2009 Third IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2009)

Modeling the Dynamics of Web-based Service and Resource-Oriented Digital Ecosystems

Tharam S. Dillon, Chen Wu
DEBI Institute, Curtin University of Technology, Perth, Australia
E-mail: tharam.dillon@cbs.curtin.edu.au

*Abstract - T***he notion of digital species is broadened to include services and resources, special issues arise in modeling the dynamics and workflows with representations associated with these services and resources. To address these issues, this paper explores two different yet related approaches: the traditional BPEL-based workflow modeling approach and the Mashup-based Web approach. In this paper, we first demonstrate two examples of service-oriented and resource-oriented digital ecosystems on the Web. We then identify key issues pertinent to both types of DES. We discuss formal definition, specifications and issues of BPEL-based approach and Mashup-based modeling techniques with computational formalisms. Finally, we propose a hybrid approach to deal with modeling the dynamics in processes associated with such Digital Ecosystems**

I. Introduction - Agent, Services, Resource-Oriented Digital Ecosystems

In the early wave of Digital Ecosystems (DES), it was assumed that agents were the primary digital species and they collaborated in a way to allow them to individually achieve their goals as a global aim. This presented a high degree of autonomy and self-organizations. The notion of DES has evolved since then towards encompassing enterprises of different sizes (big and small) to collaborate and participate in order to, for instance, to form an extended enterprise. A consequence of this is that the notion of digital species is broadened to also include services and resources. Thus a full-fledged DES becomes a Service-Oriented DES (SO-DES), a Resource-Oriented DES (RO-DES), as well as a Agent-Oriented DES (AO-DES), or a hybrid of these three types. The rise of SO- and RO-DES raises special issues in modeling the dynamics and representation associated with this. To address these issues, this paper explores two different yet related approaches: the traditional BPEL-based workflow modeling approach and the emergent Mashup-based Web approach.

An SOA realization - Web services technology particularly - should allow for quality of service models to control and manage the interactions. Web Services can be dynamically composed into applications in real time. The dynamic nature of web services allows the implementations to be platform independent and programming language-neutral.

The remainder of the paper is structured as follows: Section II illustrates examples of current SO-DES and RO-DES on the Web. Section III discusses the abstract structure of a DES. Section IV identifies essential issues need to be addressed in both types of DES. In Section V, we discuss formal definition, specifications and issues of BPEL-based approach. Mashup-based modeling techniques with computational formalisms are presented in Section VI. In Section VII, we propose a hybrid approach to deal with modeling the dynamics and workflows. The paper concludes in Section VIII.

II. Web-based Service/Resource-Oriented DES

In this section, we illustrate two examples of current SO-DES and RO-DES on the Web, the e-commerce and service provider *Amazon* and the social network infrastructure *Facebook.*

A. Amazon.com as an Digital Ecosystem

Amazon.com features a very sophisticated e-commerce platform on which anyone can become a retailer partner through the AWS (Amazon Web Services) e-commerce services. In essence, there are three categories of services made available through AWS: the Amazon platform built-in Web services, retailer partner Web services, and public Amazon Web Services. AWS developers can now build very complex, personalized applications using primitive Web services that are simple by their own nature. This is a major competitive advantage of AWS. In effect, developers have built a large variety of applications using these services, ranging from business applications to smart utility tools. This results in a mixture and mash-up world, which is part of the Internet Ecosystem.

In a resource-oriented view, The Amazon WS Platform does provide an information model that represents the detailed data structure of e-commerce products, user profiles, and user reviews. It also defines certain workflows (e.g. place an order) that facilitates common e-business processes.

In a service-oriented view, the Amazon WS process can be very flexible, which is completely dependent on the DES participants' intention. There are no tied restrictions for participants' internal architecture. AWS also provides different protocols of Web services API (e.g. SOAP-based and REST-based). DES species can choose the preferred protocols to interact with the core platform. Once the protocol is given, the participants can use various technology and software architecture to engage in the information exchange and protocol fulfillment.

