ENGINEER - Vol. XXXXVI, No. 03, pp. [21-31], 2013
© The Institution of Engineers, Sri Lanka

A Multi-tenancy Aware Architectural Framework for SaaS Application Development

W. N. T. de Alwis and C. D. Gamage

Abstract: In the era of cloud computing, multi-tenant based Software as a Service (SaaS) applications have been widely identified as the next generation of cloud applications. SaaS allows multiple user organizations to customize an application in a reliable and secure manner. However, this customization is a complex and error prone exercise. In response, researchers and practitioners have come up with SaaS architectures based on frameworks, platforms and modelling approaches to ease the complexity of SaaS application development. However, these methods and tools have not focused on aspect of development methodology being tuned to support long-term maintenance of the SaaS application.

This paper presents an architectural framework for SaaS application development that incorporates long-term maintenance requirements arising from multi-tenancy of the application. It consists of a methodology coupled with a tool chain, which brings multi-tenancy aware features to develop SaaS solutions that meet critical architectural requirements. It also includes a UML 2.0 based Profile named SaaSML for designing of main components, a skeletal framework to position these components and a methodology for benchmark evaluation of key design criteria.

Keywords: Multi-Tenant SaaS web application, SaaS Framework, Software Architecture tools, UML based modelling tool

1. Introduction

Software as a Service (SaaS) business model is impacting the software industry on how customers acquire business functionality and solutions [1]. In this model, application functionality is delivered through an online subscription model. A customer does not take ownership of the software, but instead rents a total solution that is delivered remotely. SaaS has been widely identified as the next generation of cloud applications [1]. This allows multiple users of different organizations (Whom we call as tenants) to use the same software in a reliable and secure manner. Users will be given the liberty to customize the application according to their needs by changing user interfaces, work-flows and business processes [2]. With the SaaS model, customers can reduce up-front support costs; because they no longer need to support multiple platforms and versions [3]. The end user's customized solution is hosted over the Internet giving customers the opportunity access the service on demand basis [4].

For SaaS architectures, multi-tenancy is considered as the main design principle when developing SaaS software [5]. Multi-tenancy can be defined as a single instance of the software running on the vendor's servers, serving multiple tenants. With a multi-tenant architecture, a software application is designed to virtually partition its data and configuration so that each client organization works with a customized virtual application instance. Multi-tenancy may be the most significant paradigm shift that an architect accustomed to designing isolated, single-tenant applications has to make in the era [5]. This requires an architecture that maximize the sharing of resources across tenants, but that is still able to differentiate data belonging to different customers.

In single-tenant software, challenges like configuration and versioning are solved by creating a branch in the development tree and deploying a separate instance. In a multi-tenant software, this is no longer acceptable, which means that features like these must be integrated in the application architecture, which inherently increases the code complexity and therefore makes maintenance more difficult [3]. Wrong architectural choices might

W. N. T. de Alwis, MSc (Moratuwa), BSc (Hons) Information Systems (MMU), attune Lanka (Pvt) Ltd, Colombo, Sri Lanka
Eng. (Dr.) C. D. Gamage, PhD (Monash), MEng (AIT), BSc Eng (Hons) (Moratuwa), MIE(Sri Lanka), CEng, University of Moratuwa, Moratuwa, Sri Lanka

degrade the quality attributes of the multi-tenant SaaS application over its evolution.

2. Related Work

Software Architects from Microsoft and Salesforce.com have said that meta-data driven architectures are the core logic that gets applied within multi-tenant SaaS applications [5], [6], [7]. Within a layered architecture of a SaaS web application, the components within each layer can be conceptually virtualized into tenant compartments and they will be governed by tenant specific and shared meta-data based XML or runtime specific objects. Salesforce.com allows changing application behaviour in the runtime by changing some of these meta-data structures. Changed effects can be felt immediately to tenant users through this.

Force.com provides Software Engineers to build SaaS based applications on Force.com platform to deliver robust, reliable, Internet-scale applications. The design concepts used with the Force.com platform were derived from over ten years of development on Salesforce.com application where millions of tenants make use of Customer Relationship Management (CRM) features [6]. Force.com's foundation is a metadata-driven software architecture that enables multi-tenancy to meet the extreme demands of its large user population. But it is considered as one of the most expensive cloud platforms out there.

A SaaS platform called ApprendaCloud, developed by Apprenda, Inc. allows Independent Software Vendors (ISVs) to transform a Microsoft .Net based single tenant web application to a multi-tenant web application [8]. This platform allows a mechanism to support development and distribution of applications in the SaaS model. It contains an ApprendaCloud SDK that can be integrated with Visual Studio as the IDE to develop SaaS solutions. Tight vendor lock-in from data and application is a major limitation within ApprendaCloud.

