ISSN:0975-9646

Towards Service Composition Based on Mixture of different things

Dr.I.Lakshmi
Assistant Professor,
Department of Computer Science,
Stella Maris College, Chennai, Tamil Nadu, India

G.D. DhanaLakshmi
Assistant Professor,
Department of Computer Science,
Stella Maris College, Chennai, Tamil Nadu, India

Abstract- Mashup is a sign of Web 2.0 and draws in both industry and the educated community as of late. It alludes to an impromptu synthesis innovation of Web applications that permits clients to draw upon substance recovered from outside information sources to make completely new administrations. Contrasted with conventional "designer driven" piece innovations, e.g., BPEL and WSCI, mashup gives an adaptable and simple of-utilization path for administration organization on web. It makes the customers allowed to create administrations as they wish and in addition disentangles the organization assignment. This paper makes two commitments. Firstly, we propose the mashup building design, expand current SOA model with mashup and break down how it encourages administration arrangement. Besides, we propose a mashup part model to assist engineers influence with creating their own composite administrations. A contextual analysis is given to represent how to do administration arrangement by mashup. This paper additionally talks about some fascinating points about mashup.

1. INTRODUCTION

Administration Oriented Architecture (SOA) has brought another worldview and innovation transformation to conventional programming advancement. It uses administrations as fundamental builds and presents three distinct parts, administration supplier, administration shopper and administration dealer and also their connections. Administrations in SOA are "inexactly coupled": they are created and facilitated by distinctive suppliers, portrayed in particular standard interface (e.g., WSDL), distributed in an open registry (e.g., UDDI), and can be found and asked for through standard conventions (e.g., SOAP).

SOA is changing programming advancement come closer from conventional "item driven" assembling, to "buyer driven" administration structure. Upheld by SOA administration structure advances, e.g. BPEL [6], WSCI [5], new business procedure, application or arrangement can be inherent a moderately quick and minimal effort route through the piece of conveyed administrations even in heterogeneous situations. In any case, current administration organization programming model and devices are for the most part intended for expert SOA designers to construct SOA programming or arrangement in order to take care of business issue in big business' mind boggling IT environment. Despite the fact that these strategies are capable to address venture SOA issue, there are three essential issues existing. As a matter of first importance, these advances include moderately solid prerequisites overhead about designer's ability and supporting base. SOA designers more often than not have to spend real push to ace numerous SOA advancements, e.g. BPEL, WSDL, SCA (Service Component Architecture), and in addition instruments, e.g. outline time IDE devices, and runtime middleware servers (SCA server or BPEL server). The vast majority of these devices interest for real venture on equipment and programming framework; Secondly, these innovations cannot bolster administration creation's customization on the fly.

The SOA administration sythesis configuration, advancement and testing are normally led in IDE apparatus first as per client prerequisites, and afterward conveyed on runtime server. After arrangement, the sythesis rationale can scarcely be redone effortlessly as indicated by the progressions of composite administration purchaser's necessities, as this includes a long lifecycle from configuration/improvement/testing to sending. At last, these innovations can't well backing the sythesis of legacy or existing web applications which don't or can't give web administration interfaces. These issues have turned into the obstructions for speedier and more extensive selection of SOA, particularly in the promising Web 2.0 worldview. Too known, Web 2.0 alludes to the up and coming era of web applications characterized by T. O'Reilly. The center Web 2.0 standards are "basic, low-boundary and quick" and "each client himself is the inside on internet"[6]. It tries to augment the shopper's inventiveness for new administrations so even "grandmother" can without much of a stretch fulfill an application as she wishes. In this way, we require new SOA administration creation advances in Web 2.0 worldview (counting programming model and its relating apparatuses) for clients with low programming ability prerequisites, which ought to address end-client's necessities on adaptable structure and its customization of administrations/information/applications inside of big business or on the Web. Particularly the new advancements ought to bolster:

- ❖ Leveraging web as the design-time and runtime tool for service composition, to reduce (a lot) overhead to (made up of different things) service people (who use a product or service)
- ❖ On-the-fly customization and use/military service to make the service composition to be more (able to reply or react/quick to respond) for consumer's needed thing changes
- ❖ Easy reuse and remix of existing applications and data which can be accessed through the web.

