# SOA Based Design for Migrating Legacy Applications into Multi-Tenant Services

Sidhant Rajam
Rakuten Inc.
Tokyo, Japan
sidhant.rajam@mail.rakuten.com

## ABSTRACT

Multi-tenancy is the core concept of design for modern cloud computing services. Multi-tenant applications leverage the optimum usage of the underlying resources which in turn contributes to the high profit margin by decreasing the expenses on infrastructure. Legacy applications have their own blend of resources and infrastructure which are often not shared by other applications even though all applications are owned by a single organization. Therefore, it becomes an imperative to migrate such kind of legacy applications into multi-tenant services in order to seamlessly share the resources and decrease the cost on infrastructure, operations and maintenance. This paper mainly discusses how the legacy applications can be migrated into multi-tenant services by adopting an approach of Service Oriented Architecture (SOA). SOA based design modularizes an application into the loosely coupled services which can be exposed to any third party applications through an interface. Moreover, SOA based solutions foster high level of synergy into legacy applications in terms of shared resources, unified operations and maintenance, faster development and deployment of the new applications/services etc. This paper presents the empirical and practical notes on effective utilization of the SOA based design to migrate the legacy applications into multi-tenant services. The proposed solutions in this paper offer an alternative to outsource the development and operations tasks which in turn reduces the overall cost for long term sustainable businesses.

## KEYWORDS

Service Oriented Architecture, Cloud Computing, Multi-Tenancy, Web Service, Model-View-Controller Pattern, Dependency Injection Pattern.

## 1 INTRODUCTION

Legacy applications are inevitable pieces of codes that are developed and maintained since few years ago. Such legacy applications are designed to cater a specific business requirement and often use dedicated resources such as hardware, middleware software, database, network devices etc. That means legacy applications are built on a dedicated infrastructure and meant for a particular category of clients. However, recent advent in pervasive/ubiquitous technology has contributed an enormous generation of data at high volume and high speed. Legacy applications are not capable of processing such a massive data from large number of users and applications. Therefore, legacy applications can only be sustained if it is retrofitted with respect to the cutting edge technological standards and architectures. Cloud computing is an advanced trend of delivering the business services over its Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS) models [1]. Cloud computing uses a technique of multi-tenancy in order to share the resources in order to bring good values to handle massive scale of data and users [2].
Multi-tenancy can be achieved at various levels [3,4] such as Amazon AWS is a multi-tenant at hardware level so that its clients may be sharing a physical machine [5]. On the contrary, Force.com is a multi-tenant cloud service at the database level so that its clients can share data in the same database tables [6]. Amazon AWS uses hypervisor to provide isolation between different tenants. And Force.com uses a query rewriter to provide isolation between different

tenants. Both of these approaches (such as Hypervisor isolation and Database isolation) are valid however they influence different categories of clients and audiences.

In this paper, we discuss the Service Oriented Architecture (SOA) based approach to migrate the legacy applications into multi-tenant services. We consider a typical legacy application built for advertising systems which support various advertising applications or business models such as Cost per Click (CPC), Cost per Duration (CPD) etc. These individual advertising applications are web based applications which are designed and deployed in an isolated environment. In this paper, we set the tenancy level at such kind of isolated applications and at their clients. In section 2, we explain various issues and pitfalls related to the legacy application. Section 3 elucidates the meaning and merits of multi-tenant system. It also discusses why it is necessary to transform the legacy applications into multi-tenant services. In section 4, we explain the benefits of SOA for cloud based systems that extensively make use of multi-tenant services. In section 5, we discuss the SOA based design to migrate the legacy applications into multi-tenant services. We also briefly discuss how Model-View-Controller (MVC) design pattern can be adopted in SOA to achieve various system and service level benefits. Moreover, we also explain how to define and deploy the service in this SOA based design. The conclusion is discussed in section 6.

