# Towards Service Composition Based on Mashup

Xuanzhe Liu[1,2] Yi Hui[3] Wei Sun[3], Haiqi Liang[3]
[1]*School of Electronics Engineering and Computer Science, Peking University*
[2]*Key Laboratory of High Confidence Software Technologies (Peking University)Ministry of Education*
*Beijing, China, 100871*
[3]*IBM China Research Lab, Beijing, China, 100094*
liuxzh@sei.pku.edu.cn, {huiyi weisun, lianhq}@cn.ibm.com,

## Abstract

*Mashup is a hallmark of Web 2.0 and attracts both industry and academia recently. It refers to an ad hoc composition technology of Web applications that allows users to draw upon content retrieved from external data sources to create entirely new services. Compared to traditional "developer-centric" composition technologies, e.g., BPEL and WSCI, mashup provides a flexible and easy-of-use way for service composition on web. It makes the consumers free to compose services as they wish as well as simplifies the composition task. This paper makes two contributions. Firstly, we propose the mashup architecture, extend current SOA model with mashup and analyze how it facilitates service composition. Secondly, we propose a mashup component model to help developers leverage to create their own composite services. A case study is given to illustrate how to do service composition by mashup. This paper also discusses about some interesting topics about mashup.*

## 1. Introduction

Service Oriented Architecture (SOA) has brought a new paradigm and technology revolution to traditional software development. It utilizes services as basic constructs and introduces three different roles, service provider, service consumer and service broker as well as their relationships. Services in SOA are "loosely coupled": they are developed and hosted by different providers, described in specific standard interface (e.g., WSDL), published in an accessible registry (e.g., UDDI), and can be discovered and requested via standard protocols (e.g., SOAP).

SOA is changing software development approach from traditional "product-centric" manufacturing, to "consumer-centric" service composition. Supported by SOA service composition technologies, e.g. BPEL [6], WSCI [5], new business process, application or solution can be built in a relatively rapid and low-cost way through the composition of distributed services even in heterogeneous environments.

However, current service composition programming model and tools are mainly designed for professional SOA developers to build SOA software or solution so as to solve business problem in enterprise's complex IT environment. Though these techniques are very powerful to address enterprise SOA problem, there are three important issues existing. First of all, these technologies involve relatively strong requirements overhead about developer's skill and supporting infrastructure. SOA developers usually need to spend major effort to master many SOA technologies, e.g. BPEL, WSDL, SCA (Service Component Architecture), as well as tools, e.g. design-time IDE tools, and runtime middleware servers (SCA server or BPEL server). Most of these tools demand for major investment on hardware and software infrastructure; Secondly, these technologies can not support service composition's customization on the fly. The SOA service composition design, development and testing are usually conducted in IDE tool first according to customer requirements, and then deployed on runtime server. After deployment, the composition logic can hardly be customized easily according to the changes of composite service consumer's requirements, as this involves a long lifecycle from design /development /testing to deployment. Finally, these technologies cannot well support the composition of legacy or existing web applications which don't or can't provide web service interfaces.

These issues have become the barriers for faster and wider adoption of SOA, especially in the promising Web 2.0 paradigm. As well known, Web 2.0 refers to the next generation of web applications defined by T. O'Reilly. The core Web 2.0 principles are "simple, low-barrier and fast" and "every user himself is the center on internet"[6]. It tries to maximize the consumer's creativity for new services so even "grandma" can easily accomplish an application as she wishes. Therefore, we need new SOA service composition technologies in Web 2.0 paradigm (including programming model and its corresponding tools) for users with very low programming skill requirements, which should address end-user's needs on flexible composition and its customization of services/data/applications within enterprise or on the Web. Specifically the new technologies should support:

2007 IEEE Congress on Services (SERVICES 2007)
0-7695-2926-7/07 $25.00 © 2007 IEEE

- Leveraging web as the design-time and runtime tool for service composition, so as to significantly reduce overhead to composite service consumers
- On-the-fly customization and deployment to make the service composition to be more responsive for consumer's requirement changes
- Easy reuse and remix of existing applications and data which can be accessed through the web.

