Mooshabaya - Mashup Generator for XBaya

Buddhika De Alwis[1], Supun Malinga[1], Kathiravelu Pradeeban[1], Denis Weerasiri[1], Vishaka Nanayakkara[1] and Srinath Perera[2]

[1]Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka.

[2]WSO2 Inc., Sri Lanka.

Abstract **- Visual composition of workflows enables end user to visually depict the workflow as a graph of activities in a process. Tools that support visual composition translate those visual models to traditional workflow languages such as BPEL and execute them thus freeing the end user the need of knowing workflow languages. Mashups on the other hand, provide a lightweight mechanism for ordinary user centric service composition and creation, hence considered having an active role in the web 2.0 paradigm. In this paper, we extend a visual workflow composition tool to support mashups, thus providing a comprehensive tooling platform for mashup development backed up by workflow style modelling capabilities, while expanding the reach of the workflow domain into web 2.0 resources with the potential of the mashups. Furthermore, our work opens up a new possibility of converging the mashup domain and workflow domain, thus capturing beneficial aspects from each domain.**

Index Terms - Web 2.0, Web Services, mashups, workflows.

I. Introduction

Both workflows and mashups enable users to compose services and data together to create new composite content and services. Among them, workflows, which depict a real world scenario as a sequence of steps, has found major use-cases in industry as business workflows as well as in research community as scientific workflows used in e-Science. Mashups essentially empower a user-oriented web, by providing a light-weight medium for the users to contribute to the web with their own content, even without any knowledge on complex programming skills [1].

The idea of a mashup application brings the user interactivity and the level of customization of the web information to a new level. The benefits are usually two fold. From the perspective of the web application developer, it presents him with more scope for aggregating content and functionality across various information sources. Ability to use dedicated functionalities from different providers without starting from scratch reduces the development effort and enables richer user experience. From the perspective of the user, well thought out mashing up of data and functionalities lead to coherent presentation of related information in one place. Some mashup efforts allow the user to interactively customize his web experience. Instead of developer driven content aggregation, the user is provided with an intuitive tooling interface where he can specify what data to aggregate and from where. Yahoo pipes [2], [3] is a popular service providing such facilities which basically enables the user to aggregate web feeds according to the particular individual tastes.

Mashups come in several variants according to their usage and purpose. They can be broadly categorized as server side mashups and client side mashups according to mashup runtime [4]. Client side mashups are basically designed to run inside the user's web browser. On the other hand, a mashup can also be deployed in applications such as a web application server. Server side mashups have more freedom and capability than a browser based mashup application. This is due to the lesser security restrictions and absence of cross domain communication issues which plague browser based implementations.

Another categorization can be made by looking at the dynamics of the mashup, especially on what is being aggregated in a particular mashup. It can be aggregating both data and presentation elements into one place or it can be implementing a service composition. If a mashup falls in to the first category it is called a data mashup. Second category mashups are called service mashups. In this paper we are mostly interested with server side service mashups.

The sequences of real world operations are generally depicted as a workflow which is an abstraction of the real scenario, to facilitate further assessment or processing of the given scenario. Workflows also enable capturing and developing human to machine interaction. Then these abstract workflow models can be converted into executable formats according to the requirements.

Traditionally executable workflows have been used in scientific and business domains to model and carry out repeatable processes, where in recent days the workflow is more researched for the use cases for the user oriented web, by analysing the similarities between the workflows and mashups.

This work is motivated by two goals, first we wanted to extend a visual composition tool initially done for workflows to be able to generate mashups as its execution medium, thus enabling users to visually compose mashups, and second, we wanted to explore the possibilities of bridging the workflow and mashup domains so that they can benefit from each other.

To understand the motivation for above two goals, let us consider the following use case. The use case is based on a large scale E-Science project called LEAD (Linked Environments for Atmospheric Discovery) [5], [6], and it has been the target use-case for Mooshabaya project.

Let us consider a meteorologist, who has access to weather data using the satellite feed, ground level equipment, and sensors and predicts the weather by

analysing the data. He processes these data by running them through many pre-processing, forecasting, and post processing steps, which can very easily be described as workflows. As the different conditions on the atmosphere rise, his requirements change, and often, he creates new workflows to analyse those conditions. Furthermore, he wants to execute the workflows and then monitor their progress.