B. *Facebook as an Digital Ecosystem*

Facebook is an extremely popular social networking website, by which the users can connect with friends, join groups of common interest, send messages to others, and organize events amongst communities. In early 2007, Facebook started to provide an API to its community developers, thereby inviting Facebook developers to create and deploy new applications that can seamless integrate built-in features provided by Facebook and leverage the massive number of users on the Facebook platform. More uniquely, the API also allows the application hosted on a developer's site to directly output to the original Facebook website. Therefore, it supports two-way integration for both pushing and pulling valuable information to and from the Facebook platform. Such a flexible and powerful API has generated unprecedented opportunities for Facebook 'Extension' developers. Many small and medium enterprises come into the picture, tapping into the massive user database maintained by Facebook. For example: the Faceconnector [9] mashup integrates Facebook profile information with Salesforce data in real time, providing managers with instant insights of customers in order to build a "better customer relationship".

III. Abstract Structure of a DES

We believe a DES consists of three essential components: *Core*, *Extension*, *Value*. Core represents Keystone players that provide the infrastructure. DES infrastructure provides flexible *Extension* mechanisms that allow other players to extend the scope, function of the infrastructure through various innovative and value-added channels (e.g. APIs, Web services, communications, workflows, etc.). Finally, the *Value* is generated from both the infrastructure and the applications built upon the extension mechanisms by many different small and medium players. Based on this scheme, Facebook DES has the structure as shown in Fig. 1 below:

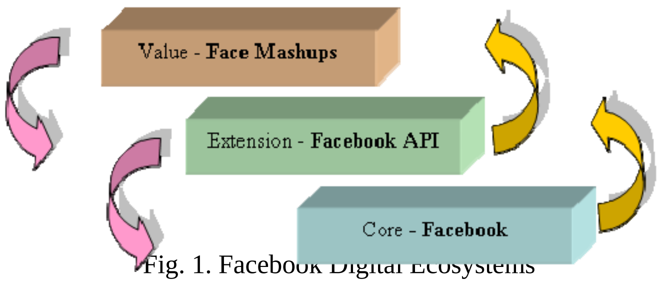

Fig. 1. Facebook Digital Ecosystems

IV. Issues in Web-based SO-DES and RO-DES

We identify four general issues for modeling the dynamics of a SO-DES or RO-DES: service discovery, interaction, composition, and Mashups. Here we use the term 'service' generally to refer to both a service or resource. Note that the distinction between services and resources indeed depends on the contexts, however, there are important differences between these two types of computing models.

A. *Service Discovery*

Service discovery has been a key aspect in the SOA research community. Web services are often deliberately built for reuse. Developers can provide specialized and sophisticated functions by using a readily implemented Web service in order to maintain lower development cost and higher efficiency. To achieve the goal of reuse, Web services firstly need to be discovered. As an essential SOA activity, service discovery paves the way for conducting further important SOA activities such as service interaction and composition in a dynamically changing business environment. Web services discovery thus has received extensive studies in recent years. In our previous study [10], we have discussed various methods of WSDL-centered service discovery based on three levels of abstraction: keyword, semantics, and structure. In [11], we used an Index-based approach augmented with modern Web technology to facilitate service discovery. In [12] we have illustrated that the relationships between service discovery, resources, and semantics. In particular, service discovery sits at the 'Semantic' Layer of the distributed computing stack. Semantic is the shared understanding of terms of the behavior of a service when it is being consumed by other applications or services. For service consumers, implicit semantics often help to understand the meaning and purpose of interacting with a service. Moreover, explicit and formal semantic representation (e.g. ontology) can be processed by machines to facilitate automated service discovery.

B. *Service Interaction*

Service interaction occurs when the service message (requests and response) flows between service provider and requester in order to fulfill the actual service delivery. Two interaction paradigms are common in SOA: SOAP-based and REST-based.

