reuse and integration. In section "Our Solution", we explain our solution in detail. We discuss and evaluate the implementation and related technologies in section "Discussion". Finally, we give our conclusions and mention future work in section "Conclusion".

Current Problems and Research

Most Web mashup technologies are based on the combination of Web services or Web feeds. Yahoo Pipes [51] and Microsoft Popfly [30] are composition tools for the aggregation, manipulation, and mashup of Web services or Web feeds from different websites via a graphical user interface. Mixup [53] is a development and runtime environment for UI integration [10]. It can quickly build complex user interfaces for easy integration by using available Web service APIs. Mashup Feeds [42] and WMSL [38] support integrated Web services as continuous queries. They create new services by connecting Web services by using join, select, and map operations. Like these methods, Google Mashup Editor [16], IBM QEDWiki (IBM Mashup Center) [22], and some other service-based methods [29, 48, 54] are also limited to the combination of existing Web services, Web feeds, or generated Web components.

In this paper, we focus on general Web applications. For the reuse and integration of parts from Web applications that do not have open APIs, the partial Web page clipping method is widely used. The user clips a selected part of a Web page, and pastes it into a personal Web page. Internet Scrapbook [26] is a tool that allows users to interactively extract components of multiple Web pages by clipping, and it assembles them into a single personal Web page. However, the extracted information is part of a static HTML (HyperText Markup Language) document and the users cannot change the layout of the extracted parts. C3W [13] provides an interface for automating data flows. It enables users to clip elements from Web pages to wrap an application and connect wrapped applications using spreadsheet-like formulas and it also enables them to clone the interface elements so that several sets of parameters and results may be handled in parallel. However, it does not appear to be easy for the C3W method to make the interaction between different Web applications (like Safari Web Clip Widgets [6]) and this method also requires a C3W-only Web browser. Extracting data from multiple Web pages by end-user programming [18] is more suitable for generating mashup applications at the client side. Marmite [49], implemented as a Firefox plug-in using JavaScript and XUL, uses a basic screen-scraping operator to extract content from Web pages and integrate it with other data sources. The operator uses a simple XPath pattern matcher, and the data is processed in a manner similar to Unix pipes. Intel MashMaker [12] is a tool for editing, querying, manipulating and visualizing continuously updated semi-structured data. It allows users to create their own mashups on the basis of data and queries produced by other users and by remote sites. However, these methods need professional Web programming ability [9] or have other limitations (e.g., these methods need components produced by portlets [25] or Web Services for Remote Portlets (WSRP) [47]) and they face the following common problems like Mashroom [44] and Dapper [11].

- "*How to find the most suitable UI component from large-scale Web resources?*". Users currently need to find functionalities from websites by using search engines

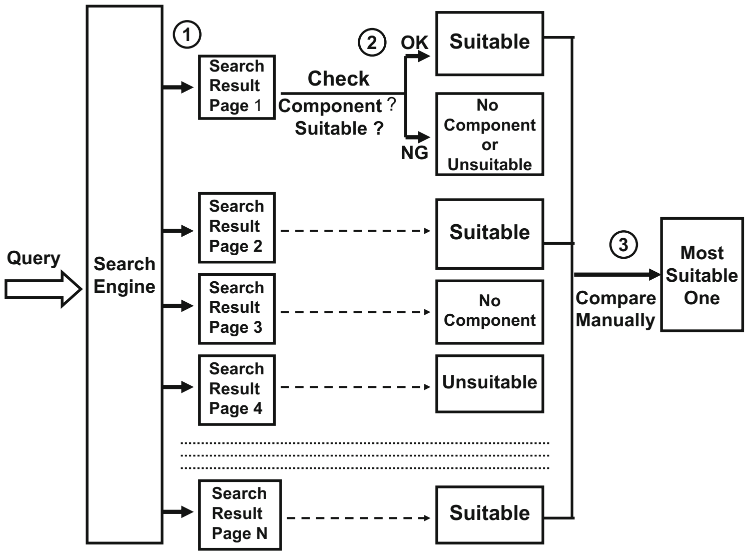


Fig. 7.4 Conventional Web functionality/component retrieval method

(e.g., Google,[3] Bing[4]) or by using a library of predefined UI components, which is also manually constructed and limited to a small scale. As shown in Fig. 7.4, if they use search engines, users must check the search results through the following steps.

1. The user sends the query request (keywords) to the search engine and gets the response page, which contains a list of Uniform Resource Locators (URLs) linked to search result pages.
2. The user checks each result page to determine whether the page contains the suitable Web functionality (UI component) or not (i.e., unsuitable component or no component).
3. The user compares the suitable components and selects one as the most suitable component (for further reuse and integration).

The conventional Web functionality retrieval method is inefficient (even ineffective) because current general Web search engines mainly use a content-oriented search mechanism and information about functionality is beyond the analysis range. Moreover, the search result ranking method of these non-functionality-oriented search engines cannot satisfy the requirements of our functionality retrieval and leads to time-consuming manual checking and comparison.

[3]http://www.google.com/
[4]http://www.bing.com/

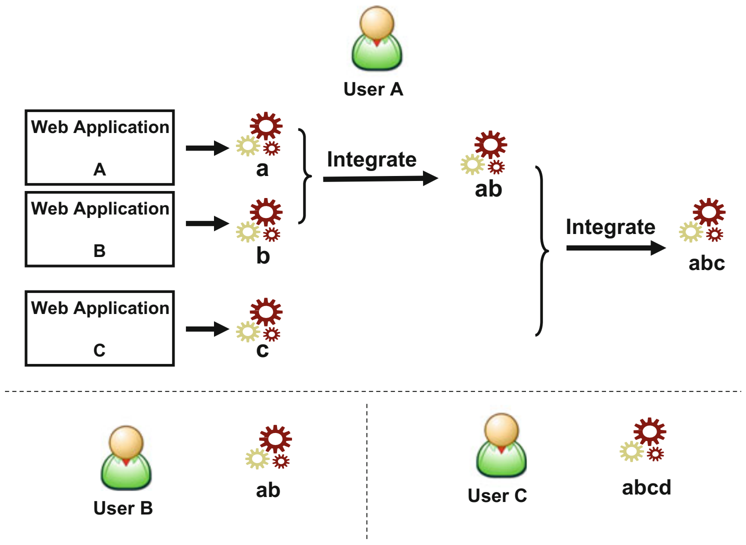


Fig. 7.5 Users need to share and reuse developed parts

In our past experiments and implementations, we had to spend more than 1 h finding and selecting each suitable component on average.

- "*How to share and reuse the developed parts of a mashup application with other users/developers?*". Users can currently select and integrate desired parts (functionality or content) from different applications to develop the mashup applications. However, it is still difficult to share parts of a developed mashup application if they do not know detailed information about the parts such as the interface or parameters. For example, as shown in Fig. 7.5, User A integrates *a* and *b* from Web applications A and B, respectively, and generates *ab*; then, *c* of Web application C is integrated and finally *abc* is generated. However, if User B wants to reuse *ab*, or if User C wants to integrate *abc* with other content such as *d* of Web application D, they must know the detailed internal structures and interface of *abc*. It is difficult for general users to share the developed parts for further reuse easily because there is not a uniform interface or structure for different Web resources and for mashup development.
- "*How to enhance the new integrated software in a stable and seamless composition of UI components from heterogeneous sources?*". Unlike in the case of general Web-service-based Web applications, many UI components (one kind of service) searched by our approach are not registered in online service repositories. Thus, these UI components may be more unstable than ones in a service repository. The more unstable they are, the higher the probability of service evolution is. Overall, these UI components may evolve in terms of their interfaces or functionality from time to time, or they may even vanish or collapse. Apart from evolutions imposed on UI components, access by different users

means that various behaviors can be expected from the same UI components. Specifically, to achieve the goal of stable and seamless composition of the UI components, we need to resolve problems related to two aspects:

- Availability of UI components: No matter how these services were discovered initially, some of them may later become unavailable from time to time. Compared with the public services provided by the major service vendors, UI components are more vulnerable to the changes arising from websites. We need to find a solution that prevents the sudden loss of UI components.
- Adaptability of UI components: These UI components may be used in multiple contexts by users who may have varying requirements regarding their functionality [28]. These differences in requirements for UI components may not be known in advance, but may instead arise from new users when such a service based on certain UI components is already in use.

