# Evolving Multi-Tenant SaaS Cloud Applications Using Model-Driven Engineering

Assylbek Jumagaliyev
School of Computing and Communications
Lancaster University
United Kingdom
a.jumagaliyev@lancaster.ac.uk

Jon Whittle
School of Computing and Communications
Lancaster University
United Kingdom
j.n.whittle@lancaster.ac.uk

Yehia Elkhatib
School of Computing and Communications
Lancaster University
United Kingdom
y.elkhatib@lancaster.ac.uk

## ABSTRACT

Cloud computing promotes multi-tenancy for efficient resource utilization by sharing hardware and software infrastructure among multiple clients. Multi-tenant applications running on a cloud infrastructure are provided to clients as Software-as-a-Service (SaaS) over the network. Despite its benefits, multi-tenancy introduces additional challenges, such as partitioning, extensibility, and customizability during the application development. Over time, after the application deployment, new requirements of clients and changes in business environment result application evolution. As the application evolves, its complexity also increases. In multi-tenancy, evolution demanded by individual clients should not affect availability, security, and performance of the application for other clients. Thus, the multi-tenancy concerns add more complexity by causing variability in design decisions. Managing this complexity requires adequate approaches and tools. In this paper, we propose modeling techniques from software product lines (SPL) and model-driven engineering (MDE) to manage variability and support evolution of multi-tenant applications and their requirements. Specifically, SPL was applied to define technological and conceptual variabilities during the application design, where MDE was suggested to manage these variabilities. We also present a process of how MDE can address evolution of multi-tenant applications using variability models.

## Keywords

Evolution; multi-tenancy; variability; cloud computing; cloud application; software product lines; model-driven engineering

## 1. INTRODUCTION

Cloud computing provides on-demand, scalable, and flexible computing resources to develop and deploy cloud applications [1]. Applications deployed on cloud are provided to clients as services over the Internet and are known as SaaS. As mentioned in [2], one key attribute of SaaS is multi-tenant efficiency, which enables economies of scale and efficient resource utilization by sharing a cloud infrastructure across multiple clients (i.e., tenants). A tenant is an organization or company with its end users that uses SaaS application.

As illustrated in Figure 1, there are generally two multi-tenancy patterns [3]: multiple instances multi-tenancy and single instance multi-tenancy. In the former, each tenant has a dedicated application instance on a shared hardware, operating system, or middleware. In the latter, tenants are served by a single application instance that runs on shared hardware and software infrastructure. We explore and address challenges that relate to the latter multi-tenancy pattern where tenants require isolation in

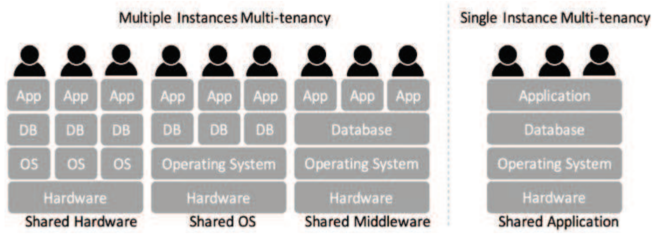


**Figure 1. Multi-tenancy patterns**

application and database. Tenants may also want to extend or customize a business process workflow to cater for their specific needs. However, extensions and customizations of individual tenants should not affect the use of the application by other tenants. Thus, partitioning, extensibility, and customizability challenges emerge during the application development.

Over time, applications evolve because of changes in tenant requirements or new tenant requirements [6]. The evolution may imply changes in the application structure. Usually, cloud applications consist of several layers (e.g., presentation layer, data logic layer, and business logic layer) and changes in any layer may entail changes in other layers. Moreover, multi-tenancy requires the following architectural considerations to be addressed. First, the application layers must be multi-tenant aware to ensure tenant isolation. Second, the application must allow per tenant customization. Finally, each layer must scale independently of each other.

Cloud providers offer various technologies and tools for cloud application development. Nevertheless, multi-tenancy concerns cause additional variability challenges in design decisions such as different multi-tenant data architectures, partitioning schemas and design patterns. The variability represents different available options to implement a certain functionality and it should be considered in the whole lifecycle of multi-tenant applications to meet tenant requirements, and to leverage resource pooling and scalability of the cloud.

Variability can be efficiently managed using SPL techniques. Mainly, SPL engineering focuses on the development of software products from reusable core assets [7]. In SPL, software systems share common functionality, but each software system has some variable functionality [5].

