Web application integration at the presentation and logic layers is an appealing approach to enhancing Web services technologies. However, it’s certainly not without challenges.

Internet and Web services technologies continue to advance, and so too must the underlying Web service APIs that support them. Written in a machine-readable format known as Web Services Description Language (WSDL), Web service APIs are specifications that Web applications follow to communicate with each other; the most common architectures used for these are SOAP and Representational State Transfer (REST).

Using mashups—widely considered the most appealing Web services technology for aggregating content into a new application—users can view diverse Web content in an integrated manner. Yahoo Pipes, Microsoft Popfly, Google Mashup Editor, and IBM Mashups were proposed as composition tools for aggregating, manipulating, and mashing up Web services from different websites by using join, select, and map operations.

However, there’s still no uniform interface to access the data, computations (application logic), and user interfaces provided by different kinds of Web content. Web content fed into most of today’s mashup applications is typically obtained from third-party sources via Web service APIs, and the integration is principally restricted to open API websites, which are limited to Web service developers and providers.

**BEYOND APIs**

But what about information reuse beyond current open APIs? Reusing UI components to integrate general Web application content and functionalities at the presentation and logic layers continues to be an important issue. In this way, content is dynamically generated or pulled in response to an end user request. Usually achieved through server-side logic processing or client-side scripting (see Figure 1), this dynamic content functionality enables users to interact with UI components, such as drop-down lists or text input fields to complete the request/response message exchange.

**Tools**

Facilitating mashup application construction without open API restrictions, end-user programming tools such as Mashroom (G. Wang et al., “Mashroom: End-User Mashup Programming Using Nested Tables,” *Proc. 18th Int’l Conf. World Wide Web* [WWW ’09], ACM, 2009, pp. 861-870), Dapper (http://
open.dapper.net, and Marmite
(J. Wong et al., “Making Mashups
with Marmite: Towards End-User
Programming for the Web,” ACM
Ann. Conf. Human Factors in Com-
puting Systems [HFCS 07], ACM,
2007, pp. 1435-1444) use basic
screen-scraping operators (such as
the XPath pattern matcher) to inte-
grate the extracted webpage content.
These tools provide great flexibil-
ity because they solve the basic
problems of—as well as meet the
requirements for—UI-component-
based integration.

Availability
Typically, reusable Web applica-
tions meet some basic standards
and are then wrapped as services.
However, sometimes providers can’t
reuse the exact user-desired service.
The Web search engine in Figure 1
shows an alternative approach in
which reuse is based on retrieved UI
components at a finer-grained level,
not just by picking up only the most
developed services. For example,
the legacy Web application, originally
not released as a Web service, could be
reused at the component level. The idea
is similar to CBSD (component-based
software development), although the
mashup technique beyond open
APIs is based on components from
Web applications.

Dynamic content
Rich client applications—which
are similar to graphics-driven
desktop applications and offer users a
better visual experience—bring ease of
operation and user-friendly interfaces
through DHTML (dynamic HTML)- and
DOM-scripting-technology-generated
interactive webpages. Widespread
use of the hide-and-display method
(H. Han et al., “Deep Mashup: A
Description-Based Framework for
Lightweight Integration of Web Con-
tent,” Proc. 19th Int’l Conf. World
Wide Web [WWW 10], ACM, 2010,
pp. 1109-1110) enables the original
script execution environment and
webpage visibility control to display
the target parts—which can’t run
normally after conventional extrac-
tion methods—removing non-target
partial webpage information. Di-
verse UI components, internal or
external scripting functions, and
even active Adobe Flash players
in webpages can be integrated
effectively.

Personalization
Web service response and ex-
ttracted partial webpage information
are usually (re)formatted in XML.
Template processors such as XSLT
are provided to transform XML data
into HTML or XHTML documents.
Various templates are available for
users to construct personalized
layouts.

Adaptability
Simply put, personalization means
adapting services according to a user
profile. To further empower the new
mashup application with self-adapted
capability, two prerequisites must be
fulfilled, as shown in Figure 1. First,
a large number of UI candidate compo-
nents from Web resources are
required. For example, in weather
forecasting, components for general
users are provided at such sites as
Weather.com, but components from
Yahoo.com are also supported if
cookie history indicates that the user
frequently accesses Yahoo.

Second, a mashup application should
be equipped with some variability-
supporting techniques, such as XML
or JSON. Similar to personalization, the
XSLT library could implement a flexible
layout of the resulting page. Usually,
different parameters enable UI compo-
nents to behave differently—this is the
variability provided by the UI compo-
nents themselves. The abundance
of Web applications and advanced
techniques enable the two prerequi-
sites to be fulfilled.