Ideally he needs a much light weight model to create workflows for the scenario rapidly as the environment conditions change rapidly. He also likes to avoid the need to learn the XML technologies or the workflow languages such as BPEL, which takes a lot of time. In this process his interest in using the web based APIs in the solution space should also be noted. He also prefers to integrate real time data collected via web feeds and feeds from the other sources like National Weather Service RSS feeds into his system of workflows.

XBaya workflow composer, which is part of the LEAD project, as well as the other composers like Triana [7], CAT (Composition Analysis Tool) [8], [9], Taverna [10], [11], Kepler [12], and Pegasus [13], enables users to compose workflows and run them. In this work, we propose an extension to the same model, where users could compose workflows using one of the above visual composers, but provide them with an option to expose as Mashups and to run them in a mashup environment. Mashups can provide a lightweight new environment to run workflows while bringing in advantages of Web 2.0 [14], like integration with RSS feeds and the ability to run in a browser.

Having these as the background, we propose Mooshabaya as a system that deploys the workflows as Mashups, instead of the traditional workflow languages. Here mashups are given a new face as a workflow language. This potentially merges the mashup domain into the workflow domain, while revealing the synergy of the domains.

This research paper is organized as follows. Section II describes the motivation of the project. Section III discusses the architecture of Mooshabaya. Section IV analyses the results, while section V talks about the related work. Section VI discusses the idea and the project, while section VII concludes the paper along with the possible future works.

II. Mashups and Workflows, Why?

Currently mashups and workflows mostly cater for divergent set of interests. Mashups are mainly used as a data aggregation technology and whereas workflows are used for process automation. Here we discuss about the conventional workflow development and the mashups regarding their development style and the execution medium.

A. Visual Composition of Workflows

Current mashup offerings cater for end-user centric application development. Based on Web 2.0 technologies the basic premise of mashup applications is enabling user to customize his web experience in an intuitive and easy way [15]. Typically little knowledge about the underlying technology plumbing is assumed from the users. User can model a data mashup on a UI canvas by dragging and dropping related data source nodes and then connect inputs and outputs of those nodes to come up with a data aggregation and a processing pipeline such as yahoo pipes. Presto and Serena are similar Mashup Composers which allow visual mashup composition through their graphical user interface [16], [17].

Once the mashup is visually composed, the Tooling framework generates JavaScript to describe the workflows and associated JavaScript plumbings which is then executed in user's browser or a mashup server. The learning curve involved in this process is minimal compared to the other enterprise level workflow languages, so the users can get started with creating their own mashups instantly without any form of coding associated. As a consequence non tech savvy users can easily come up with little applications that enhance their web experience.

If we consider workflows on the other side of the spectrum what we see most of the time is the direct opposite to the above development style. Development of a workflow of substantial value is a time consuming process even if the associated data processing nodes are present. This is in part due to inherent complexity associated with production grade workflows. To manage this complexity, a standard process involving modelling, prototyping, testing, implementation, deployment and monitoring is required. We cannot say this style of development can be abandoned in favour of do it yourself mashup style development because the inherent complexity of modelled processes makes it practically infeasible in most cases.

But there is an aspect that workflow modelling domain can benefit from mashup development. That is the usage of an approach, where user can visually model a workflow using abstract service definitions and data sources, and offload the responsibility of subsequently generating required concrete executable artifacts and plumbings. Note here the difference from the conventional workflow modelling lies in the fact that when generating concrete executable artifacts user need not to know any specifics of the underlying executable language specifics.

For an instance consider the generation of an executable business process using BPEL [18]. The ideal scenario would be after modelling the workflow user being able to export to any type of an executable format without any workflow language specific settings. The modelling phase should only include the abstractions provided by the service WSDLs, and tooling framework should deal with the specifics of the language (both conceptual and source code level) when generating executable artifacts, so that user is offloaded of learning the specifics and syntaxes of the particular language. This lowers the learning curve associated with workflows, and subsequently drives lower times to deployment. This language agnostic nature allows for greater flexibility in terms of deployment options as well. Visual workflow composers such as XBaya, lead the language specific developer oriented workflow tools

such as Eclipse BPEL designer [19], by providing easy options of workflow modelling.