The emergence of web 2.0 related technologies, e.g. REST, AJAX (Asynchronous JavaScript and XML), Wiki, provides opportunities for meeting these requirements. There is a web 2.0 concept called "mashup", which allows consumers to draw upon content retrieved from external data sources (web services, data store, web application) to create entirely new and innovative services [7]. In our opinion, mashup essentially introduces a much simpler, more cost-effective, self-served approach for service composition, that significantly reduces the complexity and barriers of SOA service composition Therefore, every consumer can compose his/her own service applications only by "drag and drop" action within a web browser. Obviously, mashup is an extremely "consumer-centric" and lightweight service composition technology, which would be more applicable to all consumers all over internet in Web 2.0 paradigm.

As a new emerging concept, mashup has attracted both industry and academia. This paper tries to explore how mashup facilitates service composition and related technical issues. It is organized as follows: In Section 2, we are going to analyze some concepts brought by mashup, including mashup architecture, SOA extension with mashup and mashup based service composition. Section 3 proposes our mashup component model and runtime mechanisms; Section 4 presents a mashup case study; Section 5 discusses about the related work; Section 6 discusses about the some interesting topics about mashup and the paper ends in Section 6 on the conclusion and future work.

## 2. Analysis of Mashup

### 2.1 Background for Mashup

The term mashup started in the audio domain, referring to artists remixing two (or more) recordings into a new entity. The founding example is the Grey album – a mashup of The Beatles' White Album and Jay-Z's black album [9]. Where the music industry has had mixed reactions to mashups (suit was predictably brought against the Grey album. Wikepedia defines mashup as: a website (URL) or web application that uses content from more than one source to create a completely new service [17].

Mashup stems from the emphasis on interactive user participation manner in which they aggregate and stitch together third-party data. It is derived of the evolution of internet computing technology, especially popularity of web 2.0. Web 2.0 makes the webpage no longer a "static" markup document (e.g., HTML), but more "dynamic" interactive data application that can be consumed (by RSS, REST or ATOM). For AJAX and Rich Internet Application (RIA) change the webpage manipulation from DOM (Document Object Model) to Widget (e.g., DOJO [11]), the data contained in webpage is organized in more componentized manner. Furthermore, with these "web components", the consumers can even accomplish some business logics in the web browser instead of accessing the layer residing in the server side. As more and more enterprises enable commercial RSS/ATOM/REST support in their services, mashup even spreads roots across the Web, drawing upon content and functionality retrieved from data sources that lay outside of its organizational boundaries.

### 2.2 Mashup Architecture

Generally speaking, mashup is usually done at the web browser, by "drag and drop" applications from different sources together. However, there must be some backend infrastructure to support mashup. From D.Merrill [7], mashup application is architecturally comprised of three different participants: API/content providers, the mashup hosting site, and the consumer's Web browser, which is very similar to the popular three-tier architecture. The architecture is shown in Figure 1.

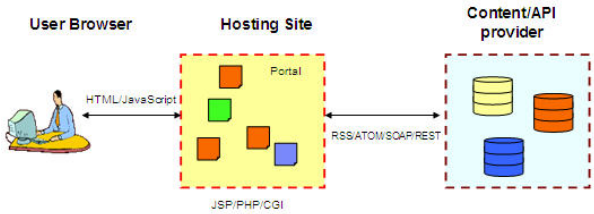


Figure 1 Mashup Architecture

- ***The API/content providers***. These are the (sometimes unwitting) providers of the content being mashed-up. To facilitate data retrieval, providers often expose their content through Web-protocols such as REST, Web Services, and RSS/Atom. However, many interesting potential data-sources do not (yet) conveniently expose APIs. Mashups that extract content from sites like Wikipedia, TV Guide, and virtually all government and public domain Web sites do so by a technique known as screen scraping [7]. In this context, screen scraping connotes the process by which a tool attempts to extract information from the content provider by attempting to parse the provider's Web

pages, which were originally intended for human consumption.