Some ISVs have extended their web application frameworks to support SaaS features and they would act in similar nature the Application Service Provider (ASP) model based web applications. This paper explores the requirements and challenges of the native multi-tenancy pattern for SaaS applications [9]. A framework with a set of multi-tenancy based common services is discussed to help Software Engineers to design and develop a high quality native multi-tenant application more efficiently. Due to the essential requirement to guarantee service quality with high share efficiency, this paper presents approaches and principles to support better isolations among tenants in many aspects such as security, performance, availability and administration [9]. Implementation and deployment viewpoints of framework are not described fully.

Development of SaaS architectures can be done through custom SaaS platforms following Service Oriented Architecture (SOA) style [10]. SOA requires loose coupling of services, which requires extensive planning and analysis to be done by the Software Architect to ensure the right SOA components and constructs to be placed to get a high degree of performance, scalability and security [10], [11].

An SOA based framework on a series of model engines, model templates, and model description files can be used to come up with a multi-tenant architecture using modelling as an approach [12]. Here the execution platform and the core engine remains as the core modules of the architecture. Execution platform is mainly for service execution management and core module engine is the core business and controlling layer for managing models. Once an end user organization gets registered, a model description file will be created and saved. Model description files will contain SaaS multi-tenant-aware features on different layers. Mainly XML based model description files are used to describe the menu details per user and Extensible Stylesheet Language Transformations (XSLT) engine is used to generate HTML. Based on versions of this model description files, SaaS features will get active on certain tenants. As the number of tenants increase, the complexity of the model description files will increase and maintenance will become more difficult in the long run.

An approach name vSaaS in iVIC platform, which is considered a virtual computing environment for Hardware as a Service (HaaS) is proposed for SaaS applications [13]. The required software is deployed on iVIC platform to ensure it is accessible through remote streaming approach. The disadvantage of this approach is that only standard software is provided to all users with no customization options. Remote streaming applications tend to take more bandwidth compared to HTTP based web applications. Limitations within virtualize platform may bring in scalability issues for individual software installed. As for this report, remotely installed software virtualization will not be taken into consideration.

Web Services Conversation Language (WSCL) is used to express the views of tenants within SaaS applications [14]. Here the business logic can be described in a conversation manner and the model used is composed of a business model and tenant model. The two parts involved in the conversation have the same interaction; but individualized operations. Through guidelines in the conversation manner, WSCL expresses the procedure for obtaining the relevant tenant model and business model in the runtime. As the tenant needs increase, changes to the models will impact on the WSCL files. This will make maintenance a difficult engineering task.

Summary of the SaaS application development architectural approaches are discussed below in Table 1.

Table 1 - Summary SaaS Architecture development approaches

Approach	Description
SaaS purpose built platforms (Eg: Force.com)	Acts as a powerful web server with a platform API, for SaaS application development. They circumvent SaaS architectural challenges. Governance limitations within platforms open up problems during production stage. Migration from one platform to another will be difficult.
SOA	This approach deals with designing software services in a bottom up approach and ensures that multi-tenant features remain intact. This approach requires extensive planning and analysis to be done by the Software Architect to ensure the right SOA components and constructs to be placed to get high degree of performance, scalability and security. In a short time frame, this is a difficult task and failing to plan, could lead to an architectural collapse.
Extend ASP based web application framework	ISVs extend their frameworks to support SaaS features and they would act in similar nature of the ASP model base web applications. With evolution, such applications collapse as the requirements from different tenant increase during the production stage. The main reason is that the foundation of such architectures was not targeted towards SaaS multi-tenant principle.
Standard software remotely streamed over virtualized environment (iVIC platform)	Provide a SaaS solution for users to access software transparently without considering hardware or software installation and configuration. OS level virtualization and remote display technologies are used. Only standard software is provided to all users with no customization options.
Model driven architecture	WSCL is a XML based modelling notation used to express the views & business logic of tenants in a conversation manner [14]. As the tenant needs increase, changes to the models will impact on the WSCL files. This will make maintenance a difficult engineering task.

3. Approach

As the approach to develop SaaS applications, we propose a methodology using SaaS architectural framework based on tenant-aware services defined within XML files. Third party web application frameworks with specialized capabilities within different layers were integrated as components to solve SaaS challenges. Model Driven Architecture (MDA) based tool was incorporated to allow Software Engineers to generate models and express different architectural viewpoints on functional requirements and design decisions. MDA tool artifacts will be placed within the SaaS application framework that will be used at different layers of the web application.

MDA is a software design approach that provides a set of guidelines for the structuring of specifications, which are expressed as models [15], [16], [17]. MDA allows developing applications and writing specifications, based on a platform-independent model (PIM) of the application or specification's business functionality and behaviour [18]. Figure 1 represents these models as boxes, and their transformations as arrows. MDA tools are used to develop, interpret, compare, align, measure, verify and transform models or meta-models [18], [19]. Unified Modelling Language (UML) has been successfully applied in Software Engineering as a general purpose modelling language [20].