SOAP defines a formal set of conventions. As the underlying messaging solution for contemporary Web services, SOAP basically addresses four major issues for connecting any Web services. First, it defines a standardized XML-based message format. Second, it specifies a process semantics in which a SOAP node can process SOAP messages in a predetermined way. Third, it defines the binding mechanism which enables SOAP message to be delivered via different transport protocols. Last, it defines how binary data can be sent using SOAP messages. It is expected that, with SOAP, one can access any Web service in a flexible and scalable manner.

However, it is interesting to observe that this appears not the case, at least when it comes to accessing the public Web services "on the Web". 85% of the developers prefer the HTTP-based communication model to SOAP-based messaging. This is partly explained by the fact that querying Amazon using REST API was about six times faster than with the SOAP API [1]. Representational State Transfer (REST) is a specific Web architectural style introduced in [2].

The RESTful Web services do not use SOAP as the underlying messaging framework. This directly points to the potential problem of SOAP binding specification, in which the HTTP binding has been formalized. This is the result of thinking of HTTP as merely a transport protocol, rather than a semantic-capable (i.e. 'smarter') application protocol, which in the case of SOAP, is the SOAP protocol per se. This also boils down to the core problem of the debate between the traditional Web services community and RESTful Web services proponents: whether SOAP is needed at all to connect and consume a Web resource (service). Based on the RESTful style, we have proposed to use RESTful style in order to syndicate Web services metadata in order to form the Web services space [12].

C. *Service Composition*

Service composition is a significant way of reusing Web services and is also one of the 'holly grails' that Web services are designed to accomplish. Indeed, Alonso et al.[13] maintain that Web services are born to solve the composition problem. From the business perspective, service composition has been conceived as the "critical missing link" between service providers and service consumers that strive to have the competitive advantage [14]. From the technical viewpoint, Web services are considered as "an optimal candidate platform" for both data and application integration [15]. Following this proposition, application integration is taking over the majority of the world software market [16]. Service composition is expected to play a key role in a very broad range of applications such as e-business, e-government, enterprise application integration, portal website, e-healthcare systems, mobile computing, etc. Therefore, research into service composition is believed to have a considerable impact on both software development and domain-specific applications.

In general, service composition includes two important steps: one is the selection of the required component web services and their composition into a composite web service. The second is the co-ordination of the execution of the different web services in the composite web service so that they are executed in the right sequence and the preconditions for a particular component service are met before execution. If such co-ordination is a centralized one can use an orchestration approach to coordination [13]. If, however, the co-ordination is distributed then one needs to use a choreography approach to co-ordination [17]. Essentially, when assembling these compositions. these paradigms involve design time composition and run time binding. The issues of validation and verification of the coordination of a particular composition of web services becomes a major issue whether one uses the orchestration or choreography model for co-ordination. In addition to the performance of the particular composition, the effect of traffic on this performance and the reliability of the composition and its improvement through fault tolerance techniques become major problems of concern..

D. *Mashups*

In recent years, Mashups have emerged as a Web-based composition approach that allows end users (vs. professional developers) to create their own applications in an efficient manner without dealing with sophisticated techniques and specifications. We believe the development of the Mashup is the direct consequence of Service Web or Hidden Web.

Unlike its predecessors such as Web 1.0 websites in which content exchange is the main purpose of the Web, the current Web has been extended into a live, computational exchange platform. In addition to server side scripts (e.g. CGI, PHP, ASP, etc.) supported by "hidden" databases, a large number of active services have emerged in the form of API such as WSDL or JSON. Service-oriented Architecture (SOA) thus represents an important architectural approach that turns millions of Web resources (content or computation) into reusable services that constitute a "Programmable Web". Today, innovative Web developers are using the Programmable Web to create values in unprecedented ways that many software engineers have never imagined before. We believe that Mashup is one of the key enabling technologies that can realize the true value of SOA on the Web.

