Concern Based SaaS Application Architectural Design

Aldo Suwandi, Inggriani Liem, Saiful Akbar

Concern Based SaaS Application Architectural Design

Aldo Suwandi, Inggriani Liem, Saiful Akbar[1)]

School of Electrical Engineering and Informatics, Institut Teknologi Bandung
West Java, Indonesia
aldosuwandi@gmail.com,inge@informatika.org,saiful@informatika.org

Abstract. With SaaS application, tenant can focus on application utilization while Independent Software Vendor (ISV) is responsible for application deployment, installation, operation and maintenance. Using Aspect Oriented Software Development (AOSD), we propose eight concerns, i.e. configurability, discriminator, measurement, monitoring, tenant management, billing management, performance management, and application management. Those concerns are integrated into a SaaS system architectural design, to enhance SaaS operational flexibility and maintainability. As proof meterof concept, we developed a SaaS operational environment using Spring and AOP. Two Java applications have been integrated to this environment after tailoring. We have tested the modules, classes and services and then the applications, to demonstrate that the platform is able to run a set of web applications as a SaaS. Using this system, ISV can modify an existing Java application easily to be a part of SaaS and measure resource usage and monitor SaaS operation by a dashboard.

Keywords— SaaS, Software as a Service, Concern, Aspect Oriented Software Development

1 Introduction

Traditional software users are typically overburdened with operational work such as maintenance and software deployment. SaaS model is raised to overcome these problems. Many software companies start adopting this model. Instead of buying, more and more individual users and companies rent software. Software maintenance and deployment that were previously handled by users now is handled by ISV.

Based on [1], [2], [3], [4], [5], [6], SaaS is a hosted software and delivered over a network on a subscription basis. SaaS operational systems could be categorized into platform centric or service provider centric [7]. ISV and tenant are two main factors of SaaS operational systems. The maturity of SaaS application is determined by tenancy, configurability, and scalability [8].
Seven pricing models for SaaS application has been studied by Kalisa [9]. Billing should be built as part of SaaS application [14]. However, this functionality must be flexible because each tenant may have a preference of its his own billing scheme.

Enhancement of SaaS application design is required. Aspect oriented software development is one of the application development model to improve the structuring, reusability, and reduce the complexity of the model compared with object oriented model [10].
Our research has studied "concerns" that should be considered to develop SaaS application based on AOSD. Separation of concern is a fundamental principle to facilitate software deployment [11]. Concern is interest which pertains to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders [12]. Core concern represents functional requirement. Cross-cutting concern represents non-functional requirement. Object oriented model is appropriate for core concern development, whereas AOSD is better for developing cross-cutting concern. Cross-cutting concern could be implemented with aspect oriented programming (AOP). Implementation of AOP is based on four design elements, namely aspect, join point, point cut and advice [4]. Aspect is a class consisting of one or a number of advices. Join point is a point on program that will be intercepted by the aspect class. Point cut is a boolean expression that determines join point. Advice is a method containing a logical process to be executed when the class aspect intercepts.
In this paper, we describe our approach in using AOSD for identifying general SaaS requirements, proposing a SaaS system architecture that contains all of the proposed concerns, and the proof of concept for the proposed architecture.

2 Related Work

Chate [13] proposed a way to build multi-tenant SaaS application either from scratch or from existing application. Corrent's SaaS-Factory is used to convert legacy database into a new multi-tenant database. Tang, et al [14] proposed SaaS Platform Management Framework as modules that can be considered to manage SaaS application platform. The platform of this framework consists of two levels, namely business and operations. Business contains related modules with billing management, whilst operation contains related modules with operational system such as metering, configuration, monitoring, etc. Those modules should be considered when implementing SaaS management platform.
Different from Chate[13] and from Tang et all[14], in this paper we focused on applying AOSD as fundamental technique to build a SaaS framework for the benefit of ISV and tenant. We also provided dashboard in the environment that we consider important for monitoring the operation of SaaS. We believe that aspect oriented software development (AOSD) can contribute as a fundamental approach for operating applications in a SaaS environment.

3 Separation of Concern

In AOSD, separation of concern is the first phase of design. Concern affecting SaaS application could be identified from SaaS application requirement.