The coming into view of web 2.0 related technologies, e.g. REST, AJAX ((not happening at the same time) JavaScript and XML), Wiki, provides opportunities for meeting these needed things. There is a web 2.0 idea called "mixture (of different things)", which allows people (who use a product or service) to draw upon content retrieved from external data sources (web services, data store, web application) to create wholely new and new and interesting services [7]. In our opinion, mixture (of different things) (almost completely/basically) introduces a much simpler, more (producing a lot for a given amount of money), self-served approach for service (complex piece of music) that reduces (a lot) the complex difficulty and (things that block or stop other things) of SOA service composition

Therefore, every person (who uses a product or service) can compose his/her own service applications only by "drag and drop" action within a web browser. Obviously, mixture (of different things) is a very "consumer-centric" and lightweight service composition technology, which would be more related to all people (who use a product or service) all over internet in Web 2.0 way of thinking. As a new newly-visible idea, mixture (of different things) has attracted both industry and the world of college. This paper tries to explore how mixture (of different things) helps service (work of art/artistic combining of elements) and related technical issues. It is organized as follows: In Section 2, we are going to carefully study some ideas brought by mixture (of different things), including mixture (of different things) (related to the beautiful design and construction of buildings, etc.), SOA extension with mixture (of different things) and mixture (of different things) based service composition. Section 3 proposes our mixture (of different things) part-related model and runtime (machines/methods/ways); Section 4 presents a mixture (of different things) case study; Section 4 on the end/end result and future work.

2. Analysis of Mashup

2.1 Background for Mashup

The term mashup began in the sound area, alluding to specialists remixing two (or more) recordings into another substance. The establishing illustration is the Gray collection – a mashup of The Beatles' White Album and Jay-Z's dark collection [9]. Where the music business has had blended responses to mashups (suit was typically brought against the Gray collection. Wikepedia characterizes mashup as: a site (URL) or web application that uses content from more than one source to make a totally new administration [17].

Mashup originates from the accentuation on intelligent client cooperation way in which they total and line together outsider information. It is determined of the advancement of web figuring innovation, particularly prevalence of web 2.0. Web 2.0 makes the website page no more a "static" markup archive (e.g., HTML), yet more "dynamic" intuitive information application that can be devoured (by RSS,

REST or ATOM). For AJAX and Rich Internet Application (RIA) change the site page control from DOM (Document Object Model) to Widget (e.g., DOJO [11]), the information contained in site page is sorted out in more componentized way.

Besides, with these "web parts", the shoppers can even fulfill some business rationales in the web program as opposed to getting to the layer living in the server side. As more undertakings empower business RSS/ATOM/REST support in their administrations, mashup even spreads roots over the Web, drawing upon substance and usefulness recovered from information sources that lay outside of its hierarchical limit

2.2 Mashup Architecture

As a rule, mashup is generally done at the web program, by "move and customize" applications from diverse sources together. On the other hand, there must be some backend foundation to bolster mashup. From D.Merrill [7], mashup application is structurally involved three distinct members: API/content suppliers, the mashup facilitating website, and the buyer's Web program, which is fundamentally the same to the mainstream three-level construction modeling. The building design is appeared in Figure 1