Especially for business process related use cases this promises a significant potential. In addition to time savings aforementioned, this also enables greater collaboration during workflow modelling phase. Since the modelling phase is largely neutral of any language specifics, the domain experts can actively participate in coming up with the abstract model of the workflow without being forced to learn a new language every time when the workflow language preference such as BPEL [19] or YAWL [20] changes. Even for scientific workflows this benefit can be significant.

B. *Workflow Execution*

At present most of the workflow languages are XML based [21]. Workflow engines does the work of parsing, validating the XML syntaxes prior to deployment. Messaging is also basically done using XML based formats. With the current trend for web service based workflows, this is not surprising. With the backing up of matured WS-* stack [22], web service based workflows can utilize the security, quality of service and other features provided by modern web service stacks. This is vital in making workflows more robust and resilient to real world conditions and requirements.

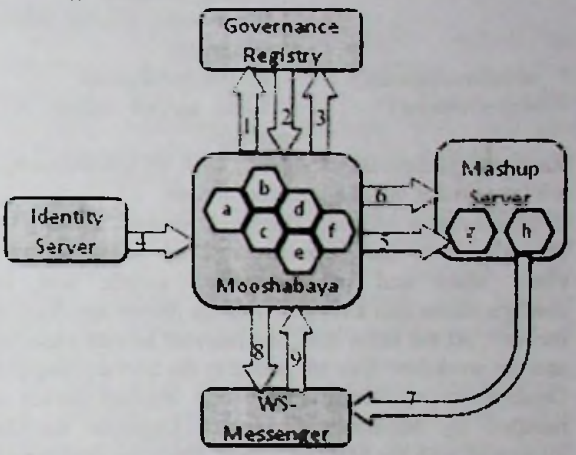
On the other hand mashups are mainly based on JavaScripts which is the de-facto language behind the bulk of Web2.0 technologies. Even though JavaScript can play with XML messages, more efficient and less bandwidth consuming message formats such as JSON [23] has been introduced. What would this mean to workflow domain?

Using JavaScript based mashups as an execution medium for workflow compositions can be handy in this regard. These mashup based workflow compositions can easily consume JSON based RESTful [24] web services which are becoming popular in web 2.0 usage scenarios along side with traditional SOAP [25] based web services.

Other significant prospect that JavaScript based mashups bring to the workflow domain is the redefinition of traditional concept of a service within a workflow. There are several challenges when it comes to data acquisition, visualization, processing and management in workflows. In a conventional workflow, data sources are mainly RPC [26] or web service based. But in the domain of mashups the data sources available can be expanded to web feeds, web scraping, social network APIs and many more as can be amply seen by the burst of web 2.0 applications using mashups. These new sources along with existing internal/external data and web services can be leveraged in creating and injecting more effective business intelligence information to workflow decision points leading to more accurate and reactive workflows. Workflow visualisation depends on its underlying language like BPEL. Based on the features of the workflow implementation language, the visualisation differs. Workflow visualisation is an important component in workflow systems. It depicts the information and data required to figure out the connections and dependencies between workflow tasks and decision making.

III. Mooshabaya Architecture

As seen above, the mashup domain exposes the advantages of web 2.0, hence 'Mooshabaya', as a mashup generator for XBaya Graphical Workflow Composer, is expected to invest on the synergy of the merge of the domains.



Operations involving

The Registry: 1 – Discover; 2 – Fetch; 3 – Add

Identity Server: 4 – Fetch

Mashup Server: 5 – Deploy; 6 – Execute

WS-Messenger: 7 – Publish; 8 – Subscribe; 9 - Notify

Components:

a – Security Component; b – Registry Integration; c – User Interface;

d – Mashup Generation; e – Monitoring; f – Mashup Deployment

g – Service Deployer; h – Eventing Host Object

Fig. 1: Mooshabaya Architecture Diagram

The overall system consists of a Registry, Mashup Server, Identity Server, WS-Messenger and the Mooshabaya workflow composer. Fig. 1 shows the components of Mooshabaya depicting the major operations involving the components.

Corresponding web service definitions are required to compose workflows with necessary web service invocations. Registry Integration Module of Mooshabaya enables the user to fetch such service metadata stored in a remote Web Services Registry instance via the user interface of Mooshabaya, by handling the integration of the Registry into the system. User can compose workflows by dragging and dropping the related metadata files and the other service components into the Mooshabaya drawing canvas and configure the service components dropped into the canvas. The created workflow models can be saved locally as well as remotely on the Registry to be used later.