Before adopting AOSD, we tried to describe the different characteristics of SaaS and non SaaS applications using McCall's Software Quality Factor. However, our observation proved that McCall Software Quality factors don't relate closely to software operational factor that is tightly coupled to SaaS operational characteristics. Most of the quality factors which are used to assess non SaaS application also can be used for SaaS application. The only difference is on maintainability factor. A non SaaS application is maintained by user, while a SaaS application is maintained by ISV. Thus, we decide using another approach to define general requirement of SaaS application. We chose to make further study based on AOSD and AOP.

Aspect-oriented software development (AOSD) is software development technology that seeks for new modularizations of software systems in order to isolate secondary or supporting functions from the main program's business logic. AOSD allows multiple concerns to be expressed separately and automatically unified into working systems. AOSD is supported by AOPL (Aspect-oriented programming languages). By combining AOSD and AOPL, designer can modularize concerns that cannot be modularized using traditional procedural or object-oriented (OO) methods. Examples of crosscutting concerns include tracing, logging, transaction, caching and resource pooling. These examples are relevant to SaaS application.

Two main actors of SaaS application are tenant and ISV. Tenant's main purpose is to use application functionalities as a service. Multi-tenant SaaS application should be configurable to fulfill tenant's requirement varieties. Furthermore tenant should have an isolated data, that can only be accessed or modified by authorized users.

To make it easier for the tenant, ISV manages application deployment and maintenance for them. ISV could have one or many applications to be operated with SaaS model. ISV requires design enhancement on SaaS application to reduce maintenance and development cost to operate multiple SaaS application. It is also important for ISV to know the performance of each SaaS operated application. Furthermore ISV also needs information of resources usage and transaction by each tenant. In order for an application to be considered as having high level maturity, not only it has to have billing functionality, but it also has to be configurable and scalable, and have multitenant [9].

In this research we focused on the design of configurable multi-tenant SaaS application. We specify six main generic-requirements of SaaS application:

a. Tenant should have appropriate functional requirement needed.
b. Tenant should have their own copy of data.
c. ISV should be provided with a centralized application management enabling it to operate many SaaS applications.
d. ISV should be informed of each application's performance status.
e. ISV should be informed of resources which are used by tenant in each application.

f. ISV should be provided with transaction management in order to manage billing transaction of each application.

These six main user requirements are shown in the diagram below (see Figure 1). User requirements derive one or more concerns. We proposed eight concerns depicted on Figure 1. Those concerns are configurability, discriminator, measurement monitoring, tenant management, billing management, performance management, and application management. We derived those concerns from ISV and tenant requirement, and crosscutting elements of SaaS application. Concerns are grouped into core concern and cross-cutting concern, referring to AOSD terminology.

