International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Multi-Tenant SaaS Application Platform: A Survey

Basant Kumar Gupta, Keisam Thoiba Meetei

PG Scholar, Computer Science and Engineering, Galgotias University, Greater Noida U.P, India

Abstract: ***Now days cloud computing is a very noble Internet based on-demand, pay-as-per-use, utility based computing platform which provides software as a service to the end user as rent basis to reduce the cost of purchasing the software as well as reducing the maintenance costs which is more sophisticated to the end user. It makes IT industry more available at the door step of the users. But here security measures over the cloud-hosted platform are less. A multi-tenant based SaaS application platform provides a security framework which improves collaboration between cloud service providers and cloud service consumers. Multi-tenant database allows multiple instances emulates single instance which provides a SaaS security framework model. Due to sharing of a single application among the multiple tenants brings down the overall cost of IT infrastructures. In this recent era of IT infrastructure Software-as-a-Service (SaaS) is becoming a dominant technology which utilize a standardized SaaS application developed by SaaS application developer. There is an application, metadata driven architecture introduced by Force.com satisfy the multitenancy of SaaS application.***

Keywords: Cloud computing, security framework, multi-tenancy, SaaS, metadata

1. Introduction

Software-as-a-Service (SaaS) is a cloud computing service delivery model of software that provides an application for multiple users via internet as a form of 'on demand service' [1]. By subscribing SaaS service, companies can use various IT services without the need to purchase and maintain their own IT infrastructure [2]. The SaaS service provider can offer SaaS services at a moderate price by making full use of the economy of scale. The maturity of SaaS service can be defined by several maturity models [3,4]. Three key attributes – Multitenancy, Scalability and Configurability are widely used in these models. Microsoft proposed SaaS Simple Maturity Model [3] which described maturity of SaaS architecture with four maturity levels. At the highest maturity level, scalability is added through a multi-tier architecture with a load-balancing feature and the system's capacity can be increased or decreased by adding or removing servers. The Maturity Level 3 provides Multitenancy which represents the ability that enables SaaS application to serve multiple tenants using a single service instance. At this level, tenants not only have feature to configure some aspects of SaaS software such as UI or data model but also feels as if they are using a dedicate server while every tenant are sharing a single server and service instance. In terms of operation cost, the benefit of economy of scale can be achieved by the Maturity Level 3. Configurability is achieved at the Maturity Level 2. Although it still requires dedicated server for each tenant, identical instances can be used because it provides a feature for tenants to configure some aspects of SaaS software as they want. The lowest level (Level 1) represents the Application Service Provider (ASP) model [5] which requires a dedicated server and service instance for each tenant (i.e. a company subscribing the service). At this level, maintaining cost of the service provider is high because it requires multiple different instances for different tenants. The multitenant architecture with single instance (Maturity Level 3 or above) was implemented by Salesforce.com [6] by introducing their metadata driven architecture and APEX programming environment.

Security is considered one of the peak ranked right of admission issues in adopting the cloud computing model, as reported by IDC [7]. A reasonably priced justification of such increasing concerns of the CCs roughly cloud security [8] includes: (1) The loss of control unfriendly than cloud hosted assets (CCs become not accomplished to hold their Security Management Process (SMP) harshly the cloud hosted IT assets); (2) The nonappearance of security guarantees in the SLAs surrounded by the CPs and CCs; and (3) the sharing of resources when than competitors or malicious users. Accordingly, no business how strongly the model is secured, consumers continue encumbrance from the loss of plan and nonappearance of trust problems. On the appendage hand, the CPs be in pain moreover than the cloud platform security issues because the cloud model is no evaluate unknown and has a lot of dimensions that must be considered surrounded by developing a holistic security model [2] including the unnamed architecture of the cloud model, the model characteristics, the long dependency stack, and the swing stakeholders security needs. These dimensions outcome in a large number of heterogeneous security controls that must be consistently managed. Approach introduces here a connection cloud security supervision framework based just roughly aligning the NISTFISMA within permissible limits [9], as one of the main security running standards, to fit associated to the cloud architectural model. The instruction required to put the NIST up to customary into effect is not possessed by one party. Thus we union the collaboration amid the key cloud stakeholders to share such required warn. Getting CCs working in all step of the SMP of their assets mitigates claims of loosing trust and counsel. This admittance helps stakeholders to residence the behind issues:

- What are the security requirements needed to protect a cloud hosted further immovable that the designate support to is used by oscillate tenants at the same era?
- What are the seize security controls that mitigate the promote adoption risks and who pick such controls?
- Are the fixed controls easy to attain to about the cloud platform or we will/can use third party controls?