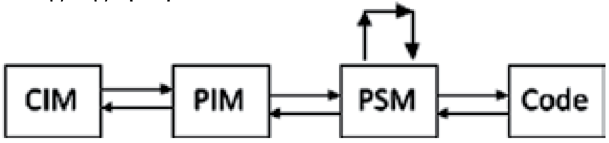

Figure 1 - Models and transformations in MDA

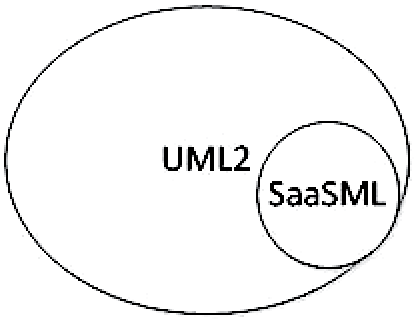

Figure 2 - SaaSML and UML representation Venn diagram

UML is a large and a complex language and it provides mechanisms to allow extensions via stereo-types [20]. UML profile can be defined as an extension of the UML standard language with specific elements. Since standard UML 2.0 doesn't fully cater the exact modelling needs or problems of SaaS multi-tenant architectures, a new UML based profile will be introduced to model SaaS applications called: Software as a Service Modelling Language (SaaSML). Based on SaaSML diagrams, Eclipse EMF tool will be used to generate Java code from UML diagrams where Java classes supporting the SaaS framework will be developed as SaaS application business components and multi-tenant based domain models. Using this approach, we ensure that the SaaS application can be developed and maintained within a short time frame using SaaSML models. Evolution of the SaaS application will be done by changes on the models which will reflect on the application code and XML file.

SaaSML is based on UML and involves modelling tenant blocks instead of modelling classes, thus providing a vocabulary that's suitable for Software Engineering on SaaS based web application. A tenant block encompasses software components within different layers of the web application. As specified in Figure 2, SaaSML reuses a subset of UML2.0. Therefore SaaSML includes ten diagrams based on UML2 Usecase, Class and Object diagrams. SaaSML can be easily understood by the Software Engineering community, due to its direct relation with UML2.

SaaSML makes it possible to generate a UML2 based specifications for Software Engineering teams, dealing with the realization of multi-tenant systems with cloud based hardware and software. Knowledge is thereby captured through models stored in a single repository, enhancing communication within the Software Engineering team. In the long term, tenant blocks can be reused as their specifications and models enable suitability assessment for tenant based customization projects.

SaaSML structure diagrams are discussed: "Core Usecase diagram" is a Usecase diagram to model core solution of the SaaS application. "Tenant Org Usecase diagram" is a Usecase diagram to model tenant specific customization. "Core Domain diagram" is a class diagram to model core solution of the SaaS application. "Tenant Org Domain diagram" is a class diagram to model different tenant specific customizations. The "User Hierarchy diagram" is an Object diagram considering authorization User Rights and standard user hierarchies within a tenant organization. The "Tenant User Provisioning diagram" is a component diagram which represents the User repositories (Eg: Ms Exchange/Gmail) which require integration as services. The "Billing Plans and Metering diagram" is a class diagram which represents the SaaS services, billing plans and metering options. The "SaaS Deployment diagram" is a deployment diagram that identifies the required hardware and software considering SaaS application deployment. "SaaS Global Settings" is an Object diagram which represents the global settings that needs to in-cooperated into the SaaS application. The "SaaS Governance Rules diagram" is a based Class diagram which represents rules which needs to be taken governance aspects to derive SLAs against Billing Plans.

4. Multi-Tenant SaaS Architectural framework Overview

A general form of SaaS architectural goals and constraints are discussed in the Table 2.

Table 2 - SaaS architectural goals and constraints

Requirement	Architectural Goal/Decision
Modifiability	Change the presentation layer, business process layer and service layer based on tenant [21].
Database Configuration	Store individual tenant data within a shared DB schema or different DB schemas [22].
Performance	SLA based tenant fair usage policies for SaaS Services [5].
Security	User authentication and authorization, data security and Web Service Service security [5].
Usability	Improve user experience of the software [21]
Fault Tolerance	Ensure proper monitoring mechanisms are in place to recover from infrastructure failures [5].
Scalability	Automatically scale infrastructure based on service usage [5]

In the methodology of the multi-tenant SaaS architectural framework, SaaS application will be designed by formulating architecture views: such as business usecase view, logical view, process view, deployment view, implementation view and architectural strategies to overcome SaaS challenges. Below we have explained these views and strategies in detail.