- ***The mashup hosting site.*** It is where the mashup is hosted. Just because this is where the mashup logic resides, it is not necessarily where it is executed. On one hand, mashups can be implemented similarly to traditional Web applications using server-side dynamic content generation technologies like Java servlets, CGI, PHP or ASP. Alternatively, mashed content can be generated directly within the client's browser through client-side scripting (e.g., JavaScript) or applets. This client-side logic is often the combination of code directly embedded in the mashup's web pages as well as scripting API libraries or applets (furnished by the content providers) referenced by these Web pages. Mashup using this approach can be termed rich internet applications (RIAs), meaning that they are very oriented towards the interactive user-experience. The benefits of client-side mashing include less overhead on behalf of the mashup server (data can be retrieved directly from the content provider) and a more seamless user-experience (pages can request updates for portions of their content without having to refresh the entire page). The Google Maps API is intended for access through browser-side JavaScript, and is an example of client-side technology. Often mashup uses a combination of both server and client-side logic to achieve their data aggregation.
- ***The consumer's Web browser***. It is where the application is rendered graphically and where user interaction takes place. As described above, mashups often use client-side logic to assemble and compose the mashed content.

## 2.3 SOA Extension with Mashup

Mashup is essentially a process that integrates data/content from different sources on the internet. Therefore, it is essentially a service composition style from SOA perspective. Considering the mashup architecture and roles in the last section, we extend the basic SOA roles (provider, broker and consumer) according to the mashup architecture in the last section.

- Mashup Component Builder (MCB): services must be provided by several providers, each of whom has its own specification and usage. For example, the services may be web services, Enterprise JavaBeans or legacy systems. The first trouble for mashup is the interoperability of these services. On the other hand, mashup is mainly the composition at UI level, while current composition at logic level. Here, the Mashup Component Model is the extension of service provider. MCB selects services from the Service Catalogue (e.g., UDDI or from screen scraping) and takes the responsibility to encapsulate all the services in a standard component model with UI presentation (will be discussed later). Then the Mashup Component is then published to a repository.
- Mashup Server: the mashup server involves three components. The Service Catalogue is extremely the UDDI server to publish services from providers. The Mashup Component Repository stores all mashup components developed by the MCB. The Monitoring provides the performance evaluation (such as dashboard) of the mashup components. It ensures the consumers choose the proper mashup components and the providers can reconfigure those inadequate ones.
- Mashup Consumer: the mashup consumers select the proper mashup components from the mashup server, compose their own application in the browser.

Then the mashup lifecycle can be illustrated in Figure 2. Obviously, the mashup is the extension of current SOA paradigm. It does not break the basic SOA principles (services are self-described and loosely coupled) while making services more easily and visually utilized by consumers.

The core issue here is the mashup component. Based on traditional service concept (such as web services), mashup component enables the interoperability between different service providers at UI level.

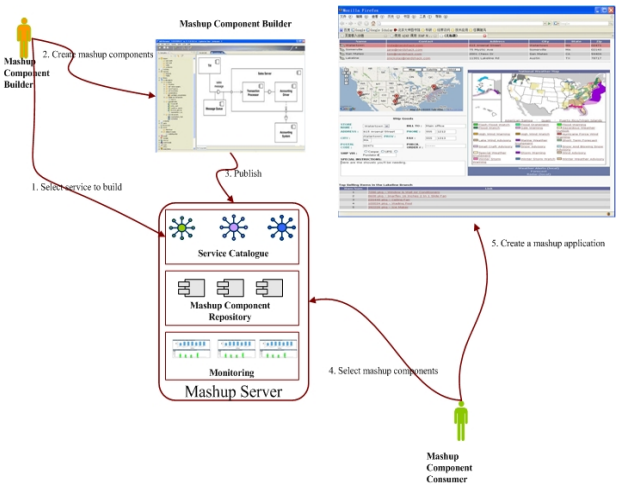


Figure 2 SOA Extension with Mashup

## 2.4 Mashup and Service Composition

As mashup is relatively new concept, it may be confusing to distinguish mashup and traditional service composition. From our point of view, composite services are more an assembly of existing services than "green

field" development [9]– they are done at interface level and don't have to be Web-based. There is the ever increasing formality of composite applications, which are typically based on SOAP Web services and frequently woven together with BPEL and developed by professional programmers. Such composition style is mainly at "interface" level instead of "application" level, which means the user should have full comprehension of service interfaces. It is especially too hard to those consumers without professional knowledge. Composite applications also tend to use an older generation of programming languages and technologies that have more overhead and ceremony. And, there is too much exposed plumbing and infrastructure.