- What are the security metrics required to play a role the security status of our cloud-hosted facilities?

2. Related Work

Initial play a role around multi-tenancy was curtains in the context of Application Service Providers (ASPs) [10], which was a nod that happed in late 1990s and abet on 2000s. The ASP concept had goals totally the same to the goals of the SaaS model and motivated multi-tenancy. However, these approaches focused upon giving out an instance per tenant, and they slip asleep the level 1 of maturity model proposed by Chong et al. Data multi-tenancy is the most explored right of entry below multi-tenancy, and is often implemented upon peak of a database. Both Jacobs et al. [11] and Chong [12] have outlined three main approaches for data outlook in a multitenant deployment- (a) surgically remove databases: gives each tenant its own database, (b) shared database when surgically remove schemas: gives each tenant its own tables, and (c) shared database along in the middle of shared schemas: shares the same table in the middle of many tenants and enforces security at the adjacent lump in the architecture. Much of the existing research upon multi-tenant SaaS have focused upon shared data architecture and security running [13, 14, 15, 16], and middleware extensions to habitat the expertly-founded concerns due to data/security/hatred. The work of [17] develops a multi-tenant placement model which decides the best server where a new tenant should be accommodated. In principle, a new tenant will be placed on the server with minimum remaining residual resource left that meets the resource requirement of the new tenant. There have also been studies on service performance issues in multi-tenant SaaS [18]. In contrast, there has been relatively little research so far on the impact tenant variability may have on the functionality and evolution of a SaaS system over its lifecycle. This is not surprising given that SaaS is a relatively recent phenomenon, and hence the initial focus is bound to be on issues that are related directly to its feasibility (such as security or performance). Models and techniques successfully employed in software product line engineering [19] have been applied in multi-tenant systems to manage configuration and customization of service variants. In particular, [20] extends variability modeling [21], which provides information for a tenant to customize the SaaS application and guides the SaaS provider for service deployment. The work of [22] discusses some potential challenges in implementation and maintenance of multi-tenant systems. It presents an architectural approach which tries to separate the multi-tenant configuration and underlying implementation as much as possible, by adopting the 3-tier architecture (authentication, configuration, and database) in the traditional single-tenant web application. Along the same lines, experiences in modifying industrial-scale single-tenant software systems to multi-tenant software have been reported in [23]. This involves extending user-authentication mechanisms, introducing tenant-specific software configuration and adding an application layer to extract tenant-specific views from the shared database. A recent paper [24] moreover studies tenant specific customizations in a single software instance, combined tenant setup.

Aulbach et. al. [25] portray such a schema mapping technique where they introduce the concept of Chunk Folding, where the logical tables are vertically partitioned into chunks and placed in every abnormal being databases, and allied considering vital.

Guo et al. [26] discuss carrying out and administration superiority, and considering the proposed recognition, they have achieved it by limiting the fan to the portion of the manual hierarchy and using Platform specific security modes (e.g. Java security) to ensure hostility. Moreover, they use data encryption to attach stored data. They auxiliary come occurring following the money for take effect isolation through controlled resource allocations (e.g. quota, monitoring resource usage and enforcing priorities), which prevents a tenant from monster affected by load upon supplement tenants.

Chong et al. [27], in one of the first discussions of multitenant applications, have proposed a maturity model where fused numbers indicate plus level of resource sharing. For instance, level 1 provides an instance per tenant, level 2 provides a configurable instance per tenant, level 3 runs a single instance that serves all customers, and finally, level 4 enables level 3 to scale taking place by government merged instances and load balancing to scale it going on.