4.1. Business Usecase View

SaaS applications allow ISVs to run global businesses on the cloud. At the start of design phase, Software Engineers need to plan the following parameters for a global live run: regions or countries, languages, currencies and regulations within regions. Business Analysts need to perform a thorough analysis on the given business domain considering these global factors. Based on the administrative isolation levels, SaaS applications will be mainly divided into two software components: [9].

- Admin Portal - SaaS vendor, vendor's partners (such as resellers & distributor) and tenant organizations' administrators would login to this portion of the SaaS application for administrative purposes. This application will cater tenant user organization management, billing plan management and usage reports (operational, financial and quality attributes). It will serve as a governance module to manage all aspects of the SaaS application. Common features supported: management of users, billing plans, payment mechanisms, selection of application services & selection of global parameters (currencies, languages, time zones, number display format &, date formats).
- User Portal - End users (tenant users) will be served through this portion of the application. Based on settings configured within Admin portal, the software services offered by the SaaS solution will be used by end users of the tenant. Most of the discussions on SaaS functional and non-functional requirements are discussed on this area of the application. End user must feel the application is personalized to cater organization specific business needs.

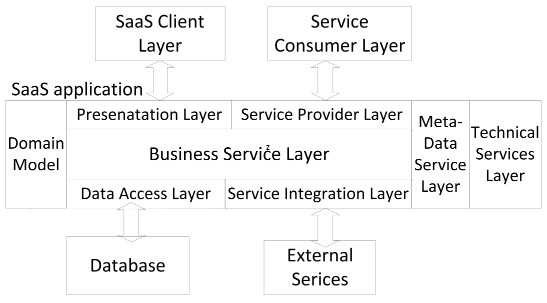

Figure 3 - SaaS Logical Architecture

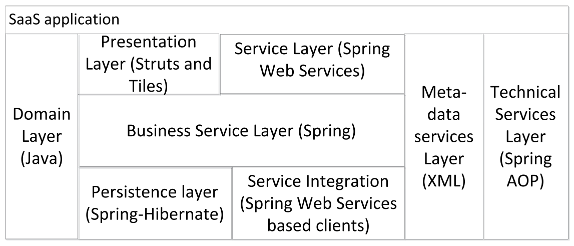
SaaS application
Presentation Layer (Struts and Tiles)
Service Layer (Spring Web Services)
Domain Layer (Java)
Business Service Layer (Spring)
Meta-data services Layer (XML)
Technical Services Layer (Spring AOP)
Persistence layer (Spring-Hibernate)
Service Integration (Spring Web Services based clients)

Figure 4 - Detailed Implementation Architecture

4.2. Logical View

We have proposed a SaaS reference architecture in Figure 3 and we have used this for SaaS web application proto-type development. The following is a description of each of its layers.

- SaaS Client layer - browser, plugin or

mashup
- Presentation layer - Presentation component such as UI components, work-flow controller components, Validation and Model objects
- Service Provider layer - External/Internal services layers retrieve information within the application
- Business Service layer - Business Process layer containing domain specific business logic, rules and constraints
- Data Access layer - Data persistent and retrieval layer to backend Databases, local file systems, cloud storage (Amazon S3 or Google Storage)
- Domain Model will represent business domain objects which will be common for all layers above. To support SaaS Architectural Cross Cutting concerns:
- Tenant-aware Services Layer - Tenant specific customization of the presentation, business process, data access, EAI layer and Service Provider layer.
- Technical Services Layer - Which includes security, logging, performance and SLA based governance restrictions.

The Figure 4 represents the third party web application frameworks used for the SaaS application framework. Here we used JSPs, Apache Struts, Tiles, Custom Tag libraries, Spring and Hibernate. All these layers are connected through Spring framework which drives SaaS multi-tenant based injection. Spring uses Dependency Injection and Aspect oriented programming (AOP) concepts will be used bring tenant based dynamic presentation layer, work-flow, business rules, service and data access layer. We also used custom annotations evaluate access control of business services used within different layers using Spring Aspect Oriented Programming. Here the domain object identified during domain analysis and SaaS framework specific controller classes will be called as Plain Old Java Objects (POJO)s.