Mashup, on the other hand, uses "almost remarkably simple, basic techniques for connecting things together" [9]. This includes guerilla-style development techniques that deliver results in preference to formal, upfront engineering. We summarize the mashup core features as follows:

- ***More Reusable***: Compared to current SOA composition technologies, such as BPEL and WSCI, mashup is more "coarse-grained" at the application level. Each mashup building block (mash up API), has its own context and business logic, usually, mash up APIs contain most of common business logic, and it is a combination of data, process and UI. Therefore, the mashup services are more reusable.
- ***Web-Based***: This means using Javascript includes of another site's software, straightforward Web services and feeds based directly on top of HTTP, and JSON (JavaScript Object Notation) for data retrieval and remixing [9]. And with initiatives like OpenAJAX, we might get first real conventions for component interoperability in the browser."
- ***Light Weight:*** Additionally, mashup is a lightweight tactical integration of multi-sourced applications or content into a single offering. Because mashup leverages data and services from public Web sites and Web applications, they're lightweight in implementation and built with a minimal amount of code. Their primary business benefit is that they can quickly meet tactical needs with reduced development costs and improved user satisfaction.
- ***End Consumer Centric:*** mash up is supposed to support programming for end consumer, not developer, without complex programming environment. Every consumer can compose his/her own service applications only by "drag and drop" action within a web browser.

# 3. Mashup Component Model and Runtime

## 3.1 Mashup Component Model

From the mashup view, web is no longer represented as a markup document, but a data driven application. Therefore, there must be a well-defined component model that can encapsulate the data from multiple sources and manipulate the existing web resources through the standard services (REST, ATOM/RSS, and so on).

In our opinion, a mashup component is a module which consists of a set of UI components and a set of backend services (either local or remote) binding into the UI components. The mashup hosting environment should provide a "container" for the component and take the overall control of process logic, send and receive data from external services and route it to the corresponding components. We classify the component model into three elements, as shown in Figure 3:

- **UI Component**: UI component represents as a set of widgets in the browser (a window, a button, a drop-down list, etc). Enhanced by AJAX, UI components and its binding service component can be connected and updated dynamically. UI component masks the service components details to the consumer so as the composition is done at UI level. In other words, to consumers, UI component is the unique entity that survives in the mashup applications.
- **Service Component**: Service component represents data manipulation interface which will contain the data content, for example, a web service interface, which can be accessed by SOAP and REST; or it can be a DB interface, which can retrieve and store data in local or remote database. Data standardization is achieved in simple script by web container or service container. In our current implementation, the service component is mainly the web services or services with open APIs (such as GoogleMap).
- **Action Component**: Action component acts like the connector between UI components and service components. For example, it defines an action driven by events (e.g., onClick or onMouseOver). It can be an action which changes the display value of a UI component, or one that invokes a service component interface.

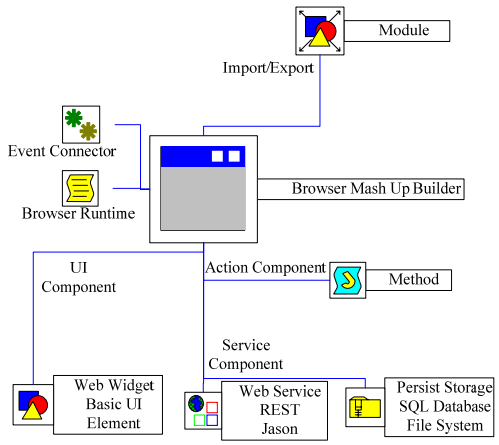


Figure 3 Mashup Component Model

### 3.2 Mashup Component Runtime

All mashup lifecycle resides in the web browser. At runtime, a mashup component is instantialized as a JavaScript class. Once the consumers select the mashup component (UI component exactly) and drag it to the browser, a piece of .js code will inserted into the HTML document.

Figure 4 shows the sequence diagram of mashup runtime. The UI component is a set of widgets described in HTML, and register itself to the web browser. It is because once the user operates the UI component (fill in text or submit a form), all the data are processed by browser. In our current implementation, the requests are sent to service components in SOAP, we apply the AJAX pattern to request, parse and receive the response. So the web browser rails a DOM event and then generates the XMLHTTPRequest object, which is the core in AJAX.