Modeling the variability can also help to efficiently evolve applications. During the application development a set of variability models can be chosen for a given cloud deployment. When the application evolves, it is possible to evolve the corresponding code by selecting another set of options from the variability model. For example, a multi-tenant data architecture can be modeled in different ways: 1) single database shared by all tenants, 2) a separate database for each tenant, or 3) multiple database instances where each instance serves a group of tenants. Initially, the developers might select a single database for all tenants. However, the security requirements of tenants may require a more isolated approach that cannot be provided in a single database instance. Therefore, the developer selects another multi-tenant architecture and the application evolves to multiple database instances.

The main contribution of our ongoing research is exploring combination of SPL and MDE techniques for managing variability in design decisions and evolving multi-tenant cloud applications. Others have advocated the integration of SPL and MDE for managing variability in multi-tenant cloud applications. For example, in [10], Orthogonal Variability Model (OVM) and Service Oriented Modeling Language (SoaML) were used to model variability and customizability in cloud applications. While in [4], a framework was proposed to model customizable multi-tenant cloud applications and to support their evolution. However, these approaches address application variability, customizability, and limited evolution scenarios, such as onboarding new tenants, removing tenants, and tenant customizations. In our approach, we use SPL to identify technological and conceptual variability prior to application implementation, where MDE concepts are applied to manage variability. Subsequently, variability models may efficiently support evolution of applications and their requirements. Moreover, we illustrate our approach by a multi-tenant application example.

The reminder of the paper is structured as follows. Section 2 describes variability in multi-tenant applications and their evolution. It also describes SPL and discusses related work in the field. Section 3 explains our approach for addressing variability and evolution challenges in multi-tenant applications. Section 4 presents a case study to motivate and illustrate our work. Finally, Section 5 concludes the presented approach.

## 2. BACKGROUND

In this section, we briefly explain variability in multi-tenant applications and their evolution. We also describe SPL and give an overview of related work.

### 2.1 Variability

Variability emerges in all levels of cloud applications. Abu-Matar et al. [4] categorized the variability into the following levels: application variability, business process variability, platform variability, provisioning variability, deployment variability and provider variability. Through this paper, we consider application variability and business process variability.

In application variability, different tenants may have different functional and non-functional requirements in addition to the core application. In business process variability, tenants may have varying business workflows. Therefore, the application must enable configuration and customization to meet tenant's goals and requirements. In [8], variability is separated as customer-driven variability and realization-driven variability. The customer-driven variability comprises tenant requirements. We can classify application and business process variability as customer-driven variability. The realization-driven variability represents different implementation options derived by customer-driven variability. In this paper, we use design decision variability as realization-driven variability.

### 2.2 Evolution

Evolution is an inevitable process in any software system [6] and multi-tenant applications are no exception. There are several reasons that trigger application evolution, such as fixing bugs, changes in business environment, improving security and reliability, changes in tenant requirements, or new tenant requirements. Applications should respond to such changes to maintain tenant satisfaction. In application level multi-tenancy, changes must be adapted at runtime without affecting availability, security, and performance of an application for other tenants. A key problem is implementing and managing required changes in applications [6].

### 2.3 SPL

SPL is a software engineering approach that focuses on the development of software products from reusable core assets [7]. It promotes feature modeling to analyze and identify the commonality and variability in applications [5]. Features are specific characteristics of an application and are classified in terms of capabilities, domain technologies, and implementation techniques [7]. Capabilities represent functional and non-functional characteristics that are provided by an application to clients. Domain technologies describe how to implement features regarding an underlying domain, where implementation techniques comprise commonly used generic approaches in the development. Features are also grouped as mandatory, optional, alternative and at-least-one-of (OR). Common features are mandatory features, while variability features may be optional, alternative or at-least-one-of. Optional features can be selected or neglected, only one feature must be selected from alternative features, and one or more features can be selected from at-least-one-of features.

### 2.4 Related work

Several authors have proposed using SPL or MDE techniques for managing variability in cloud applications to address multi-tenancy concerns. Moreover, there are some tools and frameworks for deploying, provisioning or supporting portability of cloud applications. However, none combined the strength of these two paradigms to address the multi-tenancy challenges, design decision variability challenges and evolution complexity.

#### *2.4.1 MDE and SPLs*

Mietzner et al. [8] proposed variability management in multi-tenant SaaS applications and their requirements using explicit variability models of SPL. Initially, the customer-driven variability and realization-driven variability were modeled using Orthogonal Variability Model (OVM). Then, the model was used to support customizability in applications. The authors also supported efficient SaaS applications deployment for new tenants based on the information about already deployed SaaS applications. Nevertheless, this approach addresses the application variability and does not support evolution.