Menzel et al [28, 29] proposed a model driven recognition and a language to specify security requirements upon web facilities and cloud web applications composed of web facilities. Each application instance (and its facilities) is deployed upon a VM. They assumed that (1) web applications are composed of web services unaccompanied, (2) multi-tenant security is maintained through using VMs for each tenant (simplest act), and (3) the underlying infrastructure security is not considered.

3. Configurability of SaaS Application

In this section we will define SaaS application that would be provided by the SaaS platform and metadata driven architecture would be explained to show how multitenancy is satisfied.

3.1 SaaS Application

The SaaS application operating on the proposed SaaS platform is one packaged business application with web based user interface to multiple tenants. The purpose of business application such as CRM, ERP, or Groupware is processing business transactions and collaboration among tenants' users with business data in DBMS as the center. The three tier architecture as in Figure 1 is widely used for operating business application.

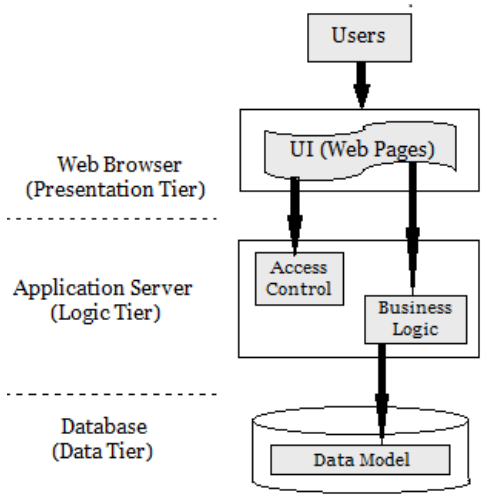

Figure 1: Execution of Business Application

Data objects for a business application are defined in Database Management System (DBMS). Different users in a tenant have different roles such as boss, manager, or employee and every role set have their own authorities for accessing company data and business logics. Based on role sets and their access authority, web pages are provided for users to request services through web browsers. Requests are transferred to the Application Server which contains business logics that processes business transactions. The main aspects of a business application consist of data model, business logic, web pages, and role sets. In addition to business applications, Business Process Management (BPM) application is provided by the proposed SaaS platform.

3.2 Multitenancy Via Metadata

Since we provide SaaS application for multiple tenants with single service instance the platform architecture needs to enable self configuration by tenants without changing the SaaS application source code for individual tenant and runtime configuration not to suspend service during the configuration. Metadata driven architecture by Salesforce.com [30] provides solution for self- and runtime configuration of SaaS application. Figure 2 shows the concept of metadata driven architecture.

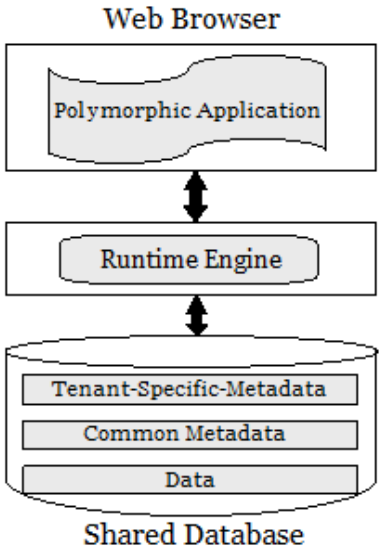

Figure 2: Metadata driven architecture

In this architecture every aspects of SaaS application that are configurable by tenants are stored in metadata database. When a tenant manager configures some aspects of SaaS application, configured information is stored separately in tenant-specific metadata. Runtime engine generates polymorphic application for individual tenant using application codebase and tenant-specific metadata at runtime. Through the polymorphic application, tenant users feel as if they are using their own business application while service instance is shared by every tenant. Even though application data are shared, they are kept secure because the polymorphic application for individual tenant accesses application data independently via optimized query for each tenant.

3.3 The Conceptual Architecture

The SaaS platform provides SaaS Application execution environment that serves multiple tenant using a single service instance. To do this, the platform is composed of several key components – configurator, runtime engine, metadata management system, and so on. Figure 3 depicts the conceptual platform architecture for the SaaS platform. Detailed descriptions of the components of the target platform are given in following subsections.