For executing the workflows, Mooshabaya Mashup Generator Module converts the composed workflows into Mashup scripts and then deploys them in a Mashup Server. In this mashup generation process, first it parses the retrieved object model and sequentially generates the

code segments required to compose the mashup. The methods of the module generate the relevant segments in the mashup scripts as listed below.

writeHeader() - Service details and global variables
writeParameters() - Service stubs
writeSetup() - Input and Output binding types, and base service body.
writeHostObjectScriptBody() - Feeds, scrapers, files and other input sources related operations.
writeInvocations() - Service Invocations
writeOutputs() - Final service output of a mashup

Required configurations such as URL of WS-Messenger are provided at mashup generation time.

Mooshabaya Mashup Deployer Module handles the deployment of the generated mashup scripts, support client stubs and other required scripts, and the configurations into a remote Mashup Server specified by the user. At the same time, the relevant service metadata and the workflow files are added to the Service Registry. Deployment of mashups into the mashup server is handled by Mooshabaya Service Deployer module, integrated with the Mashup Server. Once the deployment is done, the workflow can be executed and monitored.

Generated mashups use JavaScript stubs to invoke web services, and other input sources such as feeds and scraper inputs are invoked using JavaScript Host Objects. Here host object refers to a type of JavaScript object that is created in server side. JavaScript host objects are defined in the runtime environment that the JavaScript is executed, such as a web browser or a specific JavaScript engine, rather than the JavaScript language.

During the code generation, mashups are instrumented with directives which would generate events that depict the workflow execution, and publish them to a message broker called the WS-Messenger [27] at run time. Using the publish - subscribe system, Mooshabaya Monitoring Module monitors the execution by subscribing to the generated events.

We have used the known existing tools as the base in developing the system. XBaya Graphical Workflow Composer from Indiana University is used as the core of the Mooshabaya workflow system. We have extended XBaya to export the workflows as mashups, and the mashups are deployed into WSO2 Mashup Server [28]. WSO2 Governance Registry is used as the registry of the system.

Currently mashups generated by Mooshabaya supports invoking web services secured with basic WS-Security scenario UsernameToken over HTTPS. Associated policy definitions are required to be in service WSDL in order for Mooshabaya to correctly recognize security requirements.

IV. Results

"Mooshabaya", the proposed solution was implemented and tested for workflow creation and mashup generation for different number of web service nodes. A system with 2 GB memory, 320 GB hard disk, 2.20 GHz Intel Core Duo Processor, and Linux - Ubuntu 10.04 Lucid Lynx Operating System was used as the testing environment. The web services used for testing were ensured to have the nodes with similar functionality to avoid biased results. For this we have used dummy workflows created using the chosen numbers of multiplier web service nodes.

The tests were targeted towards measuring the effectiveness and efficiency in using Mooshabaya in the production environment as the workflow composer and the mashup generator for the real world workflows.

The time taken to generate the workflow file, BPEL file, and the mashup file were measured against the number of nodes. Similarly the size of the workflow file, BPEL file and mashup files too were measured against the number of workflow nodes. The generated file size and the file generation time were plotted against the number of nodes.

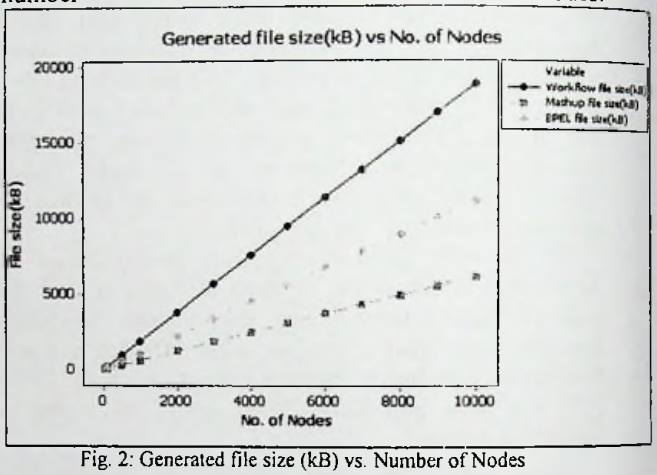


Fig. 2: Generated file size (kB) vs. Number of Nodes

As illustrated in Fig. 2, Generated File sizes showed a linear proportional relationship with the number of nodes in the workflow. Hence, for a given number of nodes, generated mashup file will have the minimum file size. This a significant advantage where the generated mashup or the BPEL script should be uploaded to a remote engine before execution. So the mashup deployment time is significantly less than BPEL deployment time, as the size of the generated mashup is pretty low.