4.3 SaaS framework strategy to solve SaaS Challenges

Table 3 explains the framework strategies adopted to tackle SaaS challenges

Table 3 - SaaS framework strategies

SaaS Challenge	SaaS Framework approach
Multi-tenant Presentation layer customization	XML template definitions used to load UI widgets, styles sheets and page components defined in design time.
Multi-Tenant Work-flow layer	XML definition used to differentiate workflows defined in design time.
Multi-Tenant based Business logic validation	XML definitions used to load tenant specific business components defined in design time.
DB configuration for Multi-tenant application	Use XML definition apply Object Relational Mapping to support shared DB schema or tenant specific DB schema.
Multi-Tenant based Service Integration and Service Exposure	Generate web service client code and use XML definitions to load tenant specific business components defined in design time.
Tenant specific Billing Subscription and Metering	Using AOP based cross cutting code to monitor usage service metering options and generate daily usage bills using crone jobs.
Multi-tenant based soft-ware build environment	Use Maven - build management tool to compile sources, run all test cases and deploy.
Performance restriction based on SLA	AOP based cross cutting code to measure service usage and restrict based on SLA.
Security to between Tenants	Use a login module based on central security services for authentication. Use access control module for authorization. Apply AOP cross cutting code to control access rights in runtime.

4.4 Process View

The process view describes the how SaaS meta-data services which get triggered within the application. Here a web request-thread based place holder (ThreadLocal) was used to keep tenant configuration data. A login module was developed to authenticate tenant users and check if the tenant billing subscription is valid. If the authentication is approved, a web session will be created and tenant configuration data will be stored within it. Later on subsequent web requests, the tenant configuration data will be stored within ThreadLocal variable. During rest of the thread execution, objects related tenant specific needs will get called within different layers. This process work-flow allows tenant based dynamic behaviour to be triggered at runtime.

4.5 Deployment View

The deployment view will resemble the hardware and web server software layers that the SaaS application will be deployed on. It needs to support scalability aspects defined within SaaS Architectural goals. It should also allow integration between different services and access services within the SaaS application to outsiders on a secure and robust manner. Figure 5 shows the main deployment components of the application. Here the SaaS client application will be a browser, mashup, plugin or desktop application that will make use of the SaaS application. Currently we have chosen Amazon EC2 based Linux box with Apache Tomcat6 as the deployment environment for the proto-type SaaS application.

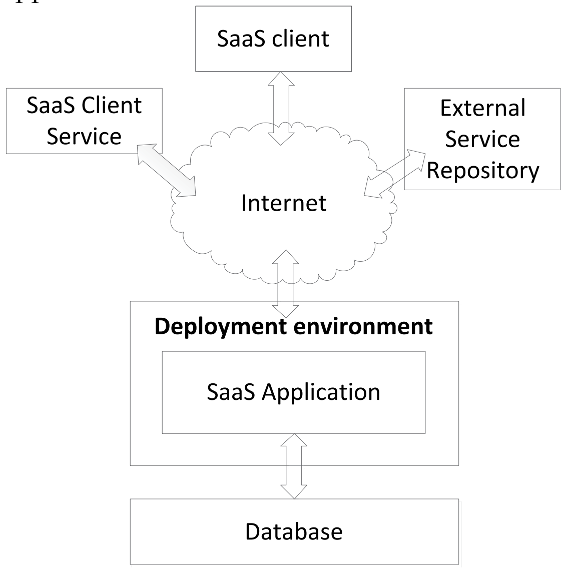

Figure 5 - Deployment Architecture of SaaS web application (general view)

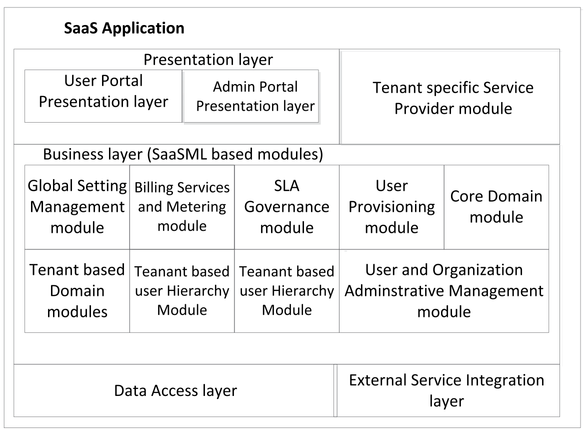
SaaS Application
Presentation layer
User Portal Presentation layer
Admin Portal Presentation layer
Tenant specific Service Provider module
Business layer (SaaSML based modules)
Global Setting Management module
Billing Services and Metering module
SLA Governance module
User Provisioning module
Core Domain module
Tenant based Domain modules
Teanant based user Hierarchy Module
Teanant based user Hierarchy Module
User and Organization Adminstrative Management module
Data Access layer
External Service Integration layer

Figure 6 - SaaSML based Component Diagram

4.6 Implementation View

Java components based on SaaSML specification which acts as modules within the Domain and Business layers of the SaaS framework. In the Figure 6 component diagram of the SaaS Architecture is displayed. These components will be placed within the Business layer of the application. In the following list we highlight individual component module, their design rationale and dependency with each other modules.