A core trouble here is the XML parsing by XMLHTTPRequest in different browser. Invoking web services in the browser needs the XML parsing nested in browser itself. However, to cope with the multiple naming spaces in SOAP message, the ability of browser differs from one another. In Mozilla Firefox, parsing XML to a DOM tree is easy (as Firefox provides getElementsByTagNameNS() to retrieve XML to DOM). However, in Internet Explorer, it needs the signature of JavaScript and involves some troubles. So we extend the XMLHTTPRequest to implement the Action Component so as it can process SOAP request/response in different browser. The Action Component provides a function invokehandlers that processes the web service call, SOAP request/response and transport, as the piece of code following:

```
InvokeHandlers = function (call, envelope, transport, state) {
  this.each(
    function(value) {
      switch(state) {
        case 'request':
          try {
            value.on_request(call,envelope, transport);
          } catch(e) {}
          break;
        case 'response':
        try{
         // process response message
         }catch(e){}
        case 'error':
        try{
         // process error message
         }catch(e){}
      }
  }
```

Besides XMLHTTPRequest, the Action Component

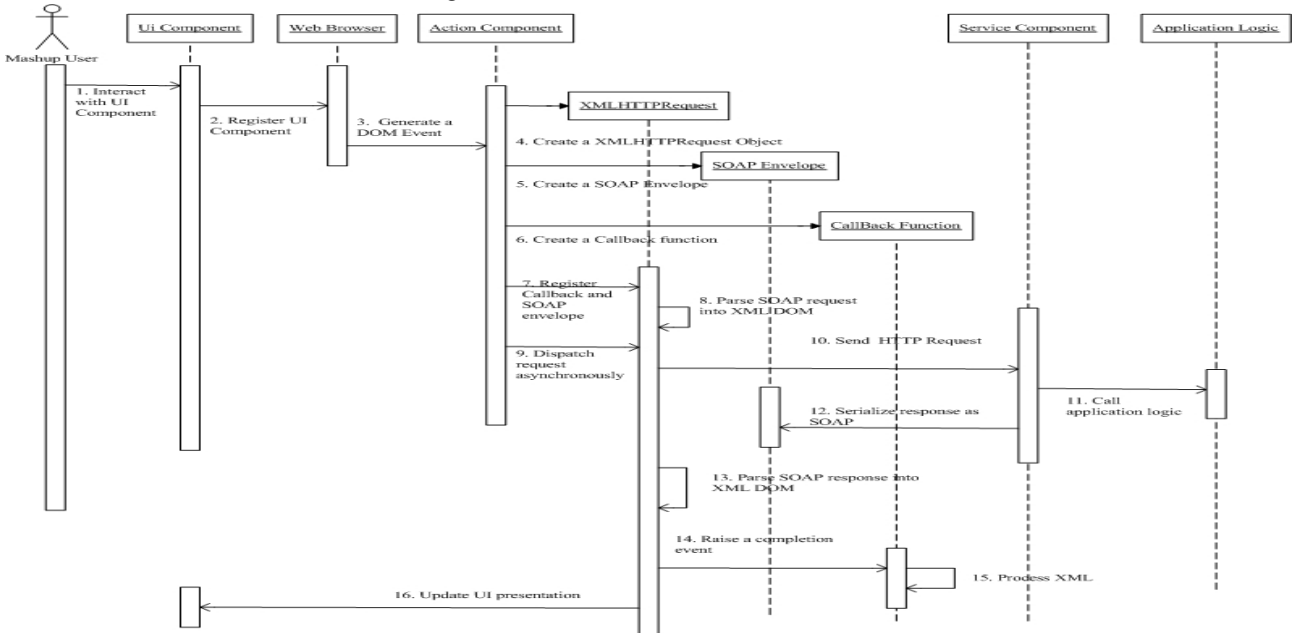


Figure 4 Sequence Diagram of Mashup Component Runtime

creates a SOAP Envelope and a callback function as well. Both should register to XMLHTTPRequest for further use. Once XMLHTTPRequest parses the SOAP envelope, the Action Component inserts a add_hanlder() function to intercept the SOAP to invokehandlers, and invokes the invoke() function to send the request to the service component. After the service component finishes the application logic, it serializes the results in the SOAP response. Then Action Component invokes XMLHTTPRequest parses the SOAP DOM and invoke() invokes the callback function to update the UI component. The invoke() function is as follows:

```
invoke: function(envelope, callback) {
  this.invokeHandlers(this,envelope,null,'request');
 //initialize the invokeHandler…


  SOAP.Envelope(xml.documentElement);
  call.invokeHandlers(call,responseEnv,transport, 'response');
  callback(this, responseEnv, transport.responseText);
  else {
         call.invokeHandlers(call,null,'error');
       }
      catch(e)
      {
       call.invokeHandlers(call,e,'error');