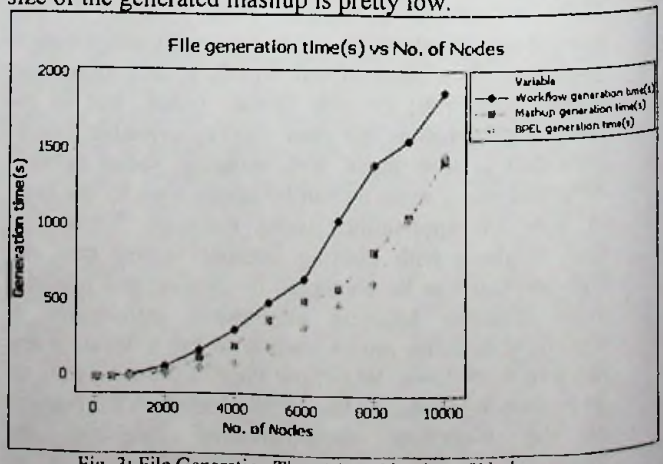


Fig. 3: File Generation Time (s) vs. Number of Nodes

The time taken to generate the mashup scripts, BPEL script, and the workflow files, showed an exponential

increase against the number of nodes in the workflows, as depicted in Fig. 3. In this case, we can see the BPEL script generation time has a higher gradient than mashup script generation time. So after 10000 nodes, we can see the script generation time of a BPEL script is larger than the respective mashup script.

These tests show that the mashup and workflow generation time are quite reasonable to use Mooshabaya in real world workflow scenarios with considerable number of workflow nodes.

V. Related work

As discussed before, advantage of converging the mashup domain with the workflow domain are enormous. Many researches have been done on both the domains to explore new possibilities and use cases. Though the design and implementation of Mooshabaya as a mashup generator for XBaya workflow composer which merge the workflow and mashup domains is a fresh idea, there have been many researches and related work going parallel to this project and there are some researches that do have some similar aspects to our research.

A process of developing and deploying a workflow management system for heritage data mashups is discussed in [29], where they have implemented the concept for a limited test environment for a particular workflow model within their heritage system. The implementation for the general workflows and a model that can be run with the final mashup system were proposed, yet not implemented.

The approach presented by Tobias Nestler et al. [30] follows the idea of integration at the presentation layer enhanced by user interface (UI) related service annotations, while [31] concentrates on the generation and composition of service front-ends. Thus both are proposing service composition at the presentation layer, targeting user-centric composition of the services.

Yahoo Pipes is an online service that allows remixing feed types and creates mashups, with a Pipes Editor which allows creating Pipes visually. This feature is resembled in Mooshabaya, which allows feeds, as well as scrapers and files to be added as nodes in the workflow editing area or the canvas of XBaya.

VI. Discussion

As the motive of web 2.0 paradigm, web services as well as the web is becoming customer oriented, and light weight. Traditional approach of SOA expects the developers master technologies such as BPEL and Jython Scripts [32], hence service composition based on web 2.0 mashups which are easy to use is proposed [33] as an alternative towards bringing the web into the centre of the development.

Researches with the goal of merging the mashup domain with the workflow domain are carried out. An analysis and a comparison of Grid workflows and web 2.0 mashups has been carried out in [34], which talks about the potential and similarities of the topics, giving a hint about the possible scenario that uses both the two. Converging workflow domain and the mashup domain also extends the reach of data sources of conventional workflows by allowing them to accumulate data from data sources like RSS and Atom web feeds and web scraping which is made possible by using JavaScript mashups.

While expanding the reach of the workflow domain, the unique workflow approach simplifies the task of workflow/mashup composition by the comprehensive tooling platform. This workflow approach provides the potential to create mashup in an effective and interactive way, hence providing an option of creating mashups in end to end manner.