Service line engineering (SLE) [9] (i.e., combination of service-oriented development and SPL) was introduced for customizable multi-tenant SaaS application development. SLE uses feature modeling to address engineering complexity and manage variability caused by application-level multi-tenancy. The main departure from SPL is that customizations are applied to a single application instance that is shared across multiple tenants. The author emphasized that SLE also supports application evolution.

Kumara et al. [11] described an approach for realizing service-based multi-tenant applications. This approach is also feature-oriented as SLE and it supports evolution by enabling runtime sharing and tenant-specific variations using Dynamic SPLs.

CloudML [12], CAML [13], and CloudDSL [14] are examples of modeling languages for cloud applications that exploited MDE techniques. CloudML automates provisioning for cloud applications that run on multiple clouds. CloudDSL supports portability of applications by describing cloud platform entities, whereas CAML supports deployment and enables migration of existing applications to cloud. However, none of these modeling languages addresses multi-tenancy in design decisions or evolution of applications.

### *2.4.2 Combining MDE and SPLs*

Shahin [10] integrated SPL and MDE to model variability for customizable SaaS applications. In this approach, SoaML was extended to model variability in all layers of Service Oriented Architecture (SOA). OVM from SPL was exploited to model variability as separate models. These separate models were used to generate a customization model for SaaS applications.

Cavalcante et al. [15] applied feature modeling to manage commonality and variability in cloud applications. In addition, they modeled costs regarding the use of cloud resources to minimize expenditure. They also used UML class diagram for features to identify dependencies.

Abu-Matar et al. [4] described a framework for modeling service-oriented customizable multi-tenant cloud applications. They exploited SPL for managing variability in services from multiple views (i.e., service-oriented views and cloud views). They also applied MDE for modeling multi-tenant aware application artifacts. In [17], the framework was complemented to support some evolution scenarios such as onboarding new tenants and removing tenants. In our approach, we address multi-tenancy concerns by modeling variability in design decisions that emerges during the architecting process. Thus, developer can use variability models for further support throughout the whole lifecycle of multi-tenant cloud applications.

## 3. OUR APPROACH

We consider an integration of feature modeling concepts and MDE techniques to address the design decision variability and evolution complexity in multi-tenant cloud applications. Our approach is based on the work of Jayaraman et al. [16]. The main idea of this approach is maintaining feature separation and detection of structural dependencies and conflicts between features during analysis and design modeling. Features or groups of features are modeled using UML, and a model composition language, MATA (Modeling Aspects using a Transformation Approach), detects relationships and conflicts. However, this approach requires additional work to support cloud application development and multi-tenancy.

Figure 2 illustrates modeling multi-tenant applications that consists of the following steps. Initially, common and variable functional and non-functional features with dependencies are captured using feature modeling. This helps to define available implementation options for the design decisions. Next, common features are used to model the core of the application using an UML composition language. Each variant feature is modeled in the MATA language with dependencies to the core UML model and relations to other features. This allows features to be modeled independently of each other and enables reuse of models. Further, a composed UML model is generated from the core UML model and selected models from models of variant features. At this stage,