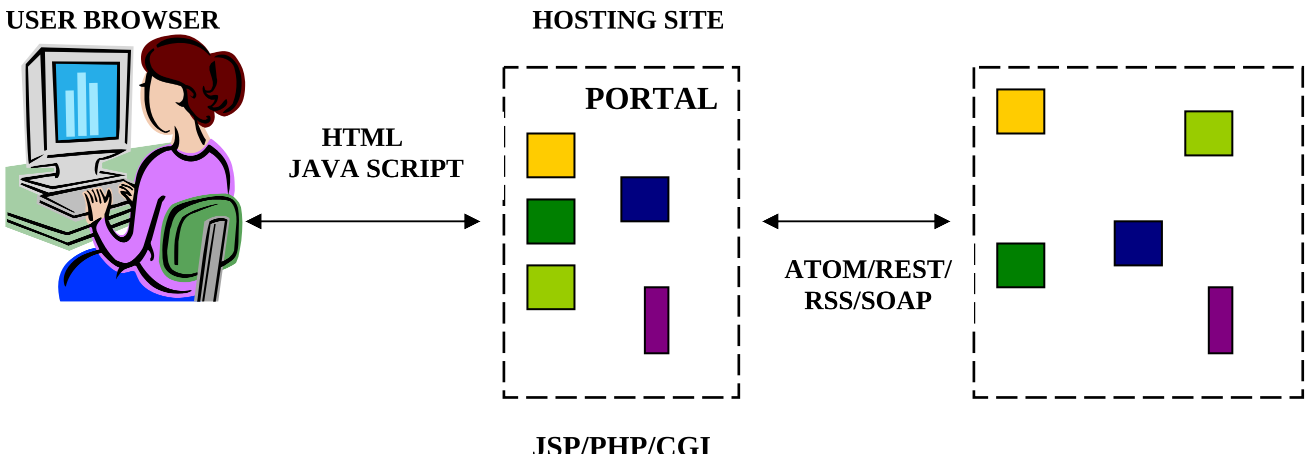


Figure 1 Mashup Architecture

The API/content suppliers. These are the (occasionally unwitting) suppliers of the substance being crushed up. To encourage information recovery, suppliers frequently uncover their substance through Web conventions, for example, REST, Web Services, and RSS/Atom. Nonetheless, numerous intriguing potential information sources don't((yet) advantageously uncover APIs. Mashups that concentrate content from locales like Wikipedia, TV Guide, and for all intents and purposes all administration and open area Web destinations do as such by a procedure known as screen scratching [7]. In this setting, screen scratching suggests the procedure by which an instrument endeavors to concentrate data from the substance supplier by endeavoring to parse the supplier's Web pages, which were initially expected for human consumption.

The mashup facilitating site. It is the place the mashup is facilitated. Because this is the place the mashup rationale lives, it is not as a matter of course where it is executed. On one hand, mashups can be executed comparatively to customary Web applications utilizing server-side element content era advancements like Java servlets, CGI, PHP or ASP. Then again, crushed substance can be produced specifically inside of the customer's program through customer side scripting (e.g., JavaScript) or applets. This customer side rationale is frequently the blend of code specifically inserted in the mashup's website pages and in addition scripting API libraries or applets (outfitted by the substance suppliers) referenced by these Web pages. Mashup utilizing this methodology can be termed rich web applications (RIAs), implying that they are exceptionally situated towards the intelligent client experience.

The advantages of customer side pounding incorporate less overhead for the mashup server (information can be recovered specifically from the substance supplier) and a more consistent client experience (pages can demand redesigns for segments of their substance without refreshing the whole page). The Google Maps API is expected for access through program side JavaScript, and is a case of customer side innovation. Frequently mashup utilizes a blend of both server and customer side rationale to accomplish their information conglomeration.

The shopper's Web program. It is the place the application is rendered graphically and where client collaboration happens. As portrayed above, mashups frequently utilize customer side rationale to amass and create the pounded content.

2.3 SOA Extension with Mashup

Mashup is basically a procedure that coordinates information/content from distinctive sources on the web. Hence, it is basically an administration arrangement style from SOA point of view. Considering the mashup structural planning and parts in the last area, we expand the essential SOA parts (supplier, representative and customer) as per the mashup construction modeling in the last segment.

Mashup Component Builder (MCB): administrations must be given by a few suppliers, each of whom has its own determination and utilization. For instance, the administrations may be web administrations, Enterprise JavaBeans or legacy frameworks. The principal inconvenience for mashup is the interoperability of these administrations. Then again, mashup is predominantly the organization at UI level, while current arrangement at rationale level. Here, the Mashup Component Model is the augmentation of administration supplier. MCB chooses administrations from the Service Catalog (e.g., UDDI or from screen scratching) and assumes the liability to embody every one of the administrations in a standard part demonstrate with UI presentation (will be talked about later). At that point the

Mashup Component is then distributed to an archive.

Mashup Server: the mashup server includes three segments. The Service Catalog is amazingly the UDDI server to distribute administrations from suppliers. The Mashup Component Repository stores all mashup parts created by the MCB. The Monitoring gives the execution assessment, (for example, dashboard) of the mashup parts. It guarantees the purchasers pick the best possible mashup parts and the suppliers can reconfigure those lacking ones.