• "*How to achieve flexible integration of various Web applications?*". The most widely used mode of integrating Web applications is currently "emulating a request submission functionality and getting the response page and then extracting partial information from the response page for further reuse and integration". It is the simplest mode and may fail to treat the following possible situations.

  - Between the page for submitting a request and the page containing the target partial information, there is more than one page transition.
  - For a request, there is more than one possible response. For example, the string comparison is often used in the emulation of optionlist-based page transitions, and the number of corresponding options is not always one.
  - Errors or exceptions are generated and thrown during the implementation of mashup applications.

  To further achieve flexible integration of UI components, a tradeoff analysis of the service granularity is required. The service granularity generally refers to the size of a service. In our approach, it means how big a new service integrated from several UI components should be. The fact that services should be large-sized or coarse-grained is often postulated as a fundamental design principle of applications based on SOA [17]. Some studies on this topic have been conducted [27, 36]. Acquiring a balance between granularity and maintenance efforts will facilitate the sharing and reuse of the developed new service with other users/developers.

• "*How to interoperate components from different origins reliably and securely?*". Currently, Web developers routinely use sophisticated scripting languages and other active client-side technologies to provide users with powerful and fascinating experiences that approximate the performance of desktop applications. However, all extracted UI components execute their functionality directly on the client-side platform, which is usually a Web browser and most Web browsers look like monolithic architectures to mashups at the operating-system level.

– From the standpoint of reliability, allocating all functionalities in the same address space and offering weak isolation between them will allow the functionalities to interfere with each other in undesirable ways.
– From a security viewpoint, all components run in a single protected domain, allowing malicious code to exploit unpatched vulnerabilities to compromise the entire mashup.

Concerning the application level, the Web browser provides only a single principal trust model based on the Same-Origin Policy (SOP) [24] as a base security mechanism to regulate cross-origin interaction among different web content [3]. The specifications of SOP state that content of one origin can access only other content from its own origin: access to resources of other origins is forbidden. This causes either component-component separation with controlled interaction or component-server communication to be impossible. However, the nature of the information/functionality integration and reuse involves interoperation between content of multiple parties' to provide extra-value. Consequently, a contradiction arises: when components of different origins are incorporated, there is either no trust or full trust. Therefore, in a lot of cases, the composers are forced to abandon security in order to get greater functionality, which incurs high security risks because the confidentially and integrity of a component from one site could be completely controlled by components from a malicious site.

On the basis of our analysis of the abovementioned questions, we assert that the current approaches cannot achieve the various types of complicated integration systematically. In our past research, we also developed solutions based on the end-user programming method [18] or non-programming description system DEEP [20] and WIKE [19]. However, those systems were also unable to solve the abovementioned problems well.

Our Solution

To achieve the integration of large-scale Web resources through component reuse, it is necessary to construct a UI component retrieval system and propose a description approach to describing the integration process. Here, we propose our solution, which is being developed to solve the current problems including retrieval, description, reliability and security, maintenance, and access control.

Component Retrieval

The component-oriented search mechanism is used in the UI component retrieval system. As shown in Fig. 7.6, the Web functionality search engine designed for component retrieval works as follows.

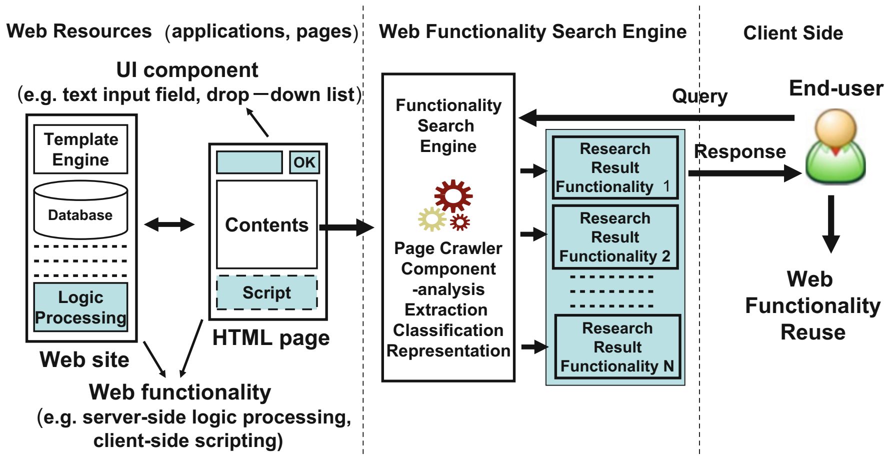

Web Resources (applications, pages)
Web Functionality Search Engine
Client Side
UI component
(e.g. text input field, drop−down list)
Template Engine
Database
Logic Processing
Web site
OK
Contents
Script
HTML page
Web functionality
(e.g. server-side logic processing, client-side scripting)
Functionality Search Engine
Page Crawler Component -analysis Extraction Classification Representation
Research Result Functionality 1
Research Result Functionality 2
Research Result Functionality N
Query
Response
End-user
Web Functionality Reuse

Fig. 7.6 Web functionality search engine

1. A crawler is used to collect Web pages. For each page, the following related information is extracted to judge whether the page contains Web functionalities.

   - Page
     - URL
     - Title
     - Meta data names and values
   - Form
     - ID (ID attribute)
     - Name (name attribute)
     - Method (HTTP method used to submit form)
     - Action (Server-side form handler)
     - Event (e.g., defined events trigger JavaScript functions)
     - Other related information (e.g., hidden values inside the form)
     - Path (XPath containing the tag names and ID values of ancestor tags)
   - Component (See Table 7.1)
     - Type (tag name)
     - Name (name attribute)
     - ID (ID attribute)
     - Value (e.g. current content of text input field)
     - Alt (alternative text for user agents not rendering the normal content)
     - Title (advisory title)
     - Event (e.g., defined events trigger JavaScript functions)
     - Other related information (e.g., text immediately preceding the text input field, text contained within the option element)
     - Structure (e.g., a single text input field, a list of links)

Table 7.1 Information extraction from component (○: extracted, ×: not extracted)

Component	Type	Name	ID	Value	Alt	Title	Event	Other
Input text	○	○	○	○	○	○	○	Previous sibling text
Input password	○	○	○	×	×	○	×	×
Input checkbox	○	○	○	×	×	○	○	Next sibling text
Input radio	○	○	○	×	×	○	○	Next sibling text
Input file	○	○	○	×	×	○	×	Previous sibling text acceptable file type
Input submit	○	○	○	×	×	○	○	×
Input button	○	○	○	×	×	○	○	×
Input image	○	○	○	×	○	○	○	×
Button	○	○	○	○	×	×	○	×
Select option	○	○	○	×	×	○	×	Optionlist text
Applet	○	○	○	×	×	○	×	Class file
Object	○	○	○	×	×	○	×	Code type and URI

2. The extracted information is split into a word list using whitespace, punctuation mark and numeric character as the delimiter. We remove articles, prepositions, and conjunctions from this word list. Finally, these words are analyzed and classified to generate the recognition pattern of each Web functionality and the index information is then created for component retrieval.