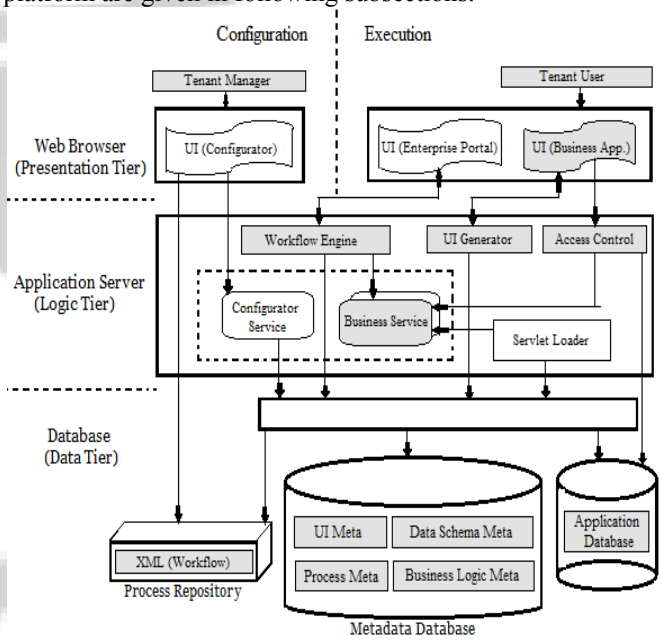

Figure 3: Conceptual architecture for SaaS platform

3.3.1 Client Application

Tenants can use SaaS services through a web browser. Enterprise portal provides a workflow application and interfaces to various business applications. When the web browser sends HTTP request, the Tomcat Application Server provides the web application developed in ExtJS based JavaScript library and responds to JSON request of the web application.

3.3.2 Configurator

Configurator is an ExtJS based web application used by the tenant manager for configuring various aspects of SaaS application. Configurable aspects in this platform are UI pages, organizational structure of the tenant, data models, workflows, and business logics. Configurator provides a

drag-and-drop interface to configure a given web page by arranging ExtJS containers and components in the web page. Moreover, tenant manager can create new web page with various ExtJS containers and components. In the platform, each container and component has unique ID. IDs of container and component that compose a certain web page are stored as UI metadata with a page. When the web page is requested by a tenant, the web page is dynamically generated with the metadata and associated application data. Configurator provides a graphical user interface to configure data model and organizational structure and access authority. Using the configurator, like a database client program, the tenant manager not only is able to view data object and its data fields but manipulates their information and access policy. Configuration of workflows and business logics can be done by a design interface of the configurator. Using the interface, the tenant manager can modify a workflow with several features – rearranging order of activities, changing assigner of an activity, and so on. The tenant manager can compose new workflow with predefined activity types.

3.3.3 Runtime Engine

The SaaS application that operates on the proposed SaaS platform is one packaged business application with web-based user interface to multiple tenants. The purpose of business application such as CRM, ERP, or Groupware is processing business transactions and collaboration among tenants' users with business data in DBMS as the center. The three tier architecture is promoted for operating business application as Figure 1. Configured aspects of SaaS application by the tenant manager are stored as metadata in the metadata database. While, codebase developed by the application developer is stored in the application database. Runtime engine plays a role in generating tenant specific application using codebase and metadata. There are mainly three subcomponents of runtime engine: UI Generator, Workflow Engine and Servlet Loader.