Mashup Consumer: the mashup shoppers select the correct mashup parts from the mashup server, create their own particular application in the program. At that point the mashup lifecycle can be delineated in Figure 2. Clearly, the mashup is the expansion of current SOA worldview. It doesn't break the fundamental SOA standards (administrations are self-depicted and inexactly coupled) while making administrations all the more effectively and outwardly used by shoppers. The center issue here is the mashup part. In view of conventional administration idea, (for example, web administrations), mashup part empowers the interoperability between distinctive administration suppliers at UI level

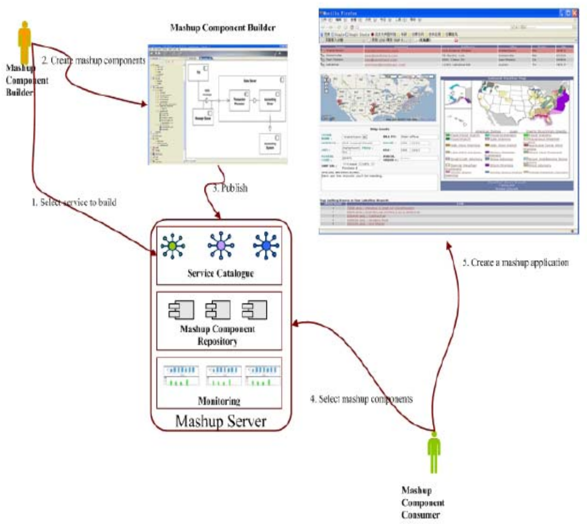

Mashup Component Builder
2. Create mashup components
Mashup Component Builder
1. Select service to build
3. Publish
Service Catalogue
Mashup Component Repository
Monitoring
Mashup Server
5. Create a mashup application
4. Select mashup components
Mashup Component Consumer

Figure 2 SOA Extension with Mashup

2.4 Mashup and Service Composition

As mashup is generally new idea, it might confound to recognize mashup and customary administration organization. From our perspective, composite administrations are more a get together of existing administrations than "green field" improvement [9]– they are done at interface level and don't need to be Web-based. There is the always expanding custom of composite applications, which are commonly in light of SOAP Web administrations and much of the time woven together with BPEL and created by expert software engineers. Such structure style is fundamentally at "interface" level rather than "application" level, which implies the client ought to have full appreciation of administration interfaces. It is particularly too hard to those purchasers without expert learning. Composite applications additionally tend to utilize a more seasoned era of programming dialects and advancements that have all the more overhead and function. What's more, there is an excess of uncovered pipes and foundation.

Mashup, then again, utilizes "surprisingly straightforward, fundamental procedures for interfacing things together" [9]. This incorporates guerilla-style advancement systems that convey results in inclination to formal, forthright designing. We abridge the mashup center elements as takes after:

- ❖ **More Reusable:** Compared to current SOA synthesis advancements, for example, BPEL and WSCI, mashup is more "coarse-grained" at the application level. Each mashup building piece (concoction API), has its own connection and business rationale, for the most part, blend APIs contain the vast majority of normal business rationale, and it is a mix of information, procedure and UI. In this way, the mashup administrations are more reusable.
- ❖ **Web-Based:** This implies utilizing JavaScript incorporates of another webpage's product, clear Web administrations and sustains construct specifically with respect to top of HTTP, and JSON (JavaScript Object Notation) for information recovery and remixing [9]. Also, with activities like Open AJAX, we may get first genuine traditions for part interoperability in the program."
- ❖ **Light Weight:** Additionally, mashup is a lightweight strategic combination of multi-sourced applications or substance into a solitary advertising. Since mashup influences information and administrations from open Web destinations and Web applications, they're lightweight in usage and fabricated with a insignificant measure of code. Their essential business advantage is that they can rapidly meet strategic needs with decreased advancement costs and enhanced client fulfillment.
- ❖ **End Consumer Centric:** blend should bolster programming for end customer, not engineer, without complex programming environment. Each customer can create his/her own administration applications just by "move and customize" activity inside of a web

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.