      }
    };
  // Process the SOAP response
```

As discussed before, users only focus on UI level composition, so the data exchange between two mashup components is processed by their own Action Components. Action Components is a piece of JavaScript code and cannot directly pass the data type to SOAP envelope, so we use JSON [12] as data wrapper. JSON is a lightweight data exchange format. It is easy for humans to read and write and for machines to parse and generate.

## 4. Case Study

In this section, we will show a case study to illustrate how quickly achieve self-service composition for a small ERP system by mashup. The ERP system provides the information of procurement or logistic department, for example, the order profile (shipping no, shipping vender, shipping date etc.). But it does not provide the service that tracks the order status. To help the users get the status view, we are going to integrate FedEX service. FedEX[13] is a global corporation that provides services to customers and businesses worldwide with a broad portfolio of transportation, e-commerce and business services. It can help get the real-time shipment status. On the other hand, we will mashup Yahoo map service and FedEX for the consumers to know the shipment locations.

First, we create a mashup component for ERP service. We name it the "*eManager*", as shown in Figure 5. Then we drag and drop the FedEX web services as another one and create an action component in the mashup builder by adding a line of scripts:
*FedexTracking.Search($("eManager.ShippingNo").value)*

This line attaches the action by which the *eManager* module can submit the shipping number to FedEX service.


Figure 5 Order Profile

Once the user interacts with *eManager* by submitting the shipping number, the Action Component will trigger the event and invoke the *FedexTracking.Search* interface. Then *FedexTracking* module will invoke the web service and return the tracking information (Status, Longitude & Latitude) of the shipments, as shown in Figure 6. We also provide the SOAP Monitoring Service to intercept the real time SOAP message.

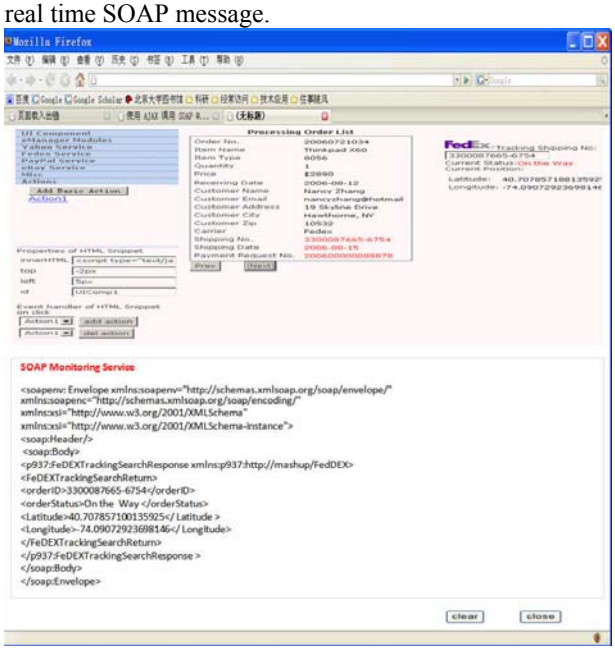


Figure 6 Mashup of FedEX service

Finally, with the similar mechanism, we add the Yahoo map service and display the map of the shipments position based on the latitude & longitude the FedEX service provided, as shown in Figure 7.