## 2 ISSUES IN LEGACY APPLICATIONS

Single tenant legacy applications are mostly not compliant to work in tandem with the cutting edge technological improvements and trends. Figure 1 shows an example of a typical legacy application which is designed to cater various advertising business models such as CPD, CPC and so on. In this example, each advertising application such as CPD and CPC are built on dedicated infrastructure including hardware, network devices, middleware software such as web and app server, development platform, core business engine etc. Such kind of vertical and single tenant design cannot achieve optimum utilization of the given resources. In turn, it elevates the cost on infrastructure, development, operations and maintenance because roll out of any new advertising application (such as NEW_2 XXX) require complete new set of redundant resources. In this example, each advertising application needs 8 instances of Apache-HTTP server, 16 instances of Tomcat App Server, separate network domain name or IP address etc.

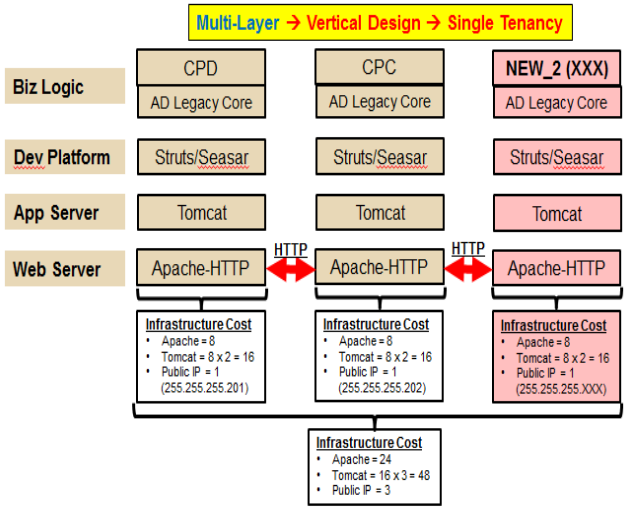


**Figure 1. Legacy application (advertising applications).**

Such high redundancy in resource utilization increases the budget on time to market. Each application is tightly coupled with its core libraries and development platform. If any application needs to invoke another application, it has to call the corresponding APIs over web (HTTP) even though the architecture is same. Such kind of HTTP invocation is an overhead and it delays the overall response time. Moreover, the underlying hardware for each application may not be optimally used because some applications can be processing heavy traffic while some applications may not be busy. In a nutshell, these advertising legacy applications cannot withstand against the large number of users and heavy traffic scenarios. Therefore, legacy applications cannot linger

around in any organization for a long time because of continually developing technologies and new dimensions of user requirements. This situation can be improved by sharing underlying infrastructure resources, technologies and architecture.

## 3 MULTI-TENANT SERVICES

The productivity of the legacy application can be leveraged by migrating it on cloud based services which are multi-tenant at various levels including hardware, platforms, applications, users etc. [3]. In multi-tenant, a tenant is an entity which consumes a service offered by its service providers. Multi-tenancy based designs shares hardware and software platforms to achieve higher synergy of available resources. Thereby, it significantly decreases a cost on infrastructure, licensing, operations, maintenance and development [2,4]. It also simplifies the data mining and aggregation. Moreover, it streamlines the release management process.

In Figure 1, the advertising applications are legacy single tenant applications and lack resource sharing and reusability. These factors lower the sustainability prospects of such applications in the long run. Therefore, it paves a big room for introducing multi-tenancy based design into these advertising applications. However, to transform such legacy applications into multi-tenant services need complete replacement and redevelopment which is very expensive and time consuming process. In the subsequent sections, we will discuss how SOA based design can simplify the migration of legacy applications into multi-tenant services.