Mashup becomes a light weight medium for workflows, instead of the traditional workflow approaches which happened to be heavy weight. Deploying workflows as mashups also facilitates developing workflows rapidly based on mashups which can be useful in workflow prototyping and testing. Mashup solutions typically handle the actions Compose, Generate, and Deploy in the mashup life cycle, while Mooshabaya by using the workflow approach to the mashups, handles Configure, Execute, and Monitor actions too, which is yet another major advantage of Mooshabaya.

Mashups can be of enterprise mashups, data mashups, and consumer mashups. Mashup tools are often categorized based on the categories they serve. While IBM Mashup, Serena, and Jackbe focus on all these three types, Yahoo Pipes focuses more on the data and consumer mashups [17]. Some mashup tools even narrow their focus down to the consumer mashups.

VII. Conclusion and future work

The research on the possible merge of the mashup and the workflow domains focused the potential advantages and the enhancements for both the domains, revealing further use cases for both. It should however be noted that the mashup generation which has been implemented on XBaya Workflow Composer from the OGCE Workflow Suite [35], can be extended in two ways, either from the mashup perspectives or from the workflow perspectives, which could be done using the current implementation of Mooshabaya as a base.

At the same time, this wouldn't mean converting high value business processes to mashups would be the ultimate solution. It will not be the case since JavaScript based mashup APIs are currently not powerful enough to capture all the non functional requirements that a typical SOAP web service based implementations can offer. Proper JavaScript implementations are required to be present to cope with these aspects. Already there are certain implementations which provide some of these aspects. WSRequest JavaScript implementation that can invoke some secure services is such an implementation.

Rather, what is more prudent is to use mashup based implementations in data intensive workflows where data aggregation also plays an important role and where service level requirements such as security and QoS are not that important. Using this approach the aggregated data can be seamlessly pipelined in to the workflow data processing nodes.

Though we have chosen XBaya as our workflow composer, the other workflow composers which have

their own specializations, also may follow the concept of using mashups to deploy the workflows. Monitoring the mashup execution as workflow monitoring and providing the end-to-end handling of the mashup life cycle are the specific achievements of Mooshabaya, as a mashup generator for a graphical workflow composer.

REFERENCES

[1] Yu, J., Benatallah, B., Casati, F., Daniel, F. (2008). "Understanding Mashup Development," IEEE Internet Computing, 12(5), 44-52.

[2] J. Fagan. Mashing up Multiple Web Feeds Using Yahoo! Pipes. Computers in Libraries, 27(10):8, 2007.

[3] Mark Pruett, "Yahoo! Pipes," First edition.

[4] Frederik De Keukelaere , Sumeer Bhola , Michael Steiner , Suresh Chari , Sachiko Yoshihama, SMash: secure component model for cross-domain mashups on unmodified browsers, Proceeding of the 17th international conference on World Wide Web, April 21-25, 2008. Beijing, China.

[5] Droegemeier, K. K., et al. Linked Environments for Atmospheric Discovery (LEAD): A CyberInfrastructure for Mesoscale Meteorology Research and Education. in 20th Conference on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology. 2004. Seattle, WA.

[6] Kelvin K. Droegemeier, V. Chandrasekar, Richard Clark, Dennis Gannon, et al, Linked Environments for Atmospheric Discovery (LEAD): Architecture, Technology, Technology Roadmap and Deployment Strategy. [Online]. Available: http://ams.confex.com/ams/pdfpapers/86256.pdf

[7] Shalil Majithia, Matthew Shields, Ian Taylor, and Ian Wang. "Triana: A Graphical Web Service Composition and Execution Toolkit." In IEEE International Conference on Web Services (ICWS'04), 2004.

[8] J. Kim, Y. Gil, and M. Spraragen, "A Knowledge-Based Approach to Interactive Workflow Composition." To appear in Workshop on Planning and Scheduling for Grid and Web Services." at International Conference on Automated Planning and Scheduling (ICAPS-2004), 2004.

[9] Jihie Kim, Marc Spraragen and Yolanda Gil, "An Intelligent Assistant for Interactive Workflow Composition," University of Southern California/Information Sciences Institute, Marina del Rey, CA 90292 USA.

[10] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn, "Taverna: a tool for building and running workflows of services.," Nucleic Acids Research, vol. 34, iss. Web Server issue, pp. 729-732, 2006.

[11] Srinath Perera, Dennis Gannon, "Enabling Web Service Extensions for Scientific Workflows." Computer Science Department, Indiana University, Bloomington IN 47405.