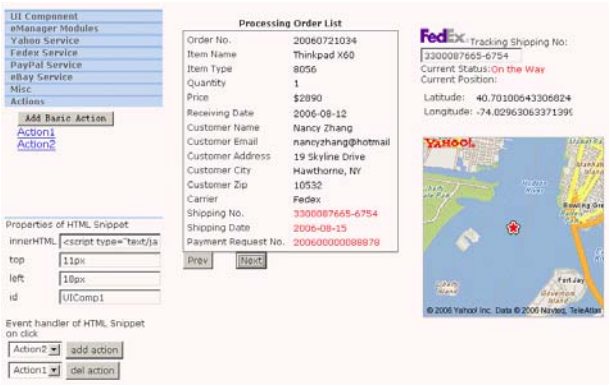


Figure 7 Mashup of Yahoo Map

## 5. Related Works

Mahsup is really a hot topic recently. In nature, it is a web-based composition approach that enables end users to create new applications more easily and quickly. Mashup has aroused attention from industry. For example, KapowTech[15] has already released a series toolkit to support web front-end applications mashup, including the screen scraper, data wrapper and self-process enactment. Even though those famous IT companies, e.g., IBM, SAP and SalesForce, also take mashup as an important approach for the next generation lightweight service composition. IBM has already taken effort on a new collaborative Wiki, which is named QEDWiki (means “Quickly and Easily Done”). Within QEDWiki, every end user can easily integrate existing services (e.g., GoogleMaps), or develop new services by using the widget model. Our work in mashup borrows much from both Kapow and QEDWiki, especially on the component model design. We are now more interested in the runtime management and maintenance for mashup, e.g., we have introduced mashup bus as a lightweight runtime infrastructure to enable the data processing and widget generation.

A big obstacle of mashup is the data retrieval and transformation. In fact, some researchers have begun to study how to use semantic modeling technologies to help ease the problem of automatic reasoning between different data sets [2]. Another problem is how to solve potential danger brought by mashup applications. As mashup integrates services with different data sources and the browser often faces session invalidation, the exception handling is very important.

## 6. Discussion

### 6.1 Highlights of Mashup

The main attraction of mashup is that it has the potential for self-services that end consumers can theoretically create. It also performs composition in the browser. This provides a sort of safe "sandbox" where users can experiment safely with powerful tools without affecting the traditional IT development, deployment, and support processes. And presumably, mashup tools would provide automatic versioning, security, and other needed enterprise software qualities. All of this potentially drops the cost of development enormously because an end consumer could just get together and create, test, and share a mashup in a few hours, instead of the laborious and time-consuming cost of budgeting, architecting, designing, project managing, testing, and maintaining the software using the elite and expensive skills of the IT department.

Another major attraction of mashup is known as the automation dilemma. In today's knowledge worker intensive businesses, rote processes are not the norm and are increasingly automated through various mechanisms today. At this point, however, we need tools that actually enable this way of working; end-user guided creation of software, IT policies that encourage the exposure of corporate information in RSS and XML feeds, and good mashup development tools that literally require no training to use. Also, the Global SOA [7] is becoming larger each and every day, providing all of us, consumers and businesses both, with a powerful inventory of unique services and data to weave into our mashups, if only there is a suitable "loom". Currently we can find web as a model for best practices in this regard, such as widget.

However, mashup currently is useful for small and simple application integration logic. It is not recommended doing complex business process integration by mashup. Gartner warns that, because mashups combine data and logic from multiple sources, they’re vulnerable to failures in any one of those sources [8]. It is because complex business process usually needs collaboration among multi partners instead of a single centric user, and requires powerful and dependable infrastructure to ensure long-run and transactions. As mashup is naturally data aggregation from multi sources and usually done at web browser, once there is session invalidation or network unreachable, the transactional properties cannot be guaranteed.

### 6.2 Challenges to Mashup

The biggest problem of mashup is the data. Mashup developers might also have to contend with several issues that IT integration managers might not, one of which is data pollution. As part of their application design, many mashups solicit public user input. As evidenced in the wiki application domain, this is a double-edged blade: it can be quite powerful because it enables open

contribution and best-of-breed data evolution, yet it can be subject to inconsistent, incorrect, or intentionally misleading data entry. The latter can cast doubts on data trustworthiness, which can ultimately compromise the value provided by the mashup.