## 4 SERVICE ORIENTED ARCHITECTURE

A service is a computational logic that is wrapped in an interface which can be exposed to external entities over standard protocols. When a service interface is exposed over web via HTTP or HTTPS protocol, it is called as a Web Service (WS) [7]. Hence, each service is composed of a triplet such as Contract or meta-data that encapsulates the details of Activities or computational logic which can be consumed over network via certain standard Protocols [7]. Contract hides the low level implementation details (Activities) of the service from its consumers. Therefore, each service is loosely coupled at an abstraction level. Figure 2, shows main characteristics or traits of SOA.

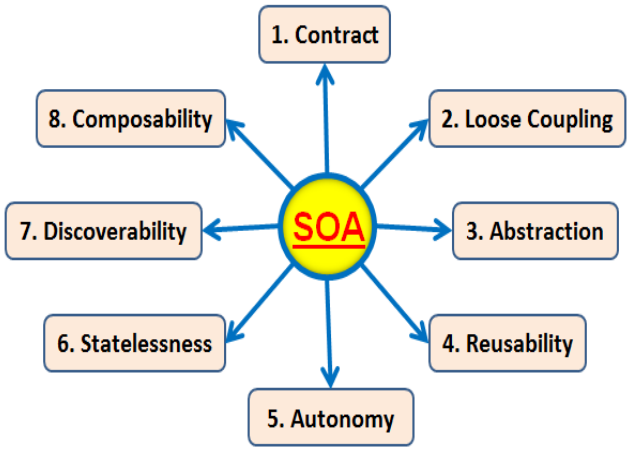


**Figure 2. Traits of Service Oriented Architecture**

All services are abstracted for its internal technological details and thereby it can be reused at any environment or framework without any dependency. Services can be maintained autonomously by the service provider who can publish the interface to the service consumer or to the common interface repository. The services can be discovered from such kind of repository that is also called as Universal Description, Discovery and Integration (UDDI). Multiple services can be integrated and composed to form a workflow in order to deliver a solution for a complex problem or business requirements [8,9].

In multi-tenant environment, one application can cause an error due to other malfunctioned applications. However, this problem can be solved by using SOA approach because each service is defined uniquely and independently. In SOA, the composition of workflow can be dynamic due to the techniques of service orchestration and choreography. The detailed description of these techniques is out of scope of this paper.

## 5 SOA BASED MIGRATION

Single tenant legacy application needs to be redeveloped or replaced in order to transform it in cloud based multi-tenant services. As discussed above, such kind of top-down re-engineering tactics are very expensive and time consuming. However, SOA based design can significantly alleviate such a high cost on development and time because each service is an independent entity of a triplet including Contract, Activities and Protocols. Therefore, it is not necessary to re-engineer or redevelop entire legacy application. Rather bottom-up approach can be used to just retrofit the application in SOA based design. It requires an initial and onetime cost of designing a SOA core engine. This engine drives the execution and exposition of all the services. The more details of SOA based migration is discussed in the following subsections.

### 5.1 SOA based Multi-Tenancy Design

Figure 3, shows the SOA based multi-tenant advertising applications. The advertising application is considered as an example and the SOA scheme depicted in Figure 3 can be used for any legacy application which is single tenant and vertically designed without shared resources.