V. BPEL-BASED SERVICE COMPOSITION

A. *Formal Definition*

BPEL (Business Process Execution Language) has emerged as the defacto standard for composing Web services using a workflow-based language. Specifically designed for Web services, a BPEL process per se is exposed as a Web service defined using the WSDL interface. Therefore, BPEL composition process is recursive such that a BPEL process can be integrated into another 'higher level' BPEL process as a regular Web service. By defining a formal set of workflow constructs (e.g. sequence, while, scope, etc.), BPEL is aimed at providing a powerful service composition model in order to tackle the complex requirements for business process engineering within and across organizations. It also supports both executable and abstract processes, the former one is used within the organization, the latter one defines the message exchange 'protocol' shared and respected amongst business partners.

B. *Validation Mechanism*

BPEL specification does not include any forms of verification and validation mechanism, which is crucial for complex service composition in enterprise computing. The issues of validation and verification of the coordination of a particular composition of web services becomes a major issue whether one uses the orchestration or choreography model for co-ordination. In addition to the performance of the particular composition, the effect of traffic on this performance and the reliability of the composition and its improvement through fault tolerance techniques become major problems of concern. In our previous work [18], we have used Colored Petri Net (CPN) to address this issue. CPN is the high level Petri net which allows for the representation of the multi-level behavioral abstractions through place/transition refinements. It allows the use of colored Tokens, complex Predicates associated with transitions and the use of guards to control the enabling of transitions before firing. The advantage includes avoiding cluttering the diagrammatic representation with excessive formalism, representing complex predicates and reduced dimensionality. The proposed model deals with the design at the semantic level in three layers, the individual components level, the dynamic formed/combined secure semantic web services level, and the behaviors of the formed/combined secure web services at the entire system level.

C. *Issues*

A recent critical review [3] suggested that BPEL has several critical issues. First of all, BPEL is too difficult to use. It is more like a classical programming language but written in a verbose XML representation that makes it very hard for normal developers to read and to write BPEL. Existing BPEL tools provided by vendors provide GUIs that support drag and drop BPEL constructs, thus only providing the one-to-one mapping from a BPEL construct to a drawing box. Therefore, they fail to approach BPEL from the high level modeling perspective, which is the only knowledge that most business analysts and domain users have. Moreover, BPEL specification does not include any forms of verification and validation mechanism, which is crucial for complex service composition in the enterprise computing to have, even though some researchers have proposed to use techniques such as Petri Net for this purpose. Lastly, so far BPEL has not fully helped to define the abstract process, BPEL's popularity as an executable process raises the question of whether BPEL is needed at all. This is because defining internal workflow within each organization can be carried out in a far more cost-effective and flexible manner without resorting to BPEL. Many user-friendly workflow diagrams, scripting languages, and tools can accomplish this task without too much difficulty. Indeed, we believe, Mashup (see Section 3.3.4), a simple way of conducting service composition on the Web, is perhaps more suitable for such a requirement if augmented with appropriate verification and security models.

Another key practical issue when deploying BPEL on the Web lies in the fact that many RESTful Web service APIs do not use WSDL to specify their interface. Rather, popular Web APIs on the ProgrammableWeb.com use various representation formats such as Microformats (*hCalendar*, *hCard*, etc.), Atom, JSON, RDF, and so on. Since BPEL4WS heavily relies on WSDL, it is not possible to directly apply BPEL into the solution of the Web-based workflow systems.

VI. Mashups

In this section, we aim to explain the basics of the Mashup and how it come into being and its underlying techniques.

A. *Formal Definition*

Mashup refers to a (or part of a) webpage or a Web application that seamlessly combines content from more than one source into an integrated user experience. It represents a new Web development approach that allows users to 'remix' various Web services, each featuring its own capability, to build an application that serves a new purpose.

To our understanding, Mashups represent a radical 'simple' way of developing a distributed software application. The implication of such a user-centered software development approach is very significant. It means that the service consumers can create and try out 'self-service' whenever needed and possible through integrating existing information services across the Web. With HTTP, Atom/RSS, and other scripting techniques (JavaScript, AJAX, JSON, etc.), a seemingly very complex application with a rich user interface can be built with relative ease using 'drag and drop' GUI operations. Moreover, due to its light weight and ease of use, the increasing number of Mashups developed will certainly inspire service providers to supply more and more useful and user-friendly information services that are available on the Web. This, in turn, will motivate end users to innovate more quality Mashups to solve various application problems. Such a good cycle results in a service mashup ecosystem, in which service consumers and service providers will benefit through such 'ubiquitous' mashup connections. At the time of writing, the ProgrammableWeb, the defacto Web 2.0 services registry, has registered 644 APIs and 2749 Mashups. Javascript is the key technology underpinning Mashups.