3.3.4 Metadata Management System

Metadata Management System provides two key features for supporting multitenancy. The first one is an access control for supporting multiple tenants. When a developer makes business logic in a SaaS application, he cannot help making complicated SQL query with tenant's ID and tenant configured data object and field name. To avoid this painful task, Metadata Management System provides metadata APIs for Logic Tier. By utilizing metadata API, Logic Tier can access shared database regardless of tenant's information. When the web browser sends request to the Application Server, business logic access to database with metadata API. Then, Metadata Management System converts the request to the optimized query to retrieves tenant specific UI pages and data. It provides secure and independent data access to shared database for each tenant. The other feature is providing extended fields for data object. At development stage, it is impossible for the developer to predict what data models and data fields the tenant manger added in the future. In this platform, therefore, ten fields are added to data model when the data model created. However, the usage of these extended fields is different for each tenant. For example, some of tenants would use three of ten extended fields for customer data model but others would use nothing. Another tenant would use the first extended field as logined at field to customer data model as Date type but the other would use it as phone number as Varchar type. Therefore, information of extended fields for every tenant should be managed in Metadata Management System. This information is used for retrieving tenant specific UIpages in runtime.

4. A Model FOR Multi-Tenant SaaS

A multi-tenant SaaS system has to be carefully designed to handle the variability that can arise due to the differing needs of tenants. At an abstract level, a SaaS system may be considered as a collection of services, where each service in turn, consists of a collection of operations that can be invoked by clients. The functionality desired by different tenants out of a service or operation may differ, thereby necessitating support for variants of these entities. As the existing literature shows [31], concepts from product-line engineering may be adopted to define variation points to which different variants may be linked, and the variability model may also be used to guide SaaS customization. Moreover, the packaging and deployment of the SaaS may be guided through a set of multi-tenancy patterns that help distinguish between components that are shared between all tenants or are specific to some tenants [32]. Technically, these constructs provide the basic foundation for supporting variability within a multi-tenant SaaS application architecture. In the SOA world such a representation has already been explored by the semantic web community to facilitate service discovery, matching or composition, leading to formalisms like OWL-S [33]. There is a believe that a similar approach can also be taken to establish the semantic underpinnings of a multi-tenant SaaS solution. Given such a semantic model for SaaS, the on boarding of tenants poses interesting optimization problems. The requirements of a tenant may be represented in terms of services and operations, and we may expect these requirements to be a mix of mandatory and optional, which provides a basis for negotiation with the SaaS vendor. Given a tenant's requirements profile, the vendor would like to identify the optimal subset of requirements it should support, so that its net profit is maximized while leading to the best commonality in the resultant system. The vendor's profit would be the difference between the expected revenue from the services/operations based on the tenant's anticipated usage profile, and the cost of additional development, which in turn will depend on the degree to which existing services/operations may be re-used e.g. through refinement. A variation-oriented semantic model for multi-tenant SaaS can thus provide a sound basis for a controlled evolution of the system. Apart from tenant on boarding, it can also help in the testing and re-factoring of such systems, as we discuss next.

5. Multi-Tenancy in SOA

To endure the SOA multi-tenancy misery, focus a propos the high level multi-tenancy architecture proposed by Chang et al. [34] where they identified metadata facilities, data facilities, process and issue facilities, security facilities, and presentation components as rotate aspects of the architecture (High-Level Architecture section in [34]). SOA applications plus connect taking place above aspects apart from the

presentation enhancement. We outfit above facilities out cold Execution, Security, and Data. Typically, triumph involves developing, deploying and doling out facilities, which are often implemented as Web facilities, and composing those services together to make sophisticated level artifacts in imitation of Business Processes, Workflows and Mash ups. Executions occupation issue and process services defined in Chang et al. Those executions may gathering and use data either from a registry, or a database, and lineage metadata in a registry. Security services elaborate the ownership and endorsement of both data as neatly as executions in the framework. As vitriolic out by Chang et al. [34] as expertly as by added publications, the want of multi-tenancy is to present swap users of the system (which we shall call tenants) estrangement in each of these spaces even though maximizing resource sharing. However, as is often the squabble, resource sharing and disaffection are a tradeoff. Furthermore, Chang et al. [34] have proposed three properties for multi-tenancy in adjunct to estrangement: scalable, multi-tenant-efficient & configurable. Here, multi tenant efficient means that same instance hosts compound tenants, a requirement for maximizing resource sharing. In a multi-tenant framework there is always a risk that the tenant disaffection is compromised due to a malfunctioning component or a chance programming error. Furthermore, forward the framework is often meant to be extensible, the risks are well along making design and money harder. Therefore, providing an on your own freshen per tenant/users in the SOA framework is a challenging encumbrance. An outlook that registered as a tenant should be nimble to rule/administer its own users, data and services. But they should be restricted from administration functionality of the overall application such as shutdown and viewing system logs. Only the administrators of the infrastructure (for whom we use the term super-admins) will be practiced to entry these functions in order to bolster on desist & goings-on the paperwork SOA framework. The application core should be familiar of handling this two level of access rule and component authors should purposefully strengthen which functions are accessible to tenants.