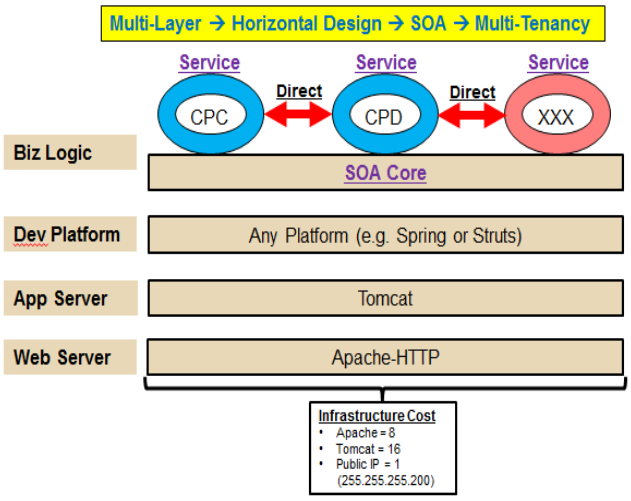


**Figure 3. SOA based migration to multi-tenant services.**

SOA core is an engine that acts as an environment for the services to deploy and execute. SOA core is built on any kind of development platform such as Spring or Struts. One of the main differences between single tenant (Figure 1) and multi-tenant (Figure 3) architecture is that the underlying resources such as development platform, application server, web server, hardware and network infrastructure are shared. This is an horizontal design that does not require redundant set up of new resources for every new application or services. Cost of infrastructure, operation, maintenance is reduced because of shared resources. Only one network domain name or IP address is required to expose all the services. Services can be consumed over REpresentational State Transfer (REST) protocol or HTTP POST method. Each service is identified and invoked by unique service id. Service id can be a context root of the main service URL e.g. http://www.adservices.com/CPC/?[query_request_parameters]. In this URL, www.adservices.com is the shared advertising service URL and /CPC/ represents CPC service or application. The query_request_parameters are the request parameters for the CPC service. In this way, any advertising service or application can be consumed over shared resources. In this example, the tenants are the advertising services (e.g., CPC, CPD etc.) and end clients of these services. Multiple services and clients consume the single instance of the shared resources via SOA core engine. Session and transaction management for each service is processed distinctly and thereby the data traffic for every service is managed in isolation.

Moreover, we have achieved a high reusability of the advertising services compared to legacy applications as shown in Figure 1. All advertising services can reuse and compose new workflows by directly accessing each other at the same layer of network communication. The services do not need to make expensive HTTP Web invocations. This speeds up the

overall processing of the service and improves the response time.

### 5.2 Model-View-Controller Pattern for SOA

Model-View-Controller (MVC) design pattern is the most prevalent pattern for designing and organizing the web based solutions [8,10]. MVC is a highly modular, reusable, expressive, and agile software development pattern.

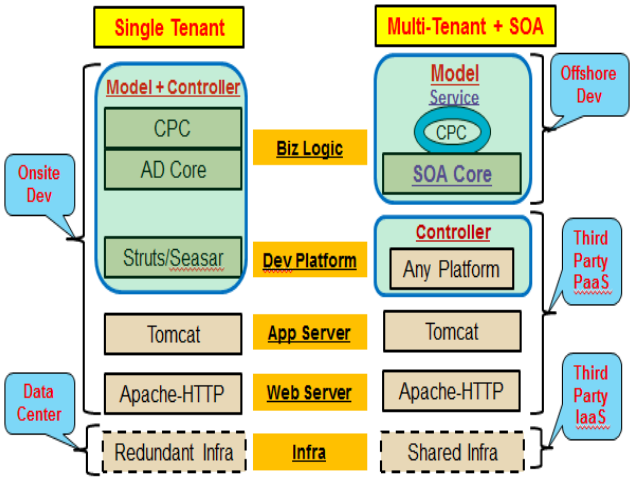


**Figure 4. Benefits of MVC in SOA for multi-tenancy.**

Figure 4 shows an adaptation of MVC pattern in both single tenant and multi-tenant environment at various layers of application development stack. For any organization, the cost on software development and infrastructure is very crucial. However, this cost can potentially be decreased by outsourcing the development and infrastructure to third party. Cloud computing's SaaS, PaaS and IaaS offerings are well suited for these outsourcing or offshoring requirements [1]. In Figure 4, left hand side shows an application development stack of single tenant legacy advertising applications. As discussed in section 2, legacy applications are tightly coupled with their underlying resources including core engine, development platform, application server, and web server. Likewise, MVC's model and controller components are tightly coupled and needs to carry out the development process locally at the onsite. For every new legacy application, there is a separate stack of hardware servers, data or IO storage, network devices (load balancer). Redundant usage of infrastructure for each application makes it very difficult to host the applications on the IaaS. Hence, the infrastructure is maintained locally at the data centers. Such kind of single tenant design is not sustainable because of lack of economical scalability.