Here, the component type is the type of request-input element, such as InputBox (text input field), OptionList (drop-down option list in the selection box), and LinkList (anchor list), in the page where the end user's request was submitted. The users search for the components by giving the component type and other search keywords.

Description and Emulation

After the desired components have been found, they are described in an annotation method for the integration. Here, we define the uniform format shown in Fig. 7.7 for the following basic entities, which describe reuse and integration based on the page transition (*Fig. 7.12 in section "Discussion" gives examples*).

- Flow: a flow represents the data flow or work flow of an integration. It works like the main function of an executive program and receives parameters from users as the request.
- Page: a page is a basic information container in a Web application. It contains the desired partial information such as the component or content, which is extracted in the manner designated by the users.
- Component: a component is used to emulate the functionality and get the response (page) according to the request.
- Collection: a collection contains polymorphic algorithms that operate on collections such as sort, reverse, and swap.

Fig. 7.7 Format of entity

```
<Entity>
    <name></name>
    <comment></comment>
    <input>
        <source></source>
        <element>
            <name></name>
            <path></path>
            <property></property>
            <type></type>
            <matcher></matcher>
        </element>
    </input>
    <output>
        <element>
            <name></name>
        </element>
    </output>
    <exception>
        <message></message>
    </exception>
</Entity>
```

- Checker: a checker is used to check whether the extracted data is valid.
- Convertor: a convertor is used to convert the data into another format.
- Pattern: a pattern is used for comparing strings, searching strings, extracting substrings, and performing other string-based operations as a selective option.

The format contains five main parts: name, comment, input, output and exception. These items reflect the end-user operations (e.g., users find the text input field, input the keywords, submit the request, and search for the target content in the response page) and give the description of the target Web content type/property.

- Name: a name is a simple description of the entity and comments contain the keywords, which reflect the entity's specifications.
- Input:
  - Source: the source of the input is the URL, link, or object output from other entities.
  - Path: a path is used to locate a piece of target content. The value of path is the XPath expression of the target part in the HTML document of the Web page. In the tree structure of an HTML document, each path represents a root node of a subtree and each subtree represents part of a Web page. The response Web pages usually have the same layout or similar layouts if the requests are sent to the same request-submit function. During the node search, if a node cannot be found by a path, similar paths will be used to search for the node. Two paths are similar to each other if they have the same forms, ignoring the difference in node order among sibling nodes when the difference in node order is within a given deviation range.
  - Property: property means text or an image, link, object, or component. Text is a character string in a Web page such as an article. Image is one instance of a graph. Link is a reference in a hypertext document to another document

  or resource. Object is one instance of a video or other multimedia file. Component is the type of request-input element.
  - Type: type contains single, list, or table. Single means a part without similar parts such as the title of an article. List means a list of parts with similar path values such as result list in the search result page. Table means a group of parts arranged in two-dimensional rows and columns such as the result records in a Google Image Search result page.
  - Matcher: a matcher performs matching operations on a string by interpreting a pattern entity.

- Output: output gives the names of output elements and the exception defines the error message generated when an exception occurs.

Compared with the Web services, it is not as easy to access and integrate Web applications because they are designed for browsing by users rather than for parsing by a computer program. The Web pages of Web applications, usually in the HTML or XHTML formats, are used to display the data in a formatted way, not to share the structured data across different information systems. Without an interface like Simple Object Access Protocol (SOAP) [40] or Representational State Transfer (REST) [37], we have to use extraction and emulation technologies to interact with Web applications. Extraction is used to reach the target content and extract it from Web pages. Emulation is used to achieve the processes of sending a request and receiving the response. For request submission, there are the POST and GET methods, and some websites use encrypted code or randomly generated code. In order to get the response Web pages from Web applications of all kinds automatically, we use HtmlUnit[5] to emulate the submission operation instead of using a URL templating mechanism. The emulation is based on the event trigger, as shown in the following examples.

- In the case of InputBox (enter the query keywords into a form-input field from the keyboard and click the submit button by using a mouse or trackpad to send the query), the text input field is found according to the path and the query keywords are input. Then the submit button's click event is triggered to send the request and get the response Web page.
- In the case of LinkList (clicking a link in the link list on a Web page to go to the target Web page), the text contained inside each link tag along the path is compared with the keyword by matcher until a matching one is found. The link's click event is then triggered to get the target Web page.
- In the case of OptionList (click an option in the drop-down list of the selectbox in a Web page to view a new Web page), the text of each option within path is compared with keyword by matcher until a matching one is found. Then the select event of option is triggered to get the target Web page.

[5] http://htmlunit.sourceforge.net/

Maintainability of Integrated Application

The essence of our approach is to broaden the scope of reusable UI components and further distill useful ones from the massive amount of Web resources. With the advantage of more reuse opportunities, however, our approach compromises in the maintainability of the new integrated applications.

First, the availability and stability of these UI components are an important issue that we must address. In our approach, many Web resources, such as some static Web pages that are officially not qualified to be “so-called” services, will be considered as potential candidates for extraction targets. Compared with Web applications based on Web services, these services are usually all provided by third-party service providers. For example, an system based on Google Office Service [14] and Amazon Storage Service [1] is usually stable because the leading IT companies have a team of engineers maintain their service. But what if office services and storage services were based on UI components from unknown websites, rather than from those major service vendors? Thus, these components are subjected to evolution, and they are more unpredictable. To overcome this problem, we propose to empower the mashup application in looking up the backup UI components in case of an emergency such as some existing components being unavailable or suddenly changed. For example, in our own implemented news release system, we usually extract the desired news from CNN[6] or the BBC[7] according to some tags or keywords. If CNN and BBC were to be unavailable for a while, our mashup application would detect the sudden loss of the corresponding UI components and immediately substitute new ones from the New York Times[8] or ABC News[9] for the lost ones. We propose to design a ranking algorithm to rate components when multiple UI components with the same or similar functionality are available. The rating strategy can be based on service recommendations based on users’ client-side performance [43].

Second, the maintenance of such an integrated application is bound to be complicated when the desired UI components are gathered together to implement a new application. Instead of being treated as a single independent service, our new integrated application logically has external coupling with many other websites from which the internal UI components are extracted. As a recent empirical study [34] pointed out, intra-service coupling (in our case, intra-service coupling refers to the coupling between UI components) can potentially have a smaller effect on maintainability (i.e. analyzability, changeability, stability, and testability) than indirect and direct extra-service coupling (in our case, intra-service coupling refers to the coupling between a UI component and other components from other website).

[6]http://www.cnn.com

[7]http://www.bbc.co.uk

[8]http://www.nytimes.com/

[9]http://abcnews.go.com/

Overall, our integrated application has high cohesion and also a certain degree of external coupling. As an aspect of security reasons, the UI component isolation principle (see details in section "Reliability and Security") is advocated in our design. It is this principle that makes the UI components (intra-service) have high cohesion and low coupling. However, the drawback is that with more isolated UI components from more external websites, the external coupling between our integrated application and the other websites will correspondingly increase. In our solution, a case-by-case tradeoff analysis between external coupling and finer-grained isolation of components for the different integrated applications will be beneficial to the maintenance.