- Core module - The core application business domain will be represented here. Mainly the Core Usecase Diagram and Core Domain Diagram would be used to define the Core module.
- Tenant module - Tenant specific customer extension to the core module will be defined within this module. Each customer specific domain object will be kept within a separate package. This module will depend on the Core module components. Tenant Org Usecase Diagram and Tenant Org Domain Diagram
- System User Rights Module - Define all user rights of the SaaS application as Enumerations.
- Tenant User Hierarchy Module - Define tenant specific user roles and mapping between the user rights. These roles could be used within Core module or Tenant module POJO methods' custom annotation to restrict business method execution based on tenant User-Role or System specific User Right.
- The Billing and Metering Diagram -This module will contain the SaaS Services and its Metering Items. A Billing Plan will have any number of services that could be applied on tenant organizations.
- SaaS Global Settings - This module will represent the possible countries, time zones, currencies, date formatting pictures as enumerations that will be used as utility services
- The SaaS Governance module -represents the possible metering items that require monitoring and restriction based on tenant SLAs.

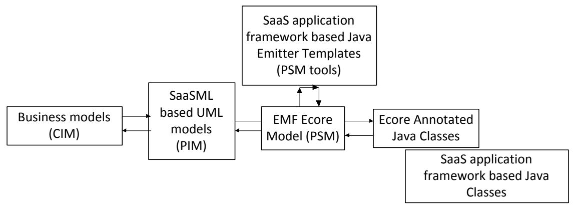
SaaS application framework based Java Emitter Templates (PSM tools)
Business models (CIM)
SaaSML based UML models (PIM)
EMF Ecore Model (PSM)
Ecore Annotated Java Classes
SaaS application framework based Java Classes

Figure 7- SaaSML based MDA approach to generate sources

4.6.1 MDA approach to generate SaaSML based components

Using Eclipse EMF framework, the UML2.0 based SaaSML diagrams will be used to derive the SaaS application domain model and business components. The Java code that is generated through this approach will be based on Eclipse ECore API. This code will mainly consist of Java Interfaces. The implementation of these interfaces can be modified with the use of a Java Emitter Template (JET). Depending on the SaaS application deployment environment, the usage of underline infrastructure services and third part libraries will vary. The generation of Java classes which are dependent on the infrastructure services and third party libraries, can be done using Java Emitter Templates integrated with the EMF framework. Here ECore models are converted to technology specific Java implementations. This will help Software Engineers to extend the application to multiple platforms even with a common PIM model based on SaaSML.

Usage of SaaSML based MDA approach is shown in Figure 7. Using SaaSML MDA based approach; code generated through SaaSML diagrams can be re-used on other SaaS applications as libraries. Extension to initial Java classes can be enforced through Java Emitter Templates which are knowledgeable on re-usable components.

4.6.2 How to couple external business components to the SaaS application?

There could be cases where a proven business logic written for specific business domains, which needs to be coupled within the SaaS application without developing modules from scratch. Let's assume a scenario where third party libraries which handle CRM based billing needs to be coupled within the SaaS application. In such a case, the SaaSML -Billing and Metering diagram needs to be modelled considering these external components or third party libraries. A PSM based JET and Ecore Models need to be used to generate Java source compatible for the billing API. With this approach SaaS application framework based billing and metering sources (that can be coupled with the external billing API) can be generated.

4.6.3 How to reuse SaaSML components of one SaaS application in another?

Based on SaaSML models, platform specific Java sources can be generated using MDA approach. These sources can be used within the SaaS application to represent SaaS functionality. Assume a new requirement comes to re-use the SaaSML Java sources within a different SaaS application. In such a case, the previously generated SaaSML models can be re-used again, but the platform specific transformation can be avoided by integrating previously generated components. JET templates and ECore models can be used to generate sources which will make use of re-usable component as an external business component. Re-usable SaaSML components can be placed within different Maven Projects that can be mapped as a dependency with other SaaS application Maven projects. With this approach, SaaSML components can be re-used within other SaaS applications.

5. Evaluation

Evaluation of the SaaS application was done by comparing it with Force.com & ApprendaCloud based SaaS applications. The aim of this whole exercise was to compare the implemented SaaS features against a similar product to understand its ranking (comparatively). As for the SaaS product evaluation, we used an AHP technique for prioritizing the product features and expert judgement based scoring of the products [24]. The ranked sum of weighted scores in descending order gives the ranking of the products as shown in Table 4. The SaaS prototype application carried a ranking of 0.378 (low), ApprendaCloud application carried a ranking of 0.477 (medium). and Force.com application carried a ranking of 0.622 (high). This shows that even a quickly developed prototype application using the proposed methodology can be competitive with established applications from globally recognized SaaS providers.