In Figure 4, right hand side shows an application development stack of migrated multi-tenant advertising services. As an inherent loose coupled characteristic, SOA core engine is independent of any kind of development environment such as Spring, Struts etc. SOA core contains only services which are model of MVC. And the development platform becomes controller of MVC. Such demarcation between model and controller provides two main advantages. Firstly, the source code development of advertising services can be outsourced to offshore which is highly cost effective compared to that of onsite development. Third party advertising services can also be consumed over SaaS cloud model. Secondly, since SOA core is not tightly coupled with any particular development platform, PaaS cloud model can be utilized to outsource the platform.

As discussed in section 5.1, multi-tenant services share the infrastructure resources irrespective of any number of applications or services deployed in it. Thereby it becomes very easy to outsource the infrastructure on IaaS cloud. In this way, multi-tenant design based on SOA approach provides a big room for significant cost cutting in development, operations and maintenance by allowing SaaS, PaaS and IaaS level cloud outsourcing.

### 5.3 Service Deployment

This subsection discusses the low level details of how to deploy a service in SOA core engine. As discussed in subsection 5.1, SOA core engine deploys, executes and exposes the

service. Figure 5 shows a model of service deployment and service definition in a multi-tenant design. In a multi-tenant environment, SOA core runs on the shared infrastructure of development platform, tomcat application server and Apache-HTTP web server. Apache-HTTP web server runs the HTTP service to interact with the end clients over Web. Tomcat server acts as an application container which provides a runtime environment of the SOA core and services to execute the business logic. Moreover, it manages the low level network calls, multi-phase transaction management, and other non-business routines which are necessary for any development platform to host the SOA core engine. Development platform provides certain library utilities necessary for data marshalling/non-marshalling, session management of a particular transaction with respect to a specific service and Application Programming Interface (API). Development platform also facilitates the design patterns such as Dependency Injection (DI), MVC, Enterprise Service Bus (ESB), proxy etc. These design patterns can be used to organize the service definition and composition [11].

However, the use of development platform is non-mandatory. Basically SOA core engine requires an application container to execute and expose the services. Development platform can be added as a plugin to utilize above mentioned commonly used library utilities and framework of various design patterns. The main benefit of the DI pattern is that it hides the service logic implementation that is not apparent from the interface [12]. Furthermore, DI pattern removes the dependency from concrete implementation of the business logic. The ESB is built on SOA based methodologies. The ESB is mainly designed for the service virtualization which is a service plumbing that concentrates on composing new services without bothering about how the services are exposed, managed and consumed [13]. The ESB is a robust integration and connectivity infrastructure for integrating services that exposes WebService (WS), intelligent routing, messaging middleware, orchestration and transformation. Such kind of features can be extended in SOA core for an efficient organization and development of services.

Definition of a service is a basic building block in the service deployment process [7]. As shown in the Figure 5, Service definition includes four basic entities as following:

***(1) Request Parameters:*** This is mainly the request items set by the client and the control parameters set by the application container such as Tomcat server. The request parameters are provided to the service endpoint layer through a technique called as marshalling and unmarshalling. The request parameters contain instructions for the service endpoint. The tomcat server is also a client for the service and its control parameters include the data of runtime environment e.g. transaction context, session context, etc.

***(2) Response Parameters:*** This contains a result of the service execution. The response parameters are sent back to the client by converting its data format through a technique called as marshalling and unmarshalling. The request and response parameters are defined in Service Interface.

***(3) Service Interface:*** This defines the signature of the service's offerings in terms of function names and Request/Response parameters [3]. This interface is written in definition language such as Web Service Description Language (WSDL). This interface definition can be stored in UDDI and exposed to the potential service consumes for invocation. However, UDDI is non-mandatory facility and the service interface can be directly exposed are shared with the target clients.