B. *Formal Specification: Computational Exchange*

Mashup with its underlying technology represents a Web computing formalism shift away from the traditional content exchange to current computational exchange [4]. Essentially, computational exchange includes two important forms: code mobility and continuation.

Code mobility refers to the software capability to dynamically change the bindings between code fragments and the location where they are executed. It involves both change in binding dynamically and relocation of code [5]. An early example of code mobility on the Web is the Java Applet, in which the Java binary code (i.e. Java classes) can be dynamically downloaded from the Web server to a Web browser, which then executes the bytecode using its embedded Java virtual machine. In the context of Mashup, Ajax now enables Web application to dynamically download the code to the browser from its origin server, thereby performing computations locally at the client side. By reducing the computational latency of presentation events, AJAX makes possible a new class of interactive applications with a degree of openness and flexibility that may be impossible in purely server-side computing.

Continuation can be viewed as a transient and abstract representation of the control state of a computation to be resumed right after the point where it was suspended. Essentially, continuation represents “the rest of the computation”. Continuation is specifically useful to deal with web-based page-centric software development as shown in the case of Ajax and Mashup. Rather than considering a web application consisting different pages, the browser-server interaction becomes a single application program (e.g. Ajax scripts) in which continuations deal with pieces of execution. Mashup pushes continuation to the next level by including different servers interactions into this single application program, which forms the Web-based workflow system. It should be noted that a continuation can be suspended/resumed on either server-side or client side, forming server-side and client-side Mashups respectively.

Formally, Maximilien et al. [7] has defined a Mashup with three primary components:

1. *Data mediation*, which is responsible for retrieving and integrating data with heterogeneous structures from multiple sources. This component involves essential tasks related to various data manipulation techniques such as conversion, filtering, matching, combination, transformation, etc.)
2. *Process*, which defines and executes the orchestration or choreography at the application level. The constructed process is developed by integrating data and functions exposed by the services through APIs.
3. *User Interface*, which allows final users to interact with the Mashup through Web browsers. This is one of the key distinction between Mashup and Web services workflow applications, where little human involvement included in modeling the choreography specification.

C. Five types of Mashup

In earlier discussion on continuation, we distinguish between a client-side Mashup and a server-side data Mashup. The author in [8] further defined five types of Mashup styles currently available in the Web 2.0 setting:

1. *Presentation Mashup*: This represents the simplest form of mashup because the underlying data and functionality do not really become integrated. Information and layout is retrieved and either remix or just placed next to each other. Some Web 'widgets' or 'gadgets' fall into this category and so do portals and other presentation mashup techniques.
2. *Client-Side Data Mashup*: This type of data Mashup retrieves information from APIs, services, feeds, and Web pages and remix it with data from another source. Sometimes client-side approach cannot provide Mashup for certain components because of the cross-domain security problem. The cross-domain problem occurs when client-side, e.g. Ajax application, tries to access data in a different domain name.
3. *Client-Side Software Mashup*: In this Mashup style, the scripts that manipulate both contextual data and processes are downloaded to the browser, thereby creating new functionality on the fly. Given contextual data of browsers has been included into certain scripting environment such as Ajax, Mashup of this type has the potential to integrate browser-based software (e.g. Firefox plugins) into novel system features. In this sense, this Mashup style resembles the characteristics of current workflow systems.
4. *Server-Side Software Mashup*: Server-side Mashup tends to have less operational problems due to less security restrictions and cross domain issues. As a result, server-side Mashups such as Yahoo Pipes or many other Mashups listed on the ProgrammableWeb.com are common. However, a key issue lies in the scalability and bottleneck problems caused by the single-point failure at the server side.
5. *Server-Side Data Mashup*: For many years, enterprise solutions for backend data integration (e.g. EAI) have utilized high-level rational database tools to match and combine data at the server-side. While integration of heterogeneous databases is still an open research issue, many DB vendors such as Oracle and Microsoft have made substantial efforts in this area. In other words, data integration at the database level can be seen as a type of Mashup at a very preliminary and low level.