Third, we are concerned with the adaptability of the UI components in our approach. A multi-tenant architecture allows many tenants (users) to share the same application (or service) instance [28]. Supporting service variability in the scope of a one-fits-all service creates extra challenges. The existing approaches allow users to adapt workflows via configuration options with only weak provisions for service adaptability [32, 39]. The high variability of multi-tenant services usually translates into higher maintenance costs. Relaxing the single-instance requirement, Model-Driven Development has been applied to produce custom services in the ERP domain [41]. Our approach to service variability may contribute to multi-tenant services (dynamic reconfiguration), and the situation where custom instances of a service are generated for users. The Software Product Line (SPL) approach to reuse [5] exploits commonalities and variability in a domain to improve productivity of developing and maintaining many similar yet different applications. The variability management techniques used in SPL [35, 52] are relevant to handling service variability [31], which is exactly the direction we plan to explore in the further studies.

To sum up, if the problems with the availability and stability of these UI components are overcome, mashup applications will be enhanced to prevent the sudden loss of their UI components and to rate the candidate UI components. To achieve a balanced design between external coupling and finer-grained isolation of components, we will use some metrics [34] to calculate the external coupling and reduce it to a manageable degree. Finally, we will also customize and encapsulate the internal UI components to satisfy the diverse requirements of customers by applying SPL techniques.

Reliability and Security

The trend of RIA technologies is accelerating the development of rich client solutions. For example, many websites use DHTML (Dynamic HTML) or Document Object Model (DOM) scripting to create interactive Web pages. They contain HTML, client-side scripts (such as JavaScript), DOM, etc. The scripts change variables (including elements outside the target parts such as hidden values) in a programmed manner in an HTML document, and affect the look and function of static content. The trend shows that there has been a dramatic increase in the number of active programs over the past few years, and these applications are larger and more complex than before. When arbitrarily mixed on the client side, such active content will lead to problems with reliability and security. For example, if a bug crashes one functionality, it may crash all the others being executed within the browser; if a functionality has an SQL injection vulnerability, it has the potential to leak private data of other functionalities. In this situation, to integrate and reuse components reliably and securely within our approach, we give four specific requirements.

- UI Component Isolation: UI components from different origins should be isolated from others by sandboxing. The mechanism should be in the system layer, which can prevent memory leaks.
- Script Separation: A functionality that contains script code needs to be separated from other functionalities. Even when scripts are from the same origin, the DOM tree and scripts must not be modifiable in an unauthorized manner by other functionalities.
- Functionality Interaction: Each functionality should have the ability to interact with other functionalities and the integrator. The restriction is that an entity that contains critical information should not be read or rewritten by others without authorization. Another aspect is that functionalities should be able to perform cross-domain communication. Then a functionality can communicate not only with others from its own origin but also with those from other origins in a controlled manner.
- Usage Control: The author of reusable components should be able to decide whether or not the specific functionality can be accessed by other components, which can bring merits about the copyright and risk declaration.

On the basis of the requirements presented above we firstly leverage the multiple processes of the operating system (OS) to isolate cross-origin components to resolve the reliability problems for UI-component-based mashup applications. And we present a security policy for separating JavaScript code from the same origin. Then concerning practicality and extensibility, it is meaningful to leverage existing standardized technologies to solve the functionality-interoperating and cross-origin communication issues in the application layer. Finally, we present a functionality access interface in order to enforce the usage control on components.

The solutions are illustrated with the scenario in Fig. 7.8: functionality N_1 from website N, which refer to scripts X_{ai} from origin X, is integrated into the integrator page A, which also contains functionality A_1 from website A. As a result, the integrator contains two parts: functionality A_1 and functionality N_1, which are from different origins. Each functionality may contain both static (HTML) content and active (Script) content.

- The iframe [23] separates one inclusion of site N (N1.html) from site A with no interaction under the SOP restriction. The architecture of our solution for component isolation makes each UI-component-based mashup application into

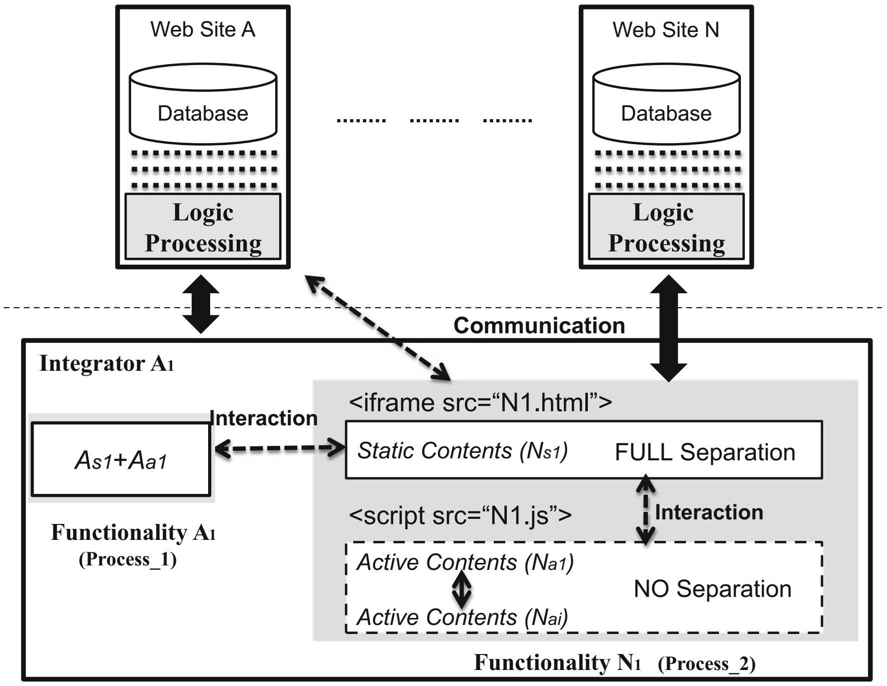


Fig. 7.8 Example of interoperation model (An *arrowed dashed line* implies that interaction or communication is restricted by the SOP)

a separate OS process. UI components from different origins (A, B,. . . , N) are isolated from each other in different processes (shadowed region on the client side). For the included page A and each functionality that the mashup contains, a new rendering process will be created; it draws HTML pages and runs JavaScript code. The included page is rendered by one process while UI components from different origins are rendered by different processes. The process creation policy is that for each UI component in a mashup, if a process for that origin has not already been created, then a new rendering process is launched.

- The component isolation targets UI components from different origins. However, as shown in Fig. 7.9, we still need to consider an appropriate security mechanism for separating JavaScripts from same origin because third-party JavaScript references are commonly used in Web applications/pages, such as X_{ai} in Fig. 7.8, which might be malicious. If access to the JavaScript context is fully allowed, then code in a malicious user's JavaScript references could override the original functions and other objects with malicious ones. In that case, when a trusted script calls the "original" function, the malicious code will be executed. To handle this case, we propose an access control policy, as shown in Table 7.2, for the JavaScripts from the same origin within a process where the *read* operation is to read the value or property of an object; the *write* operation is to create a new

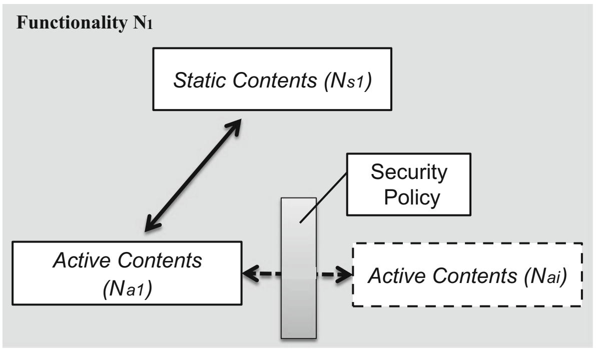


Fig. 7.9 Separation of JavaScripts from the same origin

Table 7.2 Access control policy for JavaScript objects

Access direction	Read	Write	Execute
Origin trust to destination trust	Allow	Allow	Allow
Origin trust to destination distrust	Audit	Allow	Allow
Origin distrust to destination trust	Disallow	Disallow	Audit

object, assign a value to an existing object, or define or redefine a function object; the *execute* operation is to invoke a JavaScript function.