Table 4 - Ranking of products

Factor	Attributes	Score of Force .com app	Score of ApprendaCloud app	Score of SaaS Proto-type
Functionality	Tenant account man-agement	0.06	0.05	0.06
	Metering and Billing features	0.048	0.04	0.032
	SaaS end user features	0.096	0.04	0.064
	Provisioning mecha-nisms	0.028	0.01	0.012
Architecture	Modifiability	0.126	0	0.054
	Security	0.105	0.09	0.045
	Performance	0.072	0.05	0.048
	Reliability	0.036	0.02	0.024
	Scalability	0.021	0.015	0.009
	Easiness to develop	0.018	0.15	0.012
	Governance restrictions	0.012	0.012	0.018
Total		0.622	0.477	0.378

A performance study on the prototype application was carried out using JMeter based stress testing tool with sample data sets which represented two Tenant users with different customizations. Tool was used to record workflows of the two types of tenant users and they were simulated in an iterative manner within JMeter Test Plan to generate maximum of 100 concurrent transactions per second on the SaaS web application. Acceptable response time was taken as 5 seconds and throughput as 20 at minimum. By analyzing concurrent user threads (refer Figure 8), response time (refer Figure 9) and throughput (refer Figure 10) over a predefined time duration, we didn't see any performance bottlenecks and felt the prototype application performance is acceptable. Further testing is required to identify suitable deployment environments considering high performance and scalability aspects. Using ATAM we have evaluated the SaaS prototype application. Performance is considered a key sensitivity point which will get impacted by the dynamic configuration options adopted within

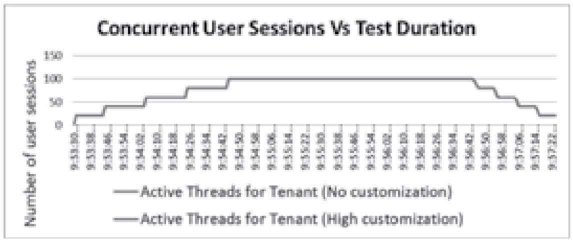

Figure 8 - Graph that represents concurrent user sessions (threads) Vs Test duration

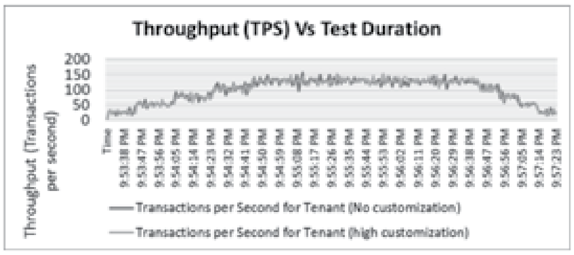

Figure 9 - Graph that represents Response time Vs Test duration

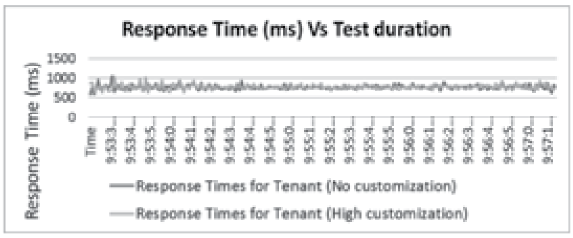

Figure 10 - Graph that represents Throughput Vs Test duration

6. Future Work

Further improvements on the performance side of framework are required based on the evaluation. Extracting modular artifacts from the SaaS framework and porting them to support different PaaS environments would be a challenging research. SaaS framework modelling tools for migration between different data modelling options can also be considered as a future research topic.

7. Conclusion

In this Journal paper, authors have proposed a SaaS architectural framework that allows MDA based tools to be used in developing SaaS solutions. SaaSML based UML profile was introduced to capture SaaS functional requirements and derive UML2.0 based diagrams to generate modular software components. A prototype SaaS application was developed as a proof of concept. It was evaluated against Force.com & ApprendaCloud based SaaS applications using AHP technique. Results showed that even a quickly developed prototype application, using the proposed established applications from globally recognized SaaS providers. The proposed architectural framework & methodology can be used for the development of SaaS applications to solve maintenance issues faced by ISVs.

References

1. Armbrust M., Fox A., Griffith R., Joseph A. D., Katz R., Konwinski A., Lee G., Patterson D., Rabkin A., Stoica I., and Zaharia M., "A View of Cloud Computing," Commun. ACM, vol. 53, pp. 50–58, April 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721672

2. Lewis G., \Basics about Cloud Computing," Software Engineering Institute, http://www.sei.cmu.edu/library/assets/whitepapers/Cloudcomputingbasics.pdf [Accessed: Feb 17, 2011], Tech. Rep.

3. Bezemer C. and Zaidman A., "Multi-tenant SaaS Applications: MOD International Conference on Management of Data, ser. maintenance dream or nightmare?" in Proceedings of the Joint SIGMOD '08. http://doi.acm.org/10.1145/1376616.1376736, ERCIM Workshop on Software Evolution (EVOL) and Interna-ACM, 2008, pp. 1195–1206.