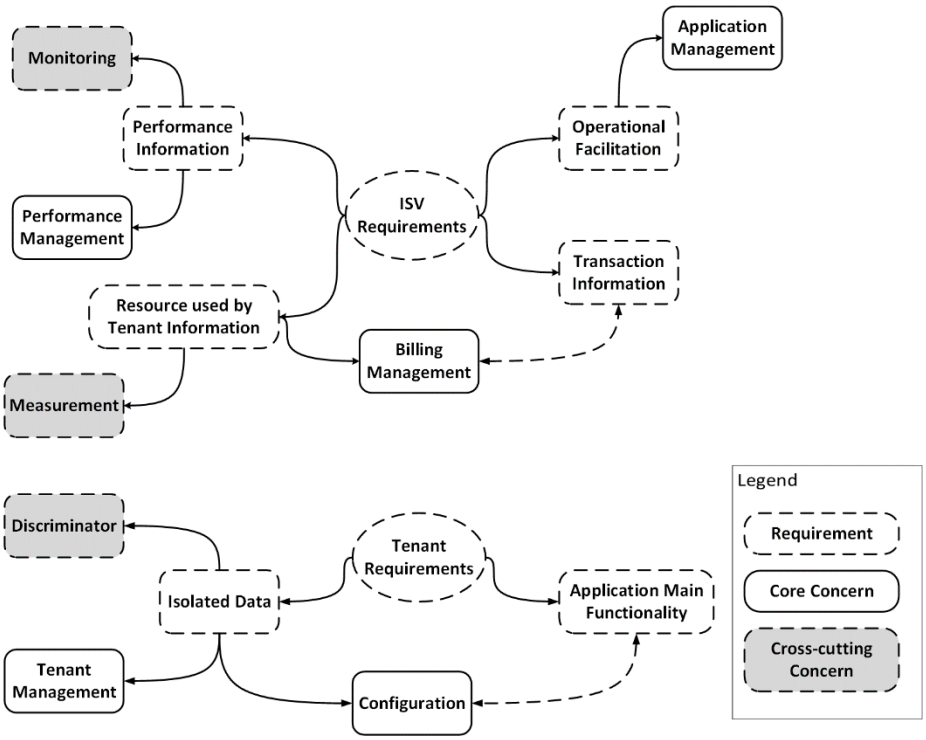

Fig. 1. Concern Derivation Process and Result

4 Proposed SaaS System Architecture

By applying AOSD, the architectural design of a SaaS system contains eight modules. Each module is the implementation of corresponding proposed concern. The architecture is shown on Figure 2.

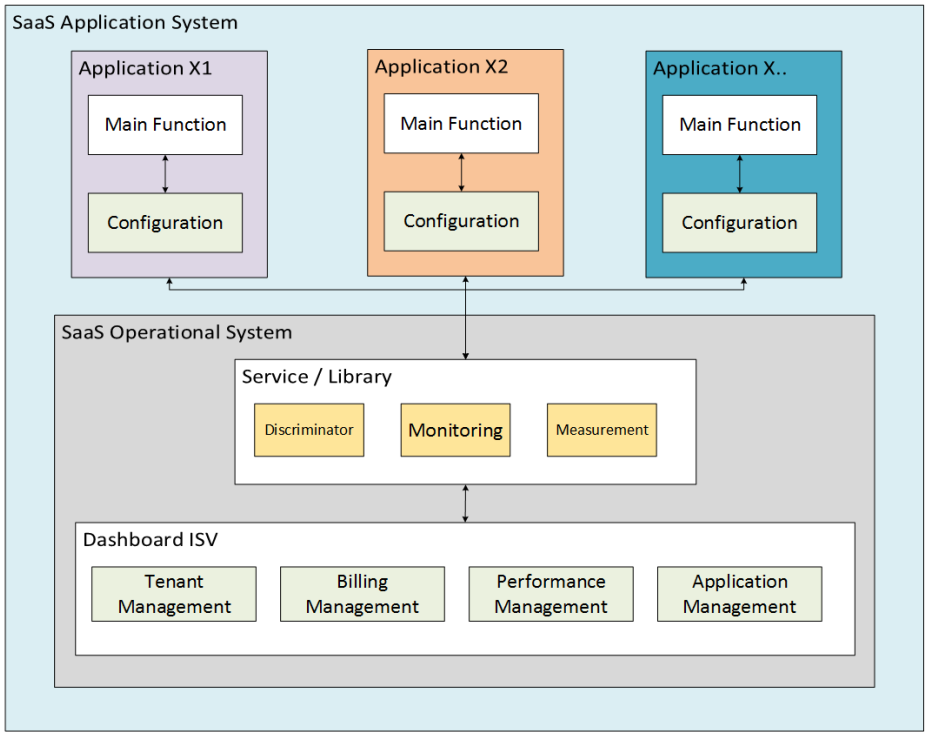

Fig. 2. Proposed SaaS System Architecture

Concerns are implemented into module, service, or library. Those concerns are integrated into SaaS system architecture that consists of SaaS operational system and set of applications.

Core concerns are implemented as modules. Concern configuration is attached into application main functionality, to support variation of configuration as proposed by W.-T.Tsai [15] i.e. GUI, service, workflow, data, and QoS. Other core concerns such as tenant management, billing management, performance management, and application management will be implemented as modules on SaaS operational system's dashboard. This dashboard provides centralized SaaS operational system information from set of applications (application x1, x2, ..., xn). It will facilitate ISV to monitor and to control its SaaS operated applications.

Discriminator, monitoring, and measurement are considered as cross-cutting concern, used by all SaaS applications. These concerns are implemented into web service or library. The Service helps application to communicate with SaaS operational system. The Library contains of aspect classes, representing the interoperability of application with SaaS operational system. These classes reduces the complexity of application operational model of SaaS application. Aspect classes in the library will be implemented with AOP.

5 Implementation

To test our proposed SaaS system architecture, a case study has been conducted as a proof of concept. A SaaS system called **Olympus** shown on Figure 3. has been developed as SaaS system implementation of the designed architecture in Figure 2.