[12] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao, Scientific Workflow Management and the Kepler System, September 2004; revised March 2005. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.59.6066&rep=rep1&type=pdf

[13] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny, "Pegasus: Mapping Scientific Workflows onto the Grid," Lecture Notes in Computer Science, 3165:11 – 20, Jan 2004.

[14] O'Reilly, Tim. What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software, March 2007. [Online]. Available: http://mpra.ub.uni-muenchen.de/4578/1/MPRA_paper_4578.pdf

[15] John Crupi and Chris Warner, Enterprise Mashups Part 1: Bringing SOA to the People, Published: May 16, 2008 (SOA Magazine Issue XVIII: May 2008). [Online]. Available: http://www.soamag.com/I18/0508-1.php

[16] A. Koschmider, V. Torres, and V. Pelechano. Elucidating the mashup hype: Definition, challenges, methodical guide and tools for mashups. In Proceedings of the 2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web at WWW2009, Madrid,Spain, April 2009.

[17] Anjomshoaa A, Bader G, and A Min Tjoa. (2009) Exploiting Mashup Architecture in Business Use Cases. In Procs of the 2009 International Conference on Network-Based Information Systems (NbiS 2009). UIPUI, Indianapolis, USA, August 19-21, 2009.

[18] Tony Andrews et al., Business Process Execution Language for Web Services, Version 1.1. [Online]. Available: http://download boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf

[19] X. Fu, T. Bultan, and J. Su. "Analysis of Interacting BPEL Web Services," in *Proceedings of WWW'04*, pages 621–630. USA, 2004. ACM Press.

[20] W. M. P. van der Aalst and A. H. M. ter Hofstede, "YAWL: yet another workflow language."

[21] Workflow Management Coalition, XML-Based Workflow and Process Management Standards: XPDL, Wf-XML [Online]. Available: http://xml.coverpages.org/wf-xml.html

[22] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, and Donald F. Ferguson, "Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More."

[23] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta, Comparison of JSON and XML Data Interchange Formats: A Case Study, Department of Computer Science, Montana State University – Bozeman, Montana, 59715, USA [Online]. Available: http://www.cs.montana. edu/izurieta/pubs/caine2009.pdf

[24] Michael zur Muehlen, Jeffrey V. Nickerson, and Keith D. Swenson, "Developing Web Services Choreography Standards – The Case of REST vs. SOAP."

[25] Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana, Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI, Francisco Curbera, IBM T.J. Watson Research Center. [Online]. Available: http://www.site.uottawa.ca/~ttran/teaching/csi5389/papers/Unraveling%20the%20Web%20Services%20Web.pdf

[26] Andrew D. Birrell , Bruce Jay Nelson, Implementing remote procedure calls, ACM Transactions on Computer Systems (TOCS), v.2 n.1, p.39-59, February 1984. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.81.2486&rep=rep1&type=pdf

[27] Yi Huang, Aleksander Slominski, Chathura Herath, Dennis Gannon, "WS-Messenger: A Web Services-Based Messaging System for Service-Oriented Grid Computing," ccgrid, pp.166-173, Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06), 2006.

[28] Jonathan Marsh. Mashup: Noun or Verb? [Online]. Available: https://www.wso2.org/repos/wso2/people/jonathan/Mashup%20Noun%20or%20Verb.pdf

[29] A. Al-Barakati, W. Zhang, M. Z. Patoli, M. Gkion, N. Beloff, P. Newbury and M. White, "An Integrated Workflow Management Solution for Heritage Information Mashups," Department of Informatics, University Sussex, Brighton, United Kingdom, BN1 9QJ.

[30] Tobias Nestler, Marius Feldmann, Andre Preußner, and Alexander Schill, Service Composition at the Presentation Layer using Web Service Annotations. [Online]. Available: http://mashart.org/composableweb2009/paper7.pdf

[31] Tobias Nestler, Lars Dannecker, and Andreas Pursche, User-centric Composition of Service Front-ends at the Presentation Layer, SAP Research Center Dresden, Germany. [Online]. Available: http://ceur-ws.org/Vol-540/ugs2009_submission_2.pdf

[32] Samuele Pedroni and Noel Rappin, "Jython Essentials," March 2002.

Liu, X., Hui, Y., Sun, W., Liang, H. "Towards service composition based on mashup," in *Proceedings of the IEEE.*