- The iframe tag separates one inclusion of site N (N1.html) from site A and even from its own scripts with no interaction under the SOP restriction. The postMessage [21] is a popular browser API extension that provides principal interaction between frames and iframes. It provides the postMessage( ) operation on a DOM window and iframes object. Combining the postMessage with iframes, which offers restricted interaction and separation, is an efficient method. The XMLHttpRequest object of JavaScript is not allowed to make cross-origin requests. If component N_1 needs to communicate with integrator A_1's remote Web server, Cross-Origin Resource Sharing (CORS) [8] is an efficient and widely supported extension of HTTP to provide cross-origin requests. It allows the remote Web server to indicate whether or not a functionality of another origin has the rights to access its resources.
- As shown in Fig. 7.10, our framework has an interface for inserting a new item called "access" into the format of entity. The access tag that assigns permissions to the UI components in an application and specifies an usage policy to protect its resources. It uses the "public" or "private" attribute to declare the *copyrightType*, which indicates whether the entity can be accessed by the normal public flow or only used privately in order to protect the copyright of a special component. Moreover, it can specify the applications/pages of which origin can

```
<access>
   <name></name>
   <comment></comment>
   <copyrightType></copyrightType>
   <permissionOrigin></permissionOrigin>
   <permissionDestination></permissionDestination>
   <accessLevel></accessLevel>
</access>
```

Fig. 7.10 Access item

have access by means of the *permissionOrigin* and *permissionDestionation* tags. Furthermore, given the type of UI component, such as InputBox or OptionList, the access item uses *accessLevel* tag, which contains "normal", "dangerous", "specific" attribute, to indicate the system's security level, which should be brought to the composer's notice, and the tag should grant permission to a functionality requesting it.

– Normal: Low risk access level. The functionality has only a fixed value or it does not access any other functionalities. For example, the OptionList and LinkList belong to this level.
– Dangerous: High risk access level. The functionality has the ability to receive input or has the right to access private data (e.g., SQL Injection). For example, the InputBox belongs to this level.
– Specific: An access level specified manually by the composer. This option is used for certain special situations and should be set with great care, as any possible composition could be done without regulation.

Discussion

We consider different performance measures to evaluate our approach and to discuss related topics in actual integration.

Retrieval and Description

The construction of the functionality/component search engine solves the component retrieval problem described in section "Current Problems and Research", and the uniform format of the basic entities provides a solution to the problems of flexible description and integration, as shown in the following example.

As shown in Fig. 7.11, we describe the function of country information search by the entities. We search for the select-option at the top page of BBC Country

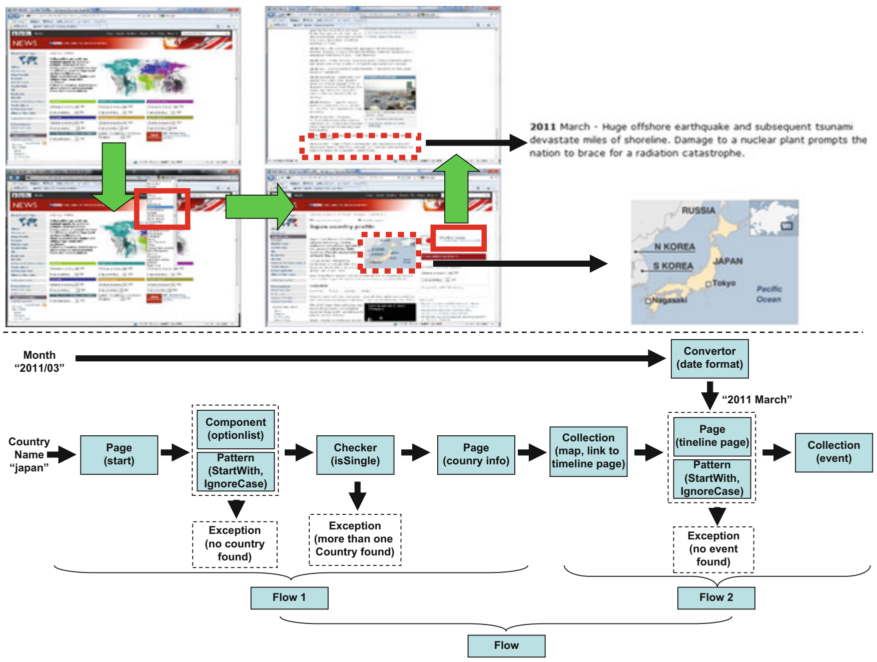

2011 March - Huge offshore earthquake and subsequent tsunami devastate miles of shoreline. Damage to a nuclear plant prompts the nation to brace for a radiation catastrophe.
RUSSIA
N KOREA
S KOREA
JAPAN
Tokyo
Nagasaki
Pacific Ocean
Month "2011/03"
Convertor (date format)
"2011 March"
Country Name "japan"
Page (start)
Component (optionlist)
Pattern (StartWith, IgnoreCase)
Checker (isSingle)
Page (counry info)
Collection (map, link to timeline page)
Page (tineline page)
Pattern (StartWith, IgnoreCase)
Collection (event)
Exception (no country found)
Exception (more than one Country found)
Exception (no event found)
Flow 1
Flow 2
Flow

Fig. 7.11 Example of page transition and entity definition

Profiles[10] and submit the request to get the target country page. We extract the map of the country and the URL of the timeline page from the country page. We search for key events using the given month and extract them from the timeline page. The definitions of Page (Start) and Component (optionlist) are given in Fig. 7.12 as examples. We use Firebug[11] to get the XPath of the target part via the Graphical User Interface (GUI) and avoid manual analysis of the HTML document. The entities are combined with each other, and finally the whole flow is generated. The matcher, pattern, and checker are used to keep the correct process, and exception messages are generated if errors occur. Moreover, the convertor and collection create the event searching function by extracting the corresponding information from a simple static Web page. As a part of the *Flow*, the *Flow 1* or *Flow 2* can be reused by other users, who do not need to know the detailed internal structures of these sub flows because their formats are the same.

We achieve component-based reuse and integration of Web content with component retrieval and entity description. Similarly, Web Service Definition Language (WSDL) [46] and Web Application Description Language (WADL) [45] are used to describe a series of Web services in detail such as abstract types, abstract interfaces,

[10] http://news.bbc.co.uk/2/hi/country_profiles/

[11] http://getfirebug.com/

```
<Entity:Page>
    <name>BBC_Country_Profiles_Page</name>
    <comment>BBC, country profile, search</comment>
    <input>
        <source>URL:http://news.bbc.co.uk/2/hi/country_profiles/default.stm</source>
        <element>
            <name>CountrySelection</name>
            <path>/html/body/div/div/div[2]/table/tbody/tr/td[2]</path>
            <property>Component:OptionList</property>
            <type>Table</type>
            <matcher></matcher>
        </element>
    </input>
    <output>
        <element>
            <name>Component:CountrySelection</name>
        </element>
    </output>
    <exception>
        <message></message>
    </exception>
</Entity:Page>

<Entity:Component>
    <name>Country_Name_Selector</name>
    <comment>BBC, country profile, search, optionlist</comment>
    <input>
        <source>Page:BBC_Country_Profiles_Page/Component:CountrySelection</source>
        <element>
            <name>CountryName</name>
            <path></path>
            <property>String</property>
            <type>Single</type>
            <matcher>Pattern:{StartWith,IgnoreCase}
            </matcher>
        </element>
    </input>
    <output>
        <element>
            <name>Page:CountryInfo</name>
        </element>
    </output>
    <exception>
        <message>no country found</message>
    </exception>
</Entity:Component>
```

Fig. 7.12 Start page and optionlist components

and concrete binding. We could not believe that all the tasks could be completed in a fully automated way by handing WSDL or WADL files to WSDL2Java or WADL2Java [2]. Compared with the WSDL and WADL, our entities can use a shorter and simpler description format and are applicable to the description of general Web applications. It is easier to read, write, reuse, and update any part of a mashup Web application than to use end-user programming methods. Our annotation approach makes it possible for users with no or little programming experience to implement Web content integration from various Web applications. Any content from any kind of Web applications can be handled.