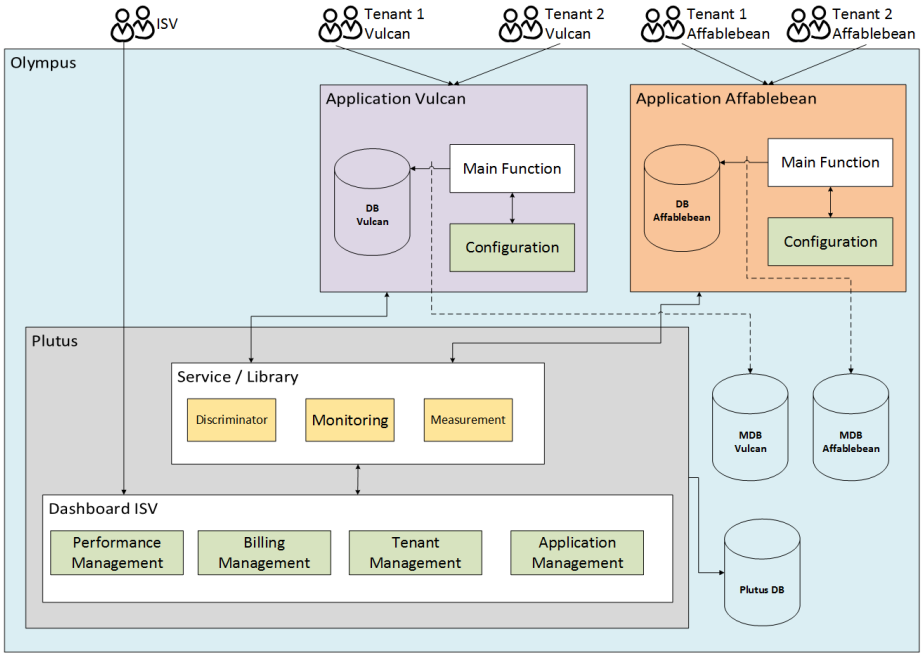

Figure 3. Example of Implementation

We have developed and implemented SaaS application (**Vulcan**) and ISV dashboard (**Plutus**). To prove that it is easy to integrate an existing application to our proposed SaaS application, we use **Affablebean**, an e-commerce application [16]. We have implemented **Vulcan** and **Affablebean** in different mode of SaaS operation and billing.

Vulcan and **Plutus** have been developed using **Spring Framework** [17], **Spring AOP** [17], **Jerseyclient** [18] and **Jamon library** [19]. **Spring Framework** and **Spring AOP** are chosen as the application framework because of their support to AOP. **Jersey client library** is used as a client for REST web service. **Jamon** is used as a library application monitoring.

ISV configures SaaS operational system such as billing model and application info through **Plutus** dashboard. AOP facilitates billing model on SaaS operated application without modification of existing code. AOP also increases reusability. For example if an application has a number of resources to be measured, programmer's task is to

implement annotation with boolean expression as join point where the resource is used.

ISV is provided with performance information such as application's method response time and error report on performance management. This information will help ISV tracing error on application and to decide maintenance activity. Performance management works by monitoring service and monitoring aspect. Techniques that have been implemented for billing can be used to monitor SaaS application.

We also integrate applications one by one, to simulate how applications in the architecture grow. Those applications are integrated with **Plutus** via combination of provided service and library to be operated as SaaS model. ISV configures SaaS operational system such as billing model and application info through **Plutus** dashboard.

Vulcan is a problem set databank application that facilitates tenant to manage, review, publish and create various types of questions such as multiple choice, true/false, short answer, etc. Test is extracted from data bank based on a predefined test blueprint. **Vulcan** provides web services to external system that is authorized to use the test. The implementation of configurable concern permits each tenant to define his own theme, question type, question statistics management, and question additional attributes. Parameterization technique is used to implement theme configuration. User preferred theme is saved in **Vulcan** database and will be used for differentiating one tenant from others. Metadata file configuration is used to implement question type modification and question statistic management. Tenant can configure the provided metadata in XML to modify or to create a new question type and statistic model. XML extension technique [21] is used to implement flexible attribute of question. Inside **Plutus**, **Vulcan** turned into multi-tenant SaaS application used by many academic institutions. ISV configures billing model of each **Vulcan** tenant. The number of questions and time usage of **Vulcan** can be limited by a specific implementation of measurement modules.