6. Future Scope

Big players of cloud providers planning to implement the platform based on the conceptual architecture. Furthermore, they are going to provide Software Development Kit (SDK) for SaaS application development that deals with commonality and variability model of aspects of SaaS application using the software product line approach.

7. Conclusions

In this paper, we show a conceptual architecture of a SaaS platform that enables executing of configurable and multitenant SaaS application. The platform allows the configurator application to configure five aspects of SaaS software. In addition, metadata driven architecture composed of Runtime Engine, Metadata Management System, and Metadata DB are applied for providing multitenancy of SaaS application. A collaboration-based security management framework for the cloud computing model is also showed here. The framework introduces an alignment of the NISTFISMA standard to fit with the cloud computing model. Utilization of the existing security automation efforts such as CPE, CWE, CVE and CAPEC to facilitate the cloud services Security Management Process (SMP) and Validation of showed framework by using it to model and secure a multitenant SaaS application with two different tenants have been done by us. A more tenant-driven evolution of a SaaS where a vendor can accommodate changes to a SaaS to meet tenant needs within the reasonable limits.

References

[1] Software as a Service, *http://en.wikipedia.org/wiki/-Software_as_a_service*
[2] Cor-Paul Bezemer and Andy Zaidman, “Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare?”, *Technical Report (TUD-SERG-2010-031)*, Delft University of Technology
[3] Gianpaolo Carraro, “Understanding SaaS Architecture: A Simple SaaS Maturity Model,” *http://msdn.microsoft.com/enca/architecture/aa699384.as px, 2006*
[4] Stephan Ried, “Forrester’s SaaS Maturity Model: Transforming Vendor Strategy while Managing Customer Expectations,” http://*www.forrester.com/Research/-Document/Excerpt/0,7211,46817,00.html*, 2008
[5] Application Service Provider, *http://en.wikipedia.org/wiki/Application_service_provider*
[6] Craig D. Weissman and Steve Bobrowski, “The design of the force.com multitenant internet application development platform”, *In Proc. of the SIGMOD*, 2009, pp. 889-896.
[7] International Data Corporate (IDC), "*Ranking of issues of Cloud Computing model,"* 2010. http://blogs.idc.com/ie/?p=730 Accessed Dec 2010.
[8] M. Almorsy, J. Grundy, I. Mueller, "An analysis of the cloud computing security problem," In the proc. of the 2010 Asia Pacific Cloud Workshop, Colocated with APSEC2010, Australia, 2010.
[9] NIST, "Risk Management Guide for Information Technology Systems," 2002, <http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf>, Accessed in June 2010.
[10] L. Tao, “Shifting paradigms with the application service provider model,” Computer, pp. 32–39, 2001.
[11] D. Jacobs and S. Aulbach, “Ruminations on multi-tenant databases,” BTW Proceedings, 2007.
[12] F. Chong, G. Carraro, and R. Wolter, “Multi-Tenant Data Architecture,” MSDN Library, Microsoft Corporation, 2006.
[13] F. Chong, G. Carraro, and R. Wolter. Multi-Tenant Data Architecture. MSDN Library, Microsoft Corporation, 2006.
[14] C. Guo et al. A Framework for Native Multi-Tenancy Application Development and Management. 9th IEEE Intl. Conf. on E-Commerce Technology and 4th IEEE Intl. Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE), 2007.
[15] S. Aulbach, T. Grust, D. Jacobs, A. Kemper and J. Rittinger. Multi-tenant Databases for Software as a