***(4) Service Endpoint:*** This is an actual service implementation for the business logic [3]. The service endpoints implements all the functions defined in the Service Interface and manipulates the Request and Response Parameters. The service endpoint can be implemented and modified in any kind of programming languages. The low level implementation details and artifacts of the

service endpoints remains abstract to the third party clients because it is resolved or exposed by a service interface.

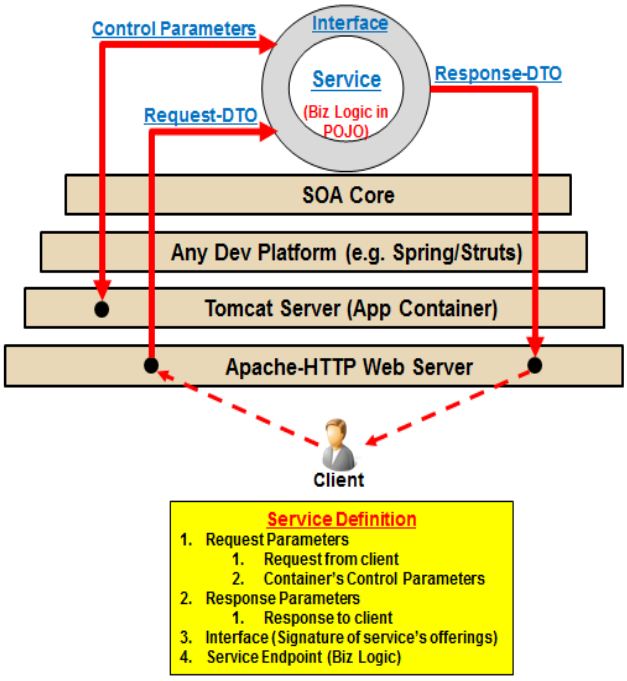


**Figure 5. Definition and deployment of service.**

During service execution, Service Endpoint is executed by the SOA core on top of the runtime environment provided by development platform or by the application container such as Tomcat server.

As explained in the Service Definition, every request parameter contains control parameters of the container. These control parameters hook the service endpoint with the runtime environment throughout the lifetime of the service execution or transaction [7]. The result of the service such as success, failure or exception is also notified to the runtime environment through these control parameters.

## 6 CONCLUSION

In a nutshell, the proposed design of SOA based migration of the legacy applications into the multi-tenant services has the potential to provide a higher sustainability prospect to the legacy applications. SOA based approach needs a bottom-up approach to transform the single tenant legacy application into multi-tenant services without a high cost on redevelopment and time to market is also fast for such services. SOA based design modularizes the legacy application into the loosely coupled independent services which can be exposed to the consumers via an interface or contract over standard protocols such as HTTP/HTTPS [8,9]. Thereby, existing eco-system of the legacy application does not change; rather it gets enriched to cope up with the trends of cutting edge modern technologies by becoming compliant to the cloud computing based SaaS, PaaS and IaaS models of service delivery.

The main advantage of multi-tenancy is a seamless sharing of the infrastructure resources such as software, platforms, hardware, network and IO devices. Such kind of resource sharing achieves optimum synergy of available resources to increase the productivity and throughput of the services. In the proposed SOA based approach, the tenants are either the services or end clients of these services. Multiple services and clients consume the single instance of the shared resources via SOA core engine. Session and transaction management for each service is processed distinctly and thereby the data traffic for every service is managed in isolation in order to secure and maintain data privacy [14]. The proposed SOA based solution also discussed the adaptation of MVC design patterns and its benefits on how to manage and operate the services remotely. This has suggested an alternative to outsource the development and operations tasks which in turn reduces the overall cost for long term sustainable businesses.