Layout Personalization

Different content extracted from different websites has its own different layout, such as size, font, color and other view styles. Except the components and the objects generated by client-side scripts, both the Web service response and the partial information extracted from Web applications are in XML format. During the integration, we need a personalizable template processor to transform XML data into HTML or XHTML documents. Our approach provides an XSL Transformation (XSLT) [50] library, which contains the following two types of XSLT files.

- We created an XSLT file (automatch.xslt), which automatically transforms the XML format into HTML format by using the type/property. It is applicable to the basic types including text, images, links and objects. For example, it uses the <img> tag to embed an image and set the extracted URL as the attribute value of "src". The transformed result is a simple list/table of Web content, and the optional attributes (font, size and etc.) are left unset. This file is a recommended default template for the transformation of extracted results.
- We designed a number of XSLT files to create XSLT library, and users can select suitable XSLT files for the target Web content. These XSLT files are applicable to often-used structures of partial information of Web applications (list, table, etc.) after some manual modifications.

Users can select our predefined XSLT files as the stylesheet transformations or modify them to get customized visual effects. Moreover, the fully original XSLT files created by users are valid in our approach.

Reliable and Secure Composition

Currently, the only browser that supports multiple-processes is Google Chrome.[12] However, our experimental results show that it still cannot deal with the UI-component-based mashups. For instance, in the case of the example shown in Fig. 7.11, which we introduced above, all application content including the select-option and the country page are executing together; here, where a reliability problem can occur if one functionality that is CPU-bound can prevent the CPU from interacting with the other functionalities. In our proposed architecture, each process runs in a separating address space. Therefore, crashes caused by one origin will not affect the content of other origins. The OS has a process scheduling mechanism, so expensive computations and the freezing of function calls from one origin will not starve the system of resources from other origins. The OS also supports the killing of any process that becomes unresponsive because of memory leakage.

[12]http://www.google.com/chrome/

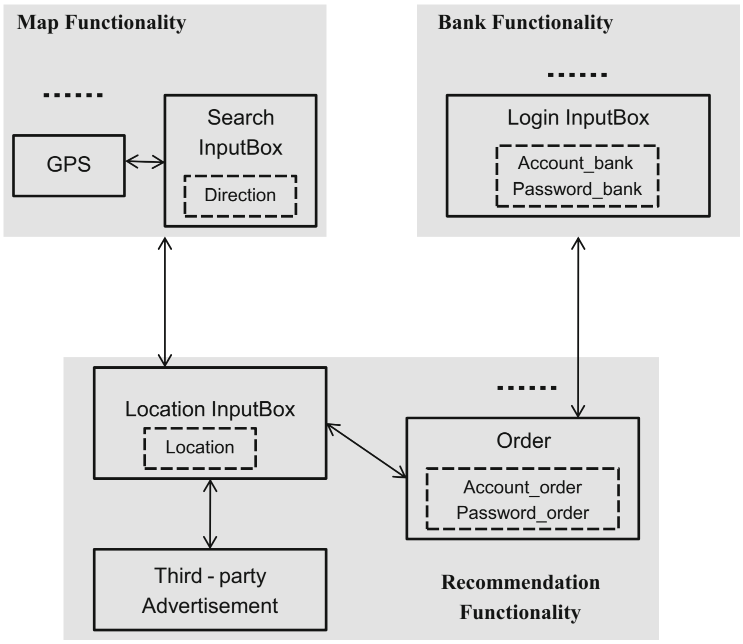


Fig. 7.13 Example of a travel mashup application

In terms of performance, processes are efficient because the time overhead for process startup and work is lower compared with page rendering and JavaScript execution times. The memory overhead is higher than in the conventional method because an HTML rendering engine must be loaded in many processes. Regarding backwards compatibility, because our approach places pages from different origins in different processes, it can support effective isolation without breaking the current architecture of Web pages, it does not bring communication boundaries among pages through the shared memory.

The aim of the example shown in Fig. 7.11 which contains almost all static content from one origin, is to address the workflow of our approach. However, we still emphasize that the reliably and security issues cannot be ignored because the active content such as the JavaScript code is widely used within attractive and complex Web applications and pages. To perceive the security requirement more directly, we start our discussion from the practical scenario shown in Fig. 7.13: you (a Web user) composite a travel mashup application into your smart phone by reusing and integrating several functionalities from existing popular Web applications. The travel application extracts a component of a map application with Global Positioning System (GPS) location and transportation query functionality, an online payment functionality from the credit card authentication component of your bank, and a functionality from a tourism agency component for getting tour or shopping recommendations. The functionalities from the map and recommendation applications need to interact in order to provide relevant tour and shopping guidance based on your current location at the time of your request. The recommendation functionality needs to communicate with the server of the bank functionality to make a reservation or purchase and may also contain a third-party advertisement part for generating extra revenue.

In this scenario, we can use the following techniques, which have been introduced above, to meet the security requirements: The map, bank, and recommendation functionalities are isolated as single OS processes within the same functionalities and interoperation of static content and active content is enabled through the combination of the iframe and the postMessage; however, we propose to use object access control to ensure that the third-party advertisements cannot request private information such as the credit card password and GPS location history. The *accessLevel* of the InputBox from each functionality should be defined as "dangerous" because a Web application could be created by a malicious user intentionally with some vulnerabilities such as SQL injection or XSS attack (user privacy accessed by the malicious functionality could be leaked). The recommendation functionality will expose an interface to the public and its interface can be used by the bank and third-party advertisement. The map and recommendation functionalities can communicate with their servers using the conventional XHR method. For communication with the other functionalities, the recommendation functionality can use CORS to make the remote server allow requests coming from other origins.

Integration Testing

When developing a Web-UI-component-based mashup application, we test each UI component individually. Clearly, components may have been developed separately, written in diverse programming language, and executed in different Web platforms. Therefore, integration testing is necessary to ensure that communication between components is correct. Some interaction-related faults are likely to be overlooked during unit-level testing. These faults can be classified into programming-related faults, which are inter-component faults, and non-programming-related faults, which are interoperability faults.

Conventional integration testing of component-based applications is based on the function decomposition, which includes the big-bang, top-down, bottom-up, and sandwich approach. Function decomposition approaches, which do not consider the internal structure of the individual components, are based on the specification of the system.