Service: Schema-Mapping Techniques. In SIGMOD, pp 1195-1206, 2008.
[16] C. Weissman and S. Bobrowski. The Design of the Force.com Multi-Tenant Internet Application Development Platform. In SIGMOD, pp 889-896, 2009.
[17] T. Kwok and A. Mohindra. Resource Calculations with Constraints and Placement of Tenants and Instances for Multi-Tenant SaaS Applications. In International Conference on Service Oriented Computing (ICSOC), 2008.
[18] X. Li, T. Liu, Y. Li and Y. Chen. SPIN: Service Performance Isolation Infrastructure in Multi-Tenancy Environment. In Internatonal Conference on Service-Oriented Computing (ICSOC), pp 649-663, 2008.
[19] K. Pohl, G. Bockle and F. Van Der Linden. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York Inc, 2005.
[20] R. Mietzner, A. Metzger, F. Leymann and K. Pohl. Variability Modeling to Support Customization and Deployment on Multi-Tenant-Aware Software as a Service Applications. In ICSE Workshop on Principles of Engineering Service Oriented Systems (PESOS), 2009.
[21] J. Bayer, S. Gerard, O. Haugen et al. Consolidated Product Line Variability Modeling. Software Product Lines, pp 195-241.
[22] C. Bezemer and A. Zaidman, "Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare?'',In Proceedings of the 4th Internaitonal Joint ERCIM/IWPSE Symposium on Software Evolution (IWPSE-EVOL), 2010
[23] C. Bezemer, A. Zaidman, B. Platzbeecker et al. Enabling Multi-tenancy: An Industrial Experience Report. In ICSM, 2010.
[24] Nitu. Configurability in SaaS (software as a service) Applications, In Proceedings of the 2nd India Software Engineering Conference (ISEC), pp 19-26, 2009.
[25] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger, "Multi-tenant databases for software as a service: schemamapping techniques," in Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 2008, pp. 1195–1206.
[26] C. Guo, W. Sun, Y. Huang, Z. Wang, B. Gao, and B. IBM, "A framework for native multi-tenancy application development and management," in International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007, pp. 551–558.
[27] F. Chong and G. Carraro, "Architecture strategies for catching the long tail," MSDN Library, Microsoft Corporation, 2006.
[28] M. Menzel, R. Warschofsky, et al, "The Service Security Lab: A Model-Driven Platform to Compose and Explore Service Security in the Cloud," 6th World Congress, SERVICES2010, pp.115-122.
[29] M. Menzel and C. Meinel, "SecureSOA Modelling Security Requirements for Service-Oriented Architectures," IEEE International Conference on Services Computing, 2010.
[30] Craig D. Weissman and Steve Bobrowski, "The design of the force.com multitenant internet application development platform", *In Proc. of the SIGMOD*, 2009, pp. 889-896.
[31] R. Mietzner, A. Metzger, F. Leymann and K. Pohl. Variability Modeling to Support Customization and Deployment on Multi-Tenant-Aware Software as a Service Applications. In ICSE Workshop on Principles of Engineering Service Oriented Systems (PESOS), 2009.
[32] R. Mietzner, F. Leymann and M. P. Papazoglou. Defining Composite Configurable SaaS Application Packages Using SCA, Variability Descriptors and Multi-Tenancy Patterns. 3rd Intl. Conference on Internet and Web Applications and Services, pp 156-161, 2008.
[33] http://www.w3.org/Submission/OWL-S
[34] F. Chong and G. Carraro, "Architecture strategies for catching the long tail," MSDN Library, Microsoft Corporation, 2006.

Author Profile

Basant Kumar Gupta received the Bachelors' degree in Information Technology in 2012 from K.N.I.P.S.S. Sultanpur, U.P. He is currently pursuing the Masters' degree in Computer Science and Engineering from Galgotias University, Greater Noida, UP. His area of Interest is Cloud Computing.

Keisam Thoiba Meetei received the Bachelor's degree in Computer Science and Engineering in 2011 from Shiv Shankar Institute of Engineering and Technology (SSIET), Punjab. He is currently pursuing the Master's degree in Computer Science and Engineering from Galgotias University, Greater Noida, UP. His area of Interest is Artificial Intelligence.