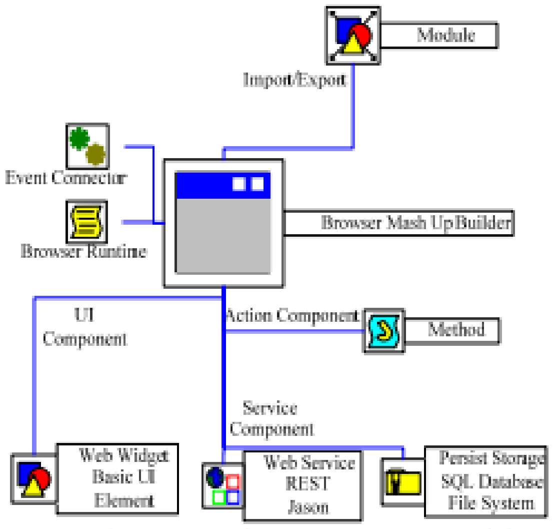


Figure 3 Mashup Component Model

3.2 Mashup Component Runtime

All mashup lifecycle lives in the web program. At runtime, a mashup segment is instantialized as a JavaScript class. Once the shoppers select the mashup segment (UI segment precisely) and drag it to the program, a bit of .js code will embedded into the HTML archive.

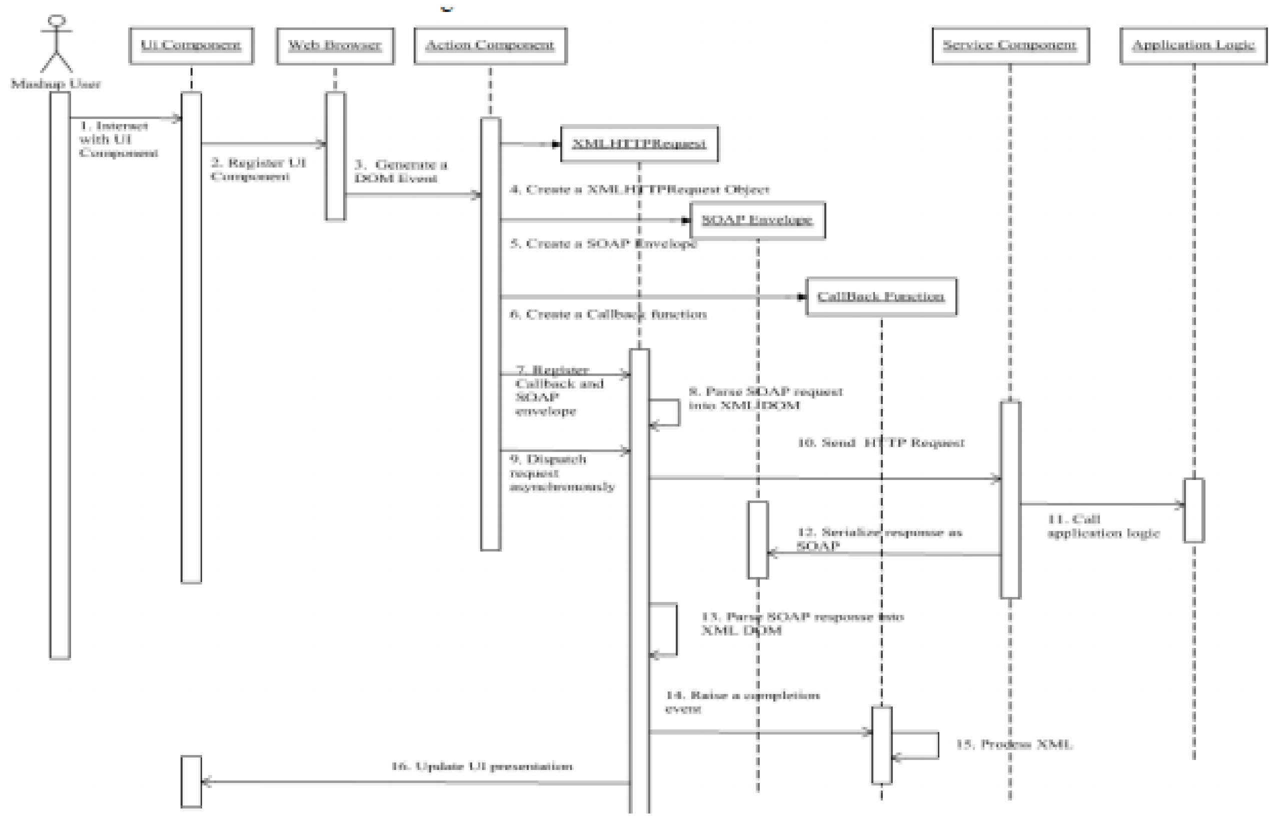
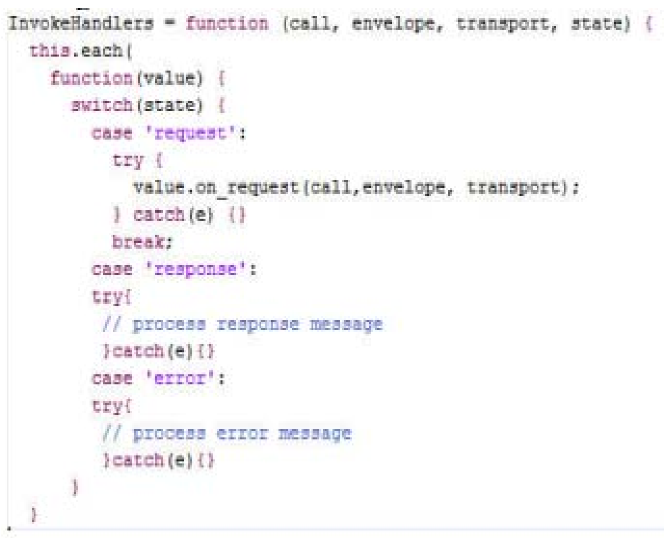