D. Issues with Mashups

The big advantage with Mashups is the agility they provide. However, Mashups do suffer from a number of negative aspects. *First*, due to the lack of semantics and shared vocabulary to describe the business process, Mashups often require intensive custom ("hard") coding effort to access and combine data and functions results from different sets of APIs. For example, a Mashup often needs to interact with many types of data structures and protocols such as Microformats, REST, SOAP, RSS, Atom, JSON. Each of these formats may have multiple versions. *Second*, unlike formal business process specifications which have a number of mature frameworks (e.g. BPMM) to support modeling tasks, Mashups lacks a generic framework that can facilitate the creation, deployment, monitoring, and governance of Mashups of different and abundant Web API and services. Third, as a network-based distributed Web applications, Mashup is not geared towards addressing some fundamental problems inherent in a distributed systems [20] such as concurrency, scalability, fault tolerance, and so on. Such a deficiency has made it very difficult to apply Mashups into the solution of enterprise computing.

VII. A Hybrid Approach

Mashups emerge from the Web 2.0 applications and mainly deals with process and data integration issues that are needed on the Web. The context of the Mashup has revealed a few characteristics that are intrinsic to Web-based workflow systems. First, in order to rapidly and constantly deliver unique value to Web communities, Web developers find it is crucial to have something that is simple to wire different functions and data into a single application that is friendly to the end user through the Web browser. For example, in Ajax, No typing of the resource being accessed is required (i.e.: message types). In contrast, most of today's workflow specifications (e.g. BPEL and WSDL) are strongly typed with respect to both data and behavior (interfaces). It is essential to institutionalize strongly typed specification to detect errors, to maintain overall consistency, and to keep transaction conformance, etc.. From the practice of Web application development, however, we have learned that the overhead imposed by typing and other forms of required artifacts external to the workflow logic itself creates a barrier to entry that excludes most Web developers.

In this paper, we propose a hybrid approach that forms a mixture of both. The motivations for proposing such a hybrid approach is two-fold:

1. Given the massive business opportunities on today's ProgrammableWeb, Workflow based or service composition needs to support composing RESTful Web services in addition to WSDL-based ones. However, given the contrast between these two paradigms of engineering services [11,12], direct merge is not feasible. It is exciting to see that a few efforts [6] have been made that aims to integrate the RESTful services into BPEL, thus forming the RESTful BPEL. However, they still centre around the infrastructure of BPEL.

2. Given the strength and weakness of both BPEL and Mashup, which one should we use? Our answer is: it really depends. For static, stable workflow, we tend to use BPEL. For community driven, transient workflow, we need to use Mashups. Therefore, we need a set of principles or guidelines that help software architects to make wise decisions. This gives rise to the hybrid approach.

We contend that for applications that need to access heterogeneous data and interfaces and that need to be updated on a regular basis, we strongly encourage developers to use Mashups. In particular, if user interaction plays important roles. For stable processes, we intend to adopt the traditional workflow system approach, in which the choreography or orchestration is defined using formal specification with security and fault-tolerance constructs. However, unlike the traditional approach, we encourage a two-step methodology here. In the first Step, developers can use situational Mashup to develop prototypes that carry out trail-and-error experiments. In the next step, once the Mashups are stable, one then migrates them gradually to workflow system with formal description and semantics. However, these two steps need to be conducted in a iterative manner, thus the Mashups are gradually evolving into a stable, formal workflow specification.

It is our belief that that such a hybrid approach can be applied to all distributed system in general, in particular, for Workflow-based system. A pressing issue to realize such a hybrid approach is the semantic representation of resource, services, and processes.