Affablebean is an existing web application that builds by NetBeans team for JEE application development tutorial. The main functionality of this application is being a point of sales between **Affablebean** store and customer. Before being integrated to **Plutus** environment, **Affablebean** is modified to be a multi-tenant SaaS application. It is assumed that there are a number of **Affablebean** stores, where each store has different kinds of product and customer. Instead of usage limitation, the resource model in **Affablebean** is implemented by a period of time

Before integrating a new application to the environment, we have to configure and to tailor it to a multitenant SaaS application, then to register the application to the environment. The following steps illustrate the process:

a. Change the old database connection to new multi-tenant database. This connection will be provided when ISV registers their application to **Plutus**. We adopted shared database separated schema model for supporting multi-tenant database ar-

chitecture. ISV should save the application database schema. The saved database schema will be used to automatically generate new parameterized schema (schema with all table name extended with tenant identifier) for each tenant when ISV registers new tenant

b. Extend the **Plutus** library to support the new database. All supporting libraries needed by new application should also be included.
c. Include service identifier method, provided by **Plutus** library on SaaS application login method. It permits application to be recognized by the system. This method is used to identify tenant and billing method when they login to SaaS application.
d. Define annotation in measurement and monitoring class using **Plutus** library, to specify which method or class needed to be metered or monitored.

6 Testing

Our SaaS system has been tested in two folds: testing the environment and dashboard, and then testing the integration of a new operated SaaS Application as shown in Figure 4. Each fold consists of unit testing, functional and integration testing. Both of the testing process (adding new application and SaaS environment) can be done separately until interoperability testing is done.

The flow of unit testing, functional testing, interoperability testing for the environment and dashboard is shown in Figure 4. **JUnit** is used to test each class containing services and module of Dashboard. All **Plutus Dashboard** functional requirement [22] has also been tested.

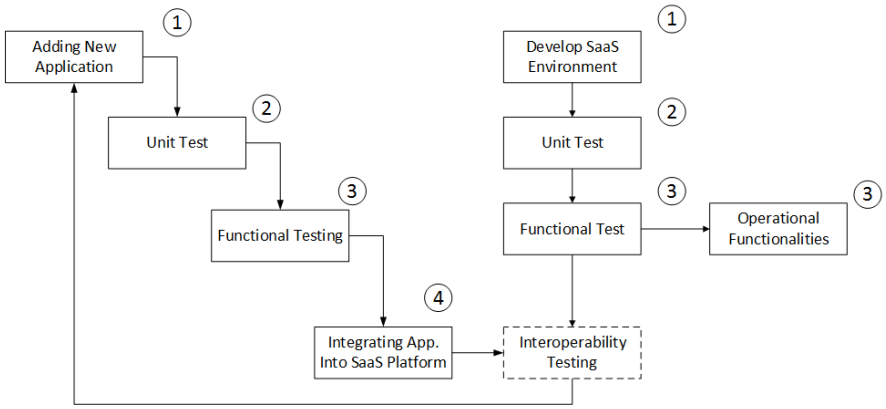

Fig. 4. The Whole Testing Proces

Some of the functionalities are related to SaaS operational issues such as measurement, transaction, and performance information. **Plutus** provides operation related functionalities that are needed by ISV such as registering SaaS application and creating new multi-tenant database, new database schema for new tenant, and new billing model. The other functionalities have been performed in interoperability testing.

Once the environment is ready, we can add SaaS application to the environment by doing the configuration steps that have been explained in section 5. This process can be easily done without modifying the existing application code, as we take benefit from AOP technology.

JUnit is also used for testing each application (**Vulcan**, **Affablebean**) before integration. The purpose of functional testing is to assure that each application will meet the requirement. This functional testing was performed on each functional requirement of **Vulcan** and **Plutus**. **Vulcan** has 18 functional requirements [22] and has been tested with test scenarios in Figure 4. Once an application passed Unit and functional test, it is ready to be deployed in **Olympus**.

Interoperability testing is used to show how easy is to start an existing application to operate. For this test, **Vulcan** and **Affablebean** are integrated into **Plutus**. These tests are conducted after functional testing is done. We register **Vulcan** and **Affablebean** in **Plutus** application management and configure billing management. We also include **Plutus** library in those application projects, and configure aspect class in **Plutus** library according to their needs. All of these integration processes do not require modification from the existing code. However, the three testing processes must be done, each time ISV needs to add a new application to the environment.