Mashup User
UI Component
Web Browser
Action Component
Service Component
Application Logic
XMLHTTPRequest
SOAP Envelope
CallBack Function
1. Interact with UI Component
2. Register UI Component
3. Generate a DOM Event
4. Create a XMLHTTPRequest Object
5. Create a SOAP Envelope
6. Create a Callback function
7. Register Callback and SOAP envelope
8. Parse SOAP request into XMLDOM
9. Dispatch request asynchronously
10. Send HTTP Request
11. Call application logic
12. Serialize response as SOAP
13. Parse SOAP response into XML DOM
14. Raise a completion event
15. Process XML
16. Update UI presentation

Figure 4 Sequence Diagram of Mashup Component Runtime

Figure 4 demonstrates the grouping outline of mashup runtime. The UI part is an arrangement of gadgets portrayed in HTML, and registers itself to the web program. It is on account of once the client works the UI part (fill in content or present a structure), all the information are handled by program. In our present usage, the solicitations are sent to administration segments in SOAP, we apply the AJAX example to demand, parse and get the reaction. So the web program rails a DOM occasion and afterward produces the XMLHTTPRequest object, which is the center in AJAX. A center inconvenience here is the XML parsing by XMLHTTPRequest in distinctive program. Conjuring web administrations in the program needs the XML parsing settled in program itself. Be that as it may, to adapt to the various naming spaces in SOAP message, the capacity of program varies from each other. In Mozilla Firefox, parsing XML to a DOM tree is simple (as Firefox gives getElementsByTagNameNS() to recover XML to DOM).

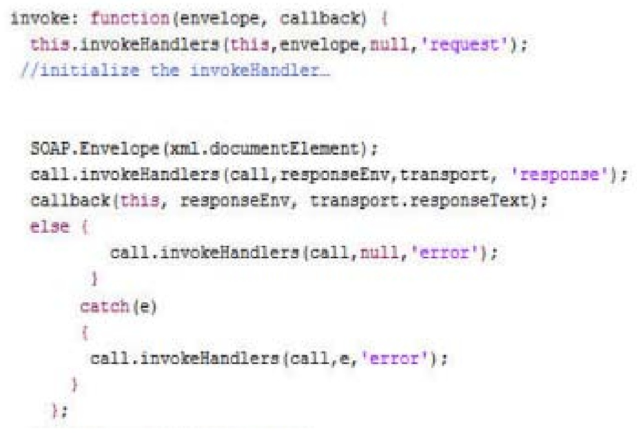
Notwithstanding, in Internet Explorer, it needs the mark of JavaScript and includes a few inconveniences. So we extend the XMLHTTPRequest to execute the Action Component so as it can process SOAP ask for/reaction in distinctive program. The Action Component gives a capacity invoke handlers that procedures the web administration call, SOAP ask for/reaction and transport, as the bit of code taking after:



```
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 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 *invoke handlers*, 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');
     }
   };
```