Like the traditional enterprise IT managers that find themselves face to the challenge of integrating legacy data sources (e.g., to create cooperate dashboards that reflect current business conditions), mashup developers face the difficulties of deriving shared semantic meaning between the multi-source datasets. Regardless of data missing or incomplete data mappings, they may either discover the data they want to integrate is not suitable, so it needs further process. For example, the customer opportunity record might be entered inconsistently, using common abbreviations for names (such as "CustomerOpp" in one CRM system and "Cus Opp" in another), making automated reasoning about equality difficult, even with good heuristics.

Another host of integration issues facing mashup developers arise when screen scraping techniques must be used for data acquisition. As discussed in the previous section, deriving parsing and acquisition tools and data models requires significant reverse-engineering effort. Even in the best case where these tools and models can be created, all it takes is a refactoring of how the source site presents its content (or mothballing and abandonment) to break the integration process, and cause mashup application failure. Microformat [14] is an emerging technology that enables web pages can be read by both human beings and machines. However, there are only a few microformat specifications currently, and most websites are not using microformats. Screen scraping will be getting easier if websites are following the semantic standards in processing their data over the web.

## 7. Conclusion and Future Work

In this paper, we deeply investigate mashup, which is a web-based service composition that simplifies the consumers to create new applications just by very simple actions. We analyze the technical background of mahsup, introduce the mashup architecture and make careful comparison with current SOA composition approaches. We propose our solution to mashup, design a mashup component model and the runtime mechanisms.

As mashup is a new and immature technology, there are still a lot of problems left. In our future work, we will mainly focus on the runtime management and maintenance of mashup.

## 8. Acknowledgements

This work is supported by the National Natural Science Foundation of China under Grant No 90612011, 90412011, 60403030, the National Grand Fundamental Research 973 Program of China under Grant No. 2005CB321805, the IBM University Joint Study Program and the IBM Ph.D Fellowship Program.

## 9. References

[1]. M. Brambilla, S. Ceri, S. Comai, C. Tziviskou. Exception Handling in Workflow-Driven Web Applications. International Conference on World Wide Web (WWW) 2005, pp170-179, ACM 1595930469. Chiba, Japan, May 10-14, 2005.

[2]. Schraefel, m. c., Smith, D. A., Russell, A. and Wilson, M. L. (2006) Semantic Web meets Web 2.0 (and vice versa): The Value of the Mundane for the Semantic Web. Submitted to The 5th International Semantic Web Conference, Athens, GA, USA.

[3]. Web Service Choreography Interface 1.0 Specification, http://dev2dev.bea.com/techtrack/wsci.jsp , 2002.

[4]. T. Andrews et al. Business Process Execution Language (BPEL), version 1.1. Technical report, BEA Systems and International Business Machines Corporation, Microsoft Corporation, SAP AG and Siebel Systems, May 2003.

[5]. Resource Description Framework (RDF). http://www.w3.org/RDF/

[6]. T O'Reilly. What is Web 2.0--Design Patterns and Business Models for the next Generation of Software. http://scholar.google.com/url?sa=U&q=http://intervention.ch/rebell.tv/presse/O%27Reilly%2520Network_%2520What%2520Is%2520Web%25202.pdf

[7]. Duane Merrill. Mashups: The new breed of Web app--An introduction to mashups. http://www-128.ibm.com/developerworks/xml/library/x-mashups.html

[8]. Gartner's 2006 Emerging Technologies Hype Cycle Highlights Key Technology Themes. http://www.gartner.com/it/page.jsp?id=495475

[9]. Dion Hinchcliffe. The quest for enterprise mashup tools http://blogs.zdnet.com/Hinchcliffe/?p=59

[10]. Dion Hinchcliffe. Web 2.0 and SOA: Contrived or Converging? http://web2.wsj2.com/web_20_and_soa_contrived_or_converging.htm

[11]. http://dojotoolkit.org/

[12]. http://www.json.org/

[13]. http://www.fedex.com/

[14]. http://microformats.org/

[15]. http://www.kapowtech.com/

[16]. http://www.openajax.net/wordpress/

[17]. Wikipedia. Entry for "The Grey Album". Available at: http://en.wikipedia.org/wiki/The_Grey_Album Accessed May 20, 2006..