4. "Cloud Basics Software as a Service," Microsoft Corperation., http://www.microsoft.com/industry/government/guides/cloud computing/-SaaS.aspx [Accessed: Oct 20, 2011], Tech. Rep.

5. Chong F. and Carraro G., "Architecture Strategies for Catching the Long Tail," Microsoft Corporation, http://msdn.microsoft.com/en-us/library/aa479069.aspx [Accessed: Oct 23, 2010], Tech. Rep.

6. Weissman C. D. and Bobrowski S., "The Design of the force.com Multitenant Internet Application Development Platform," in Proceedings of the 35th SIGMOD International Conference on Management of data, ser. SIGMOD '09. http://doi.acm.org/10.1145/1559845.1559942 :ACM, 2009, pp. 889–896.

7. "Convert your Web Application to a Multi-tenant SaaS Solution," IBM Developer works, http://www.ibm.com/developerworks/cloud/library/cl-multitenantsaas/?ca=drs- , CT316, Dec.

8. Han J. and Kim G., "Integration Technology of Literature Contents Based on SaaS," in 2011 International Conference on Information Science and Applications (ICISA). IEEE, Apr. 2011, pp. 1–5.

9. Guo C., Sun W., Huang Y., Wang Z., and Gao B., "A Framework for Native Multi-tenancy Application Development and Management," 2007.

10. Mietzner R., Unger T., Titze R., and Leymann F., "Combining Different Multi-tenancy Patterns in Service-oriented Applications," in Enterprise Distributed Object Computing Conference, 2009. EDOC'09. IEEE International. IEEE, 2009, pp. 131– 140.

11. Sun W., Zhang K., Chen S., Zhang X., and Liang H., "Software as a Service: An Integration Perspective," Service-Oriented Computing–ICSOC 2007, pp. 558–569, 2010.

12. Jiang X., Zhang Y., and Liu S., "A Well-Designed SaaS Application Platform based on Model-driven Approach," in Grid and Cloud Computing, International Conference on, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 276–281.

13. Zhong L., Wo T., Li J., and Li B., "A Virtualization-Based SaaS Enabling Architecture for Cloud Computing," in 2010 Sixth International Conference on Autonomic and Autonomous Systems (ICAS). IEEE, Mar. 2010, pp. 144–149.

14. Ying L., Bin Z., Guoqi L., Deshuai W., and Yichuan Z., "Personalized Modelling for SaaS Based on Extended WSCL," in 2010 IEEE Asia-Pacific Services Computing Conference. IEEE, 2010, pp. 355–362.

15. Brown A., "An Introduction to Model Driven Architecture," IBM developer works, http://www.ibm.com/developerworks/rational/library/3100.html [Accessed: Jan 11, 2011], Tech. Rep.

16. Schmidt D., "Model-Driven Engineering," IEEE computer, vol. 39, no. 2, pp. 25–31, 2006.

17. Seidewitz E., "What models mean?" Software, IEEE, vol. 20, no. 5, pp. 26–32, 2003.

18. "Model Driven Architecture (mda) faq..." Object Management Group, Inc., http://www.omg.org/mda/faqmda.htm [Accessed: Jan 11, 2011], Frequently asked questions report.

19. Sendall S. and Kozaczynski W., "Model Transformation: The Heart and Soul of model-Driven Software Development," Software, IEEE, vol. 20, no. 5, pp. 42–45, 2003.

20. Schattkowsky T., "Uml 2.0-overview and Perspectives in soc Design," in Proceedings of the Conference on Design, Automation and Test in Europe-Volume 2. IEEE Computer Society, 2005, pp. 832–833.

21. Weiping L., "An Analysis of New Features for Workflow System in the Saas Software," in Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human, ser. ICIS '09. http://doi.acm.org/10.1145/1655925.1655946 : ACM, 2009, pp. 110–114.

22. Aulbach S., Grust T., Jacobs D., Kemper A., and Rittinger J., http://www.sei.cmu.edu/library/assets/whitepapers/Cloudcomputingbasics.pdf, "Multi-Tenant Databases for Software as a Service: Schema-[Accessed: Feb 17, 2011], Tech. Rep. mapping techniques," in Proceedings of the 2008 ACM SIG

23. Budinsky F., Brodsky S., and Merks E., Eclipse Modelling Frame-Work. Addison-Wesley Boston;, 2003.

24. Godse M. and Mulik S., "An approach for Selecting Software-as-a-Service (SaaS) Product," in IEEE International Conference on Cloud Computing, 2009. CLOUD '09. IEEE, Sep. 2009, pp. 155–158.