Integration testing
Because of the heterogeneity
and implementation transparency
associated with Web applications,
the model-based integration testing
approach is widely used in Web-UI-
component-based mashups. The
Component Interaction Graph (a
graph describing the interrelation
of components), which addresses
relationships such as messaging,
usage, and so on, is the core model.
The graph can be designed to
model UI component interactions.
by depicting interaction scenarios, and it can also be used to develop test adequacy criteria. Interface and event testing ensure that every interaction between two UI components is tested.

**NEW CHALLENGES**

In terms of systematic reuse and integration of large-scale Web content, as shown in Figure 1, mashup applications face additional technical challenges during the overall workflow, depending on the developmental need. For example, a strict security policy must be employed for online payment processing, and copyright transfer and clearance is an inevitable issue for public mashup applications.

**Security**

Users’ browsing activities when using mashups could subject them to security risks, including malware infection and privacy leakage. In addition, all “programs” from different origins in the OS layer are allocated in a single protection domain in memory, and a crashed part would cause all other parts to crash. In the application layer, should users want interactive mashups in which content interacts with other content from differing origins, users must utilize specific security policies and mechanisms to enhance browsers’ Same Origin Policy (SOP) to prevent malicious behaviors from different components such as authentication abuse and script injection.

**Extraction stability**

XPath-based partial information extraction is usually based on the fact that the responding webpages always use identical or similar layouts. If the responding webpages use different layouts, the extraction precision is low because the targeted parts vary with webpage layouts. Moreover, website layouts are updated irregularly and not on any predictable schedule.

**Availability**

Many Web resources, such as static webpages that aren’t officially qualified to be “services,” can be considered as reusable components. However, some of these pages might later become unavailable. Compared with open services provided by major vendors, UI components are more vulnerable to website changes, and could therefore disrupt mashup content. Furthermore, the unpredictability of the components due to evolution poses another availability challenge. Therefore, we propose that the mashup application be empowered to look up the backup UI components when some existing components suddenly disconnect.

**Adaptability**

We’re optimistic about the potential for adaptability: some UI components support different behaviors according to various input parameters, which are configuration options set by users. But this is also a challenge because many UI components still don’t support variety in configuration options. Because we’re only reusing these UI components for data exchange, we might not be able to change their source code to support adaptability. Thus, the difficulty is that we can access these UI components, but we can’t extend them.

**Granularity**

To better establish the flexible integration of third-party UI components and publishable reuse services, service developers should complete granularity trade-off analysis.

Service granularity typically refers to service size. Here, granularity is how big the integrated new service from several UI components should be. That services should be large-sized or coarse-grained is often postulated as a fundamental design principle of applications in service-oriented architecture, or SOA (R. Haesen et al., “On the Definition of Service Granularity and Its Architectural Impact,” Proc. 20th Int’l Conf. Advanced Information Systems Eng., Springer, 2008, pp. 375-389). Establishing a balance between granularity and maintenance effort supports the sharing and reuse of the newly developed service with other users and developers.

**Retrieval**

Using search engines such as Google or Bing, users can review results pages to determine whether they contain suitable Web components or not; they can then compare components to select the one that’s most suitable. This conventional retrieval method is inefficient (even ineffective) because current general search engines mainly use a content-oriented search mechanism, so functionality information is beyond the analysis range. Moreover, these non-functionality-oriented search result rankings can’t satisfy the functionality-oriented retrieval requirements and so require time-consuming manual verification and comparison.

**Copyright**

The challenge around legal and practical reuse of Web content can’t be left off our list. Open Web service APIs usually have a detailed copyright policy for users. Compared with open APIs, general Web applications don’t provide clear end-use-programming-oriented interfaces. Most intellectual property, such as photographs, Web news, blogs, and map information, is protected by copyright. For this type
Continued development in this important area—despite some clearly daunting challenges—has the potential for a more customizable and dynamic Internet experience.

Acknowledgments

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).

Hao Han is a research associate in the Department of Information Science at Kanagawa University, Japan. Contact him at han@computer.org.

Yinxing Xue is a research fellow in the Temasek Lab at National University of Singapore. Contact him at tslxuey@nus.edu.sg.

Keizo Oyama is a professor at the National Institute of Informatics (NII) and the Department of Informatics in the Graduate University for Advanced Studies (SOKENDAI), Japan. Contact him at oyama@nii.ac.jp.

Editor: Mike Hinchey, Lero—the Irish Software Engineering Research Centre; mike.hinchey@lero.ie

Selected CS articles and columns are available for free at http://ComputingNow.computer.org.