## REFERENCES

[1] P. Mell, and T. Grance, 2011. Definition of Cloud Computing. The National Institute of Standards and Technology (NIST). United States Department of Commerce Special Publication 800-145 (Sep. 2011). DOI= http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[2] S. Kajeepeta, 2010. Multi-tenancy in the cloud: Why it matters. Article in Computerworld (April 2010). DOI= http://www.computerworld.com/article/2517005/data-center/multi-tenancy-in-the-cloud--why-it-matters.html.

[3] A. Azeez, S. Parera, D. Gamage, R. Linton, P. Siriwardana, D. Leelaratne, S. Weerawarana, and P. Fremantle, 2010. Multi-tenant SOA Middleware for Cloud Computing. In Proceedings of the Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on Cloud Computing (Miami, FL, US, July 05 - 10, 2010). CLOUD2010, IEEE, 458-465. DOI= http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5557959.

[4] S. Walraven, E. Truyen, and W. Joosen, 2011. A Middleware Layer for Flexible and Cost-Efficient Multi-tenant Applications. In Proceedings of 12th ACM/IFIP/USENIX international conference on Middleware (Lisbon, Portugal, December 12 - 16, 2011). Middleware'11, Springer-Verlag, Berlin, Heidelberg, 370-389. DOI= http://dl.acm.org/citation.cfm?id=2188487.

[5] Juniper Networks. 2012. Securing Multi-Tenancy and Cloud Computing. Whitepaper of Juniper Networks, Inc (March 2012). DOI= https://www.juniper.net/us/en/local/pdf/whitepapers/2000381-en.pdf.

[6] S. Bobrowski, 2008. The Force.com Multitenant Architecture Understanding the Design of Salesforce.com's Internet Application Development Platform. Whitepaper of Salesforce.com. DOI= http://www.developerforce.com/media/ForcedotcomBookLibrary/Force.com_Multitenancy_WP_101508.pdf.

[7] T. Erl, 2008. SOA Principal of Service Design. Prentice Hall. Boston, US. pp. 25-160.

[8] S. Rajam, R. Cortez, A. Vazhenin, and S. Bhalla, 2010. Enterprise Service Bus Dependency Injection on MVC Design Patterns. In Proceedings of TENCON 2010 - 2010 IEEE Region 10 Conference (Fukuoka, Japan, November 21 - 24, 2010). TENCON2010, IEEE, 1015-1020. DOI= http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5686452.

[9] S. Rajam, R. Cortez, A. Vazhenin, and S. Bhalla, 2010. E-Learning Computational Cloud (eLC2): Web Services Platform to Enhance Task Collaboration. In Proceedings of 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (Toronto, ON, Canada, August 31 - September 3, 2010). WI-IAT2010, IEEE, 350-355. DOI= http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5614599.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, 1994. Design Patterns Elements of Reusable Object Oriented Software. Addison-Wesley, New York.

[11] U. Zdun, 2008. Pattern-based design of a service-oriented middleware for remote object federations. ACM Transactions Internet Technology (TOIT),vol. 8,issue 3 (May. 2008), 1-38. DOI= http://doi.acm.org/10.1145/1361186.1361191.

[12] M. Fowler, 2004. Inversion of Control Containers and the Dependency Injection pattern. (January 2004). DOI= http://martinfowler.com/articles/injection.html.

[13] M.T. Schmidt, B. Hutchison, P. Lambros, and R. Phippen, 2005. The Enterprise Service Bus: Making service-oriented architecture real. Journal IBM Systems Journal,vol. 44,issue 4 (2005), 781-797. DOI=http://dl.acm.org/citation.cfm?id=1126978.

[14] S. Rajam, R. Cortez, A. Vazhenin and S. Bhalla, 2010. Design Patterns in Enterprise Application Integration for e-Learning Arena, 13th International Conference on Humans and Computers (HC2010) ACM In Cooperation, Aizu-Wakamatsu, Japan, pp 81-88, December 8-10, 2010. DOI=http://dl.acm.org/citation.cfm?id=1994508.