Owing to the heterogeneity and implementation transparency of Web-UI-component-based applications, we propose a model-based approach to integration testing. The core model here is the Component Interaction Graph (CIG), and the testing method is UML-based state testing. Here, CIG addresses the relationships that include message-communication, usage, etc. CIG can be defined to model interactions between UI components by depicting interaction scenario. CIG can be used to develop a family of test adequacy criteria, such as all nodes and all links coverage. In a component-based application, a component may interact with other components directly through an invocation of the interfaces exposed by those components, an exception, or a user action that triggers an event. A component may also interact with other components indirectly through a sequence of events. Interface and event testing ensure that every interaction between two UI components is exercised. UML provides high-level information for the internal behavior of components, which can be processed efficiently and used effectively during testing. In UML, collaboration diagrams and sequence diagrams are used to represent interactions between different objects in a component. A state diagram can be used to characterize the internal behaviors of objects in a component. Object Constraint Language (OCL) can be used to formally specify the behavior and constraints of a UI component. UML collaboration diagrams and sequence diagrams focus on interactions among various objects within a use case (say a component for simplicity here). If a UI component covers multiple use cases, we need to develop a consolidated collaboration diagram. If a use case includes more than one component, we need to develop a subset of the collaboration diagram. In UML sequence diagrams, interactions are ordered by time, while in collaboration diagrams, the same information is presented with the interactions ordered by the numeric message sequence.

With collaboration diagrams, we can obtain all possible sequences that could be invoked to model how interfaces and events interact with each other. The behavior of a Web UI component can be modeled more precisely by combining the collaboration diagram with statechart diagrams. By using the statechart diagram, we can further refine the interactions between interfaces and events. Given a statechart diagram, context-dependence relationships include not only all possible sequences in a collaboration diagram, but all possible combinations of the sequences that are shown in the statechart diagram as well.

Service Granularity

Insofar our approach has not achieved automatic and flexible composition of the intra-services (the UI components) based on the optimization of service granularity for better sharing and reuse. At the current stage of our prototype, we are focusing on addressing the maintenance and security issues discussed in section “Our Solution”. Since the optimization of service granularity is such an advanced issue, it should be considered mainly after the desired UI components have been extracted from external websites and integrated by the mashup application as the intra-services.

Service granularity has never been stated as a purely technical issue [4,17,27,36]. More specifically, Haesen et al. proposed that the decision about service granularity should be made according to the following types of granularity [17]:

- Functionality granularity referring to the size and scope of functions offered by a service. In our integrated application, if two UI components are often used together, we can wrap these two components as one bigger service and further share and reuse this service with other applications. For example, in a travel mashup application, the users want to use the GPS location components provided by Google Maps to find the path to a destination; meanwhile, users are also concerned about the weather situation at the destination and the traffic situation on the way to the destination. Thus, we integrate the UI components for GPS search from Google, the weather enquiry from Yahoo, and the traffic situation enquiry from some real-time traffic monitoring system. These UI components are encapsulated as a bigger service from this viewpoint of functionality granularity.
- Data granularity reflecting the amount of data that is exchanged. In our application, if two UI components have their own data dependency from external websites, it will be difficult to compose these two components as a bigger one. Nevertheless, in the travel mashup application, the UI components from a bank website require login and authentication, and the UI components from an online hotel reservation website also require registration and login. Consequently, it is inconvenient to compose these two components, as they have their own authorization operations. Actually, the data granularities are attributed to the coupling, cohesion, and statelessness [33] of the intra-services.
- Business value granularity of a service indicating to what extent the service provides added business value. This aspect actually does not affect the design of the service granularity in our solution. Business value granularity is about attaining cost effectiveness from the business viewpoint. To maximize the profits from these services, the service vendors will provide a coarse-grained service to general customers and a finer-grained service to some special customers. As our application is to build open and reusable services from existing UI components, the business motivation may not be our major concern.

Kulkarni et al. [27] also argued that a tradeoff decision should take into consideration non-functional requirements, particularly performance issue. Except performance, Steghuis [4] reported that designers must make tradeoffs about service granularity in and between the following areas: flexibility in business processes and reusability. Thus, service granularity is actually relevant to every aspect of the system. Further exploration on this topic is included in our plans for future work.

Copyright

We have to face up to the problem of copyright in information reuse and integration. Copyright is the right of the owner to reproduce or permit someone else to reproduce copyrighted works. Websites usually own the copyright of content that they publish, so we need permission from the websites before we can reuse it.

Many websites or services give a clear declaration about the copyright. Public domain, such as the works of Shakespeare and Beethoven, means publicly available and covered by “No rights reserved”. The most common type of copyright is “All rights reserved”. In this case, if we use an image on our website that we did not create ourselves, we must get permission from the owner. It is also common for the text, HTML, and Script elements of a Web page to be taken and reused. If we have not obtained permission, we have violated the owner’s copyright. Between these two copyright policies, there is a more flexible copyright model: “Some rights reserved”. Wikipedia is provided under the terms of the Creative Commons [7] public license, which is devoted to expanding the range of creative works available for others to build upon legally and share.

Open Web service APIs are provided by some websites. The websites usually give a detailed policy of copyright for users. Information reuse based on open APIs has to obey the copyright policy defined by websites. Compared with open APIs, general Web applications do not provide clear end-user-programming-oriented interfaces. The legal and practical reuse of content extracted from Web applications cannot be left out of the consideration.

Here, we classify the content of Web applications into two categories as follows.

- No rights reserved content. Copyright does not protect facts, ideas, systems, or methods of operation. For example, facts like the population or national flag of a country may be freely reused without permission. Moreover, there are some special permissions. For example, reuse of the titles (not body) of Web news articles, book introduction headlines, and other brief descriptions is permitted without copyright. Furthermore, some social services/applications, such as the micro-blog Twitter, permit general Web users to share and reuse their content.
- All rights reserved content. Typically, most of the intellectually created content is protected by copyright. For example, photographs, Web news/blog content, and map information. For content of these types, reuse is forbidden without copyright permission. So, we need to declare the copyright information if we want to reuse the content. If others use our description and definition of component-based reuse, the declaration (*copyrightType*) should and must be inherited.

Conclusion

In this paper, we described current problems of Web-UI-component-based large-scale Web application reuse and integration and presented a UI component retrieval system that achieves quick and effective component searching. We also proposed a description method to define the basic entities in application reuse and integration. Moreover, we analyzed the contradictions between the conventional security mechanism SOP and the nature of mashup interoperation. We then presented our security requirements and solutions. Finally, we presented a systematic maintenance method and discussed personalization, testing, and copyright. Our retrieval and description system enables typical Web users to construct mashup Web applications easily and quickly.

As future work, we plan to conduct a more comprehensive analysis of current problems and modify our approach as follows.

- We will combine and develop various techniques/tools and propose a friendly GUI for users to generate the entity description file more easily and completely automatically.
- We will explore more flexible ways of integration and construct an open community for sharing various mashup components. The technologies of the Semantic Web and RDF will be used to centralize various Web applications, Web services, and other Web content.
- We will propose a quantitative and qualitative analysis as the evaluation standard for components (e.g., speed, stability and reuse range).
- We will develop a fine-grained access mechanism that can trace the information flow in the functionalities.
- We will propose a more stable extraction method to solve the problems of frequently changing content formats.

Acknowledgements This work was supported by a Grant-in-Aid for Scientific Research A (No.22240007) from the Japan Society for the Promotion of Science (JSPS).