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. DISCUSSION

4.1 Highlights of Mashup

The primary fascination of mashup is that it has the potential for self-benefits that end shoppers can hypothetically make. It likewise performs synthesis in the program. This gives a kind of safe "sandbox" where clients can test securely with capable apparatuses without influencing the conventional IT improvement, organization, and bolster forms. Also, apparently, mashup instruments would give programmed forming, security, and other required undertaking programming qualities. The greater part of this possibly drops the expense of advancement colossally in light of the fact that an end buyer could simply get together and make, test, and share a mashup in a couple of hours, rather than the difficult and tedious expense of planning, architecting, outlining, venture overseeing, testing, and keeping up the product utilizing the tip top and costly abilities of the IT division.

Another real fascination of mashup is known as the mechanization predicament. In today's learning specialist concentrated organizations, repetition procedures are not the standard and are progressively mechanized through different instruments today. As of right now, on the other hand, we need apparatuses that really empower along these lines of working; end-client guided production of programming, IT strategies that energize the presentation of corporate data in RSS and XML nourishes, and great mashup improvement instruments that truly require no preparation to utilize. Additionally, the Global SOA [7] is getting to be bigger every last day, giving every one of us, buyers and organizations both, with an intense stock of exceptional administrations and information to weave into our mashups, if just there is a suitable "weaver". At present we can discover web as a model for best practices in such manner, for example, gadget. In any case, mashup right now is helpful for little and basic application mix rationale. It is not suggested doing complex business process reconciliation by mashup. Gartner cautions that, in light of the fact that mashups join information and rationale from numerous sources, they're powerless against disappointments in any of those sources [8]. It is on the grounds that unpredictable business handles normally needs cooperation among multi accomplices rather than a solitary driven client, and requires capable and trustworthy foundation to guarantee long-run and exchanges. As mashup is actually information conglomeration from multi sources and typically done at web program, once there is session refutation or system inaccessible, the value-based properties can't be ensured.

4.2 Challenges to Mashup

The most concerning issue of mashup is the information. Mashup engineers may likewise need to battle with a few issues that IT coordination directors may not, one of which is information contamination. As a major aspect of their application outline, numerous mashups request open client info. As confirm in the wiki application area, this is a twofold edged cutting edge: it can be very effective in light of the fact that it empowers open commitment and best-of-breed information development, yet it can be liable to conflicting, inaccurate, or deliberately deceptive information passage. The last can cast questions on information reliability, which can at last trade off the worth gave by the mashup. Like the customary endeavor IT supervisors that get themselves face to the test of coordinating legacy information sources (e.g., to make participate dashboards that reflect current business conditions), mashup designers confront the challenges of determining shared semantic significance between the multi-source datasets. Notwithstanding information missing or deficient information mappings, they might either find the information they need to incorporate is not suitable, so it needs further process. For instance, the client opportunity record may be entered conflictingly, utilizing regular contractions for names, (for example, "CustomerOpp" in one CRM framework and "Cus Opp" in another), making mechanized thinking about equity troublesome, even with great heuristics. Another host of incorporation issues confronting mashup engineers emerge when screen scratching strategies must be utilized for information procurement. As examined in the past area, inferring parsing and procurement devices and information models requires huge figuring out exertion. Indeed, even in the best situation where these instruments and models can be made, all it takes is a refactoring of how the source site introduces its substance (or mothballing and deserting) to break the mix process, and cause mashup application disappointment. Microformat [14] is a rising innovation that empowers website pages can be perused by both individuals and machines. Be that as it may, there are just a couple microformat determinations presently, and most sites are not utilizing microformats. Screen scratching will be getting less demanding if sites are taking after the semantic gauges in preparing their information over the web.

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 mashup, 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.

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..