VIII. Conclusions

We envision a full-fledged digital ecosystem that is of a mixture of Service-Oriented, Resource-Oriented as well as a Agent-Oriented paradigm. This has raised several issues for researchers and practitioners to model the dynamics of the DES. In this paper, we propose a hybrid approach that integrates both BPEL-based and Mashup-based methods to tackle the issues for Web-based digital ecosystems.

References

[1] A. Trachtenberg, "A. PHP web services without SOAP," http://www.onlamp.com/pub/a/php/2003/10/30/amazon_rest.html 2003.

[2] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," *PhD Dissertations, University of California, Irvine CA, USA*, 2000.

[3] W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, N. Russell, H. M. W. Verbeek, and P. Wohed, "Life After BPEL?," 2006.

[4] J. R. Erenkrantz, M. Gorlick, G. Suryanarayana, and R. N. Taylor, "From Representation to Computations: The Evolution of Web Architectures," presented at 6th Joint Meeting of the European Software Engineering Conference and the ACM SIFSOFT Int'l Synposium on the FOundations of Software Engineering, Dubrovnik, Croatia, 2007.

[5] A. Fuggetta and G. P. Picco, "Understanding Code Mobility," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol. 24, pp. 342 - 361, 1998.

[6] C. Pautasso, "BPEL for REST," presented at The 6th International Conference on Business Process Management, Milan, Italy, 2008.

[7] E. M. Maximilien, H.Wilkinson, N. Desai, and S.Tai, A Domain-Specific Language for Web APIs and Services Mashups, ICSOC, 2007, LNCS 4749, pp. 13–26,

[8] D. Hinchcliffe, Is IBM making enterprise mashups respectable?, http://blogs.zdnet.com/Hinchcliffe/?p=49, accessed on 13 May 2009.

[9] Faceconnector, http://salesbookapp.com/faceconnector/

[10] C. Wu and E., Chang, Searching services "on the Web": A public Web services discovery approach, in the proceedings of 3rd IEEE International Conference on SIGNAL-IMAGE TECHNOLOGY and INTERNET- BASED SYSTEMS, 16 – 19 Dec 2007, Shanghai

[11] C. Wu and E. Chang Aligning with the Web: An Atom- based Architecture for Web Services Discovery, Service-Oriented Computing and Applications, Vol 1, Issue 2, Springer, London, ISSN: 1863-2386

[12] T. Dillon, C. Wu and E. Chang, An abstract layered model for Web-inclusive distributed computing leading to enhancing GRIDSpace with Web 2.0, on the Special Issue at the Concurrency and Computation: Practice and Experience, Wiley InterScience 2008

[13] Alonso, G., Gasati, F., Kuno, H. & Machiraju, V. (2004) Web Services: Concepts, Architectures, and Applications, Springer Verlag.

[14] Currie, W. L. & Parikh, M. A. (2006) Value creation in web services: An integrative model. Journal of Strategic Information Systems, 15, 153 - 174.

[15] Cong, S., Hunt, E. & Dittrich, K. R. (2006) IEIP: an Inter-Enterprise Integration Platform for e-Commerce Based on Web Service Mediation. IEEE European Conference on Web Services.

[16] Benatallah, B. (2005) Developing Adapters for Web Services Integration. LNCS 3520. Springer-Verlag.

[17] Leyman F. (2006) Keynote at OTM, conference November 2006, Montpellier.

[18] Hammer D.K., Hanish A.A. and Dillon T., (1998), Modeling behavior and dependability of object-oriented real-time systems, Int. Journal of Computer Systems Science and Eng.. 13(3) 1998, 139-150

[19] Goff, M.K.: Network Distributed Computing: Fitscapes and Fallacies. Prentice Hall, Upper Saddle River, NJ (2003)

[20] Coulouris, G., Dollimore, J. & Kindberg, T. (2001) Distributed Systems: Concepts and Design (3rd Edition), Addison Wesley