References

1. Amazon Web Services. http://aws.amazon.com/. Accessed on 2011
2. Axis. http://ws.apache.org/axis/java/. Accessed on 2011
3. Barth A, Jackson C, Reis C (2008) Security architecture of the chromium browser. Stanford technical report
4. Claudia S (2006) Service granularity in SOA-projects: a trade-off analysis. Master's thesis, Electrical Engineering, Mathematics and Computer Science, University of Twente
5. Clements P, Northrop L (2002) Software product lines: practices and patterns. Addison-Wesley, Boston
6. Creating Web Clip Widgets. http://www.apple.com/pro/tips/webclip.html. Accessed on 2011
7. Creative Commons. http://creativecommons.org/. Accessed on 2011
8. Cross-Origin Resource Sharing. http://www.w3.org/TR/cors/. Accessed on 2011
9. Daniel F, Matera M (2009) Turning Web applications into mashup components: issues, models, and solutions. In: The proceedings of the 9th international conference on web engineering, San Sebastián, pp 45–60
10. Daniel F, Yu J, Benatallah B, Casati F, Matera M, Saint-Paul R (2007) Understanding UI integration: a survey of problems, technologies, and opportunities. Internet Comput 11(3):59–66
11. Dapper. http://www.dapper.net. Accessed on 2011
12. Ennals R, Garofalakis M (2007) MashMaker: mashups for the masses. In: The proceedings of the 33th SIGMOD international conference on management of data, Beijing, pp 1116–1118
13. Fujima J, Lunzer A, Hornbaek K, Tanaka Y (2004) C3W: clipping, connecting and cloning for the Web. In: The proceedings of the 13th international world wide web conference, Manhattan, pp 444–445

14. Google Docs. http://docs.google.com/. Accessed on 2011
15. Google Maps API. http://code.google.com/apis/maps/. Accessed on 2011
16. Google Mashup Editor. http://editor.googlemashups.com. Accessed on 2011
17. Haesen R, Snoeck M, Lemahieu W, Poelmans S (2008) On the definition of service granularity and its architectural impact. In: The proceedings of the 20th international conference on advanced information systems engineering, Montpellier
18. Han H, Tokuda T (2008) A method for integration of Web applications based on information extraction. In: The proceedings of the 8th international conference on web engineering, Yorktown Heights, pp 189–195
19. Han H, Tokuda T (2008) WIKE: a Web information/knowledge extraction system for Web service generation. In: The proceedings of the 8th international conference on web engineering, Yorktown Heights, pp 354–357
20. Han H, Guo J, Tokuda T (2010) Deep mashup: a description-based framework for lightweight integration of Web contents. In: The proceedings of the 19th international conference on world wide web, Raleigh, pp 1109–1110
21. Hickson I, Hyatt D Html 5 working draft – cross-document messaging. http://www.w3.org/TR/html5/comms.html. Accessed on 2011
22. IBM Mashup Center. http://www-01.ibm.com/software/info/mashup-center/. Accessed on 2011
23. iframe. http://www.w3.org/TR/html4/present/frames.html. Accessed on 2011
24. Jackson C, Bortz A, Boneh D, Mitchell JC (2006) Protecting browser state from Web privacy attacks. In: The proceedings of the 15th international conference on world wide web, Edinburgh, pp 737–744
25. Java Portlet. http://www.jcp.org/en/jsr/detail?id=286. Accessed on 2011
26. Koseki Y, Sugiura A (1998) Internet scrapbook: automating Web browsing tasks by demonstration. In: The proceedings of the 11th annual symposium on user interface software and technology, San Francisco, pp 9–18
27. Kulkarni N, Dwivedi V (2008) The role of service granularity in a successful SOA realization a case study. In: The proceedings of the 2008 IEEE congress on services, Honolulu
28. Kwok T, Nguyen T, Lam L (2008) A software as a service with multi-tenancy support for an electronic contract management application. In: The proceedings of the 2008 IEEE international conference on services computing, Honolulu
29. Maximilien EM, Wilkinson H, Desai N, Tai S (2007) A domain-specific language for Web APIs and services mashups. In: The proceedings of the 5th international conference on service-oriented computing, Vienna, pp 13–26
30. Microsoft Popfly. http://www.popfly.com. Accessed on 2011
31. Mietzner R, Metzger A, Leymann F, Pohl K (2009) Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications. In: The proceedings of the 2009 ICSE workshop on principles of engineering service oriented systems, Vancouver, pp 18–25
32. Nitu (2009) Configurability in SaaS (software as a service) applications. In: The proceedings of the 2nd India software engineering conference, Pune
33. Papazoglou MP, van den Heuvel W-J (2006) Service-oriented design and development methodology. Int J Web Eng Technol 2(4):412–442
34. Perepletchikov M, Ryan C (2011) A controlled experiment for evaluating the impact of coupling on the maintainability of service-oriented software. IEEE Trans Softw Eng 37(4):449–465
35. Pettersson U, Jarzabek S (2005) Industrial experience with building a Web portal product line using a lightweight, reactive approach. In: The proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on foundations of software engineering, Lisbon, pp 326–335
36. Reldin P, Sundling P (2007) Explaining SOA service granularity: how IT-strategy shapes services. Master's thesis, Department of Management and Engineering Industrial Economics, Institute of Technology, Linkoping University

37. Representational State Transfer. http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. Accessed on 2011
38. Sabbouh M, Higginson J, Semy S, Gagne D (2007) Web mashup scripting language. In: The proceedings of the 16th international conference on world wide web, Banff, pp 1035–1036
39. Sengupta B, Roychoudhury A (2011) Engineering multi-tenant software-as-a-service systems. In: The proceedings of the 3rd international workshop on principles of engineering service-oriented systems, Waikiki, Honolulu, Hawaii
40. Simple Object Access Protocol. http://www.w3.org/TR/soap/. Accessed on 2011
41. Stollberg M, Muth M (2009) Service customization by variability modeling. In: The proceedings of the 2009 international conference on service-oriented computing, Stockholm
42. Tatemura J, Sawires A, Po O, Chen S, Candan KS, Agrawal D, Goveas M (2007) Mashup feeds: continuous queries over Web services. In: The proceedings of the 33th SIGMOD international conference on management of data, Beijing, pp 1128–1130
43. Thio N, Karunasekera S (2007) Web service recommendation based on client-side performance estimation. In: The proceedings of the 18th Australian software engineering conference, Melbourne, pp 81–89
44. Wang G, Yang S, Han Y (2009) Mashroom: end-user mashup programming using nested tables. In: The proceedings of the 18th international conference on world wide web, Madrid, pp 861–870
45. Web Application Description Language. https://wadl.dev.java.net/. Accessed on 2011
46. Web Service Definition Language. http://www.w3.org/TR/wsdl. Accessed on 2011
47. Web Services for Remote Portlets. http://www.oasis-open.org/committees/wsrp/. Accessed on 2011
48. Wohlstadter E, Li P, Cannon B (2009) Web service mashup middleware with partitioning of XML pipelines. In: The proceedings of 7th international conference on web services, Los Angeles, pp 91–98
49. Wong J, Hong JI (2007) Making mashups with marmite: towards end-user programming for the Web. In: The proceedings of the SIGCHI conference on human factors in computing systems, San Jose, pp 1435–1444
50. XSL Transformations. http://www.w3.org/TR/xslt20/. Accessed on 2011
51. Yahoo Pipes. http://pipes.yahoo.com/pipes/. Accessed on 2011
52. Ye P, Peng X, Xue Y, Jarzabek S (2009) A case study of variation mechanism in an industrial product line. In: The proceedings of the 11th international conference on software reuse: formal foundations of reuse and domain engineering, Falls Church, pp 126–136
53. Yu J, Benatallah B, Saint-Paul R, Casati F, Daniel F, Matera M (2007) A framework for rapid integration of presentation components. In: The proceedings of the 16th international conference on world wide web, Banff, pp 923–932
54. Zhao Q, Huang G, Huang J, Liu X, Mei H (2008) A Web-based mashup environment for on-the-fly service composition. In: The proceedings of 4th international symposium on service-oriented system engineering, Jhongli, pp 32–37
55. Zibuschka J, Herbert M, Rossnagel H (2010) Towards privacy-enhancing identity management in mashup-providing platforms. In: The proceedings of the 24th annual IFIP WG 11.3 working conference on data and applications security and privacy, Rome, pp 273–286