7 Conclusion

In this research we have identified general SaaS application requirement from two points of views that of ISV and of tenant. Concerns are derived from those requirements. Eight (8) concerns are identified, including tenant management, discriminator, monitoring, performance management, measurement, billing management, and application management. Each concern has been implemented as modules, service, or library. This research also proposed SaaS system architecture that contains all of the proposed concerns. Those eight concerns are implemented as library and services. As a result, an architecture consists of sets of SaaS applications and a SaaS operational system are proposed. Each application in the set of applications will utilize SaaS operational system as a centralized management system, so it can be operated as SaaS model. By using AOSD approaches on this architectural design, both ISV and tenant can take benefit. Using the AOP technology, ISV easily deploy an application before operation where tenant can have his own database and configuration. Furthermore, through resource measurement and performance monitoring, ISV and tenant have a transparent and accountable billing. ISV can separate its concern to tenant requirement, and reuse **Plutus**. Integration of **Affablebean** showed that the architecture permits a non aspect oriented application to be integrated and operated as SaaS after small effort of tailoring.

8 References

1. M. Turner, D. Budgen and P. Brereton, "Turning Software into a Service," IEEE Computer Society, pp. 38-44, 2003.
2. "What is Software as a Service," [Online]. Available: http://www.salesforce.com/saas/.
3. G. Carraro and F. Chong, "Software as a Service (SaaS): An Enterprise Perspective," October 2006. http://msdn.microsoft.com/en-us/library/aa905332.aspx.
4. I. Sommerville, Software Engineering, Pearson, 2009.
5. P. A. Laplante, J. Zhang and J. Vias, "Distinguishing Between SaaS and SOA," IEEE Computer Society, pp. 46-50, 2008.
6. D. C. Chou and A. Y. Chou, "Software as a Service (SaaS) as an outsourcing model: an economic model analysis," pp. 386-391, 2011.
7. C. T. Hancheng LIAO, "An Anatomy to SaaS Business Mode Based on Internet," International Conference on Management of e-Commerce and e-Government, pp. 215-220, 2008.
8. F. Chong and G. Carraro, "SaaS Simple Maturity Model," 2006. http://blogs.msdn.com/b/gianpaolo/archive/2006/03/06/544354.aspx.
9. A. Kalisa, Kajian Software as a Service denganStudiKasusRumahSakit di Daerah Rural danTerpencil, Bandung, 2012.
10. J. Brichau and D. T. Hondt, "Aspect-Oriented Software Development - An Introduction," pp. 1-20.
11. S. M. R. I. Sutton, "Concern Modeling for Aspect-Oriented Software Development," in Aspect Oriented Software Development, 2004, pp. 479-505.
12. IEEE, "IEEE recommended practice for architectural description of software-intensive-system," IEEE Std, 2000.
13. S. Chate, "Convert your web application to a multi-tenant SaaS solution," 14 December 2010. [Online]. Available: http://www.ibm.com/developerworks/cloud/library/cl-multitenantsaas/.
14. K. Tang, J. M. Zhang and Z. B. Jiang, "Framework for SaaS Management Platform," IEEE International Conference on E-Business Engineering, pp. 345-350, 2010.
15. W.-T. Tsai and X. Sun, "SaaS Multi-Tenant Application Customization," IEEE Seventh International Symposium on Service-Oriented System Engineering, pp. 1-12, 2013.
16. "The NetBeans E-commerce Tutorial - Designing the Application," 11 September 2013. [Online]. Available: https://netbeans.org/kb/docs/javaee/ecommerce/design.html.
17. "Spring Framework," GoPivotal, 2013. http://projects.spring.io/spring-framework/. [Accessed 26 September 2013].
18. "Jersey - RESTful Web Services in Java.," Oracle, 2013. https://jersey.java.net/. [Accessed 26 September 2013].
19. S. Souza, "JAMon (Java Application Monitor)," 2013. http://jamonapi.sourceforge.net/. [Accessed 26 September 2013].
20. F. Chong, G. Carraro and R. Wolter, "Multi-Tenant Data Architecture," June 2009. [Online]. Available: http://msdn.microsoft.com/en-us/library/aa479086.aspx.
21. S. Aulbach, T. Grust, D. Jacobs, A. Kemper and M. Seibold, "A Comparison of Flexible Schema for Software as a Service," SIGMOD, pp. 881-888, 2009.
22. A. Suwandi, Kajian Aspek pada Aplikasi SaaS, Bandung: ITB, 2013.