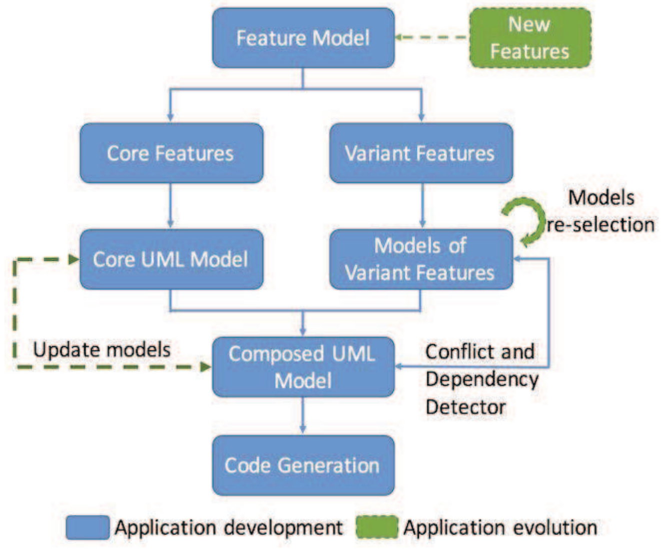


**Figure 2. Multi-tenant application development and evolution with MATA**

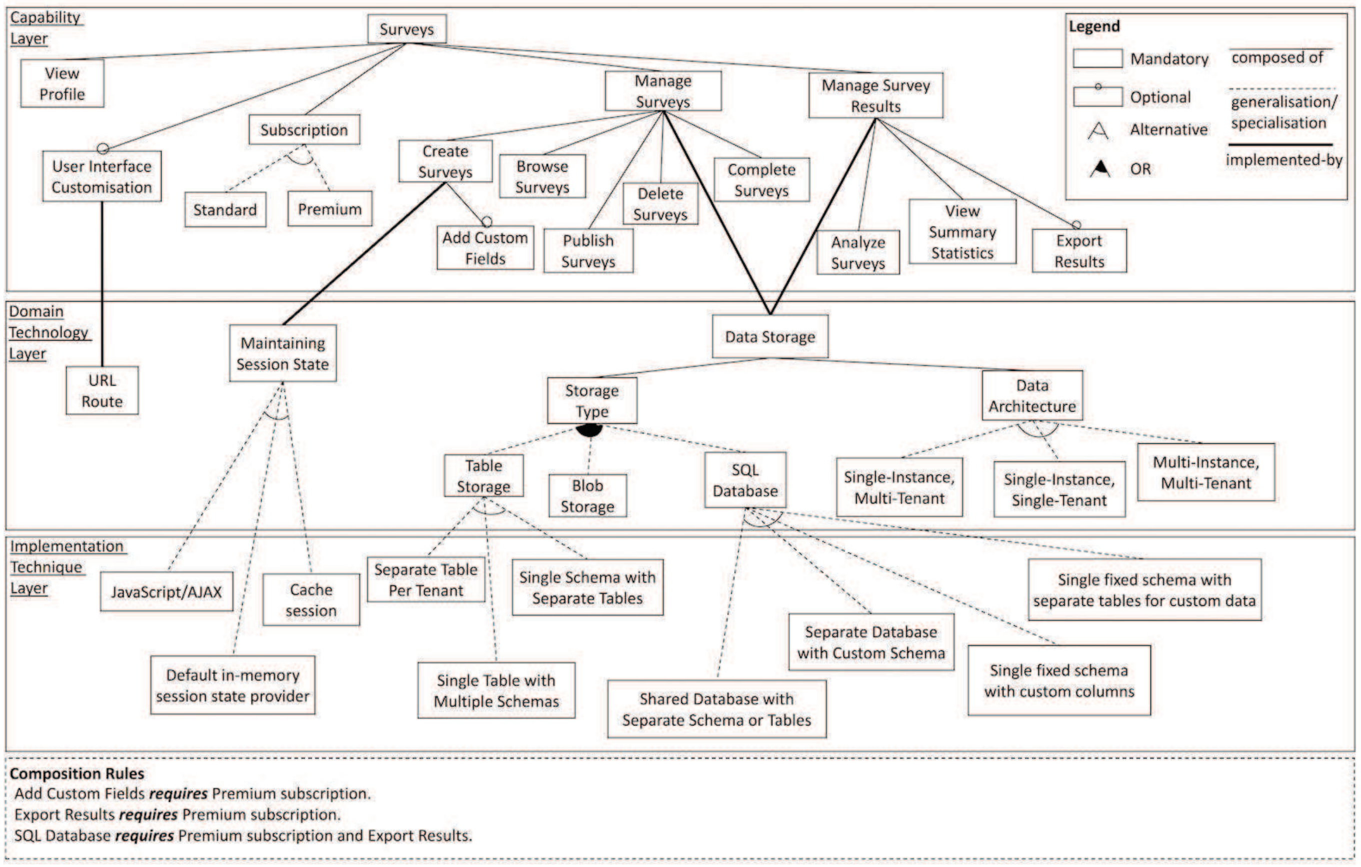
conflicts and dependencies of models are checked. Finally, source code specific to a particular cloud platform is generated.

Figure 2 also describes application evolution which may require models re-selection, adding new features, or a combination of both. In the case of model re-selection, developers pick appropriate features from the models of variant features. When evolution demands adding new features, developers identify whether new features are common or variable. The new common features affect the existing core UML model, whereas for each variable feature a corresponding model of variant feature is created. There might be cases when all new features are common or variable. In the former, only the core UML model is updated. While in the latter, new models are added to the models of feature variants and it requires models re-selection. Then, developers generate a composed UML model and source code.

## 4. CASE STUDY

To explore our approach, we present a Surveys service [2] case study by Microsoft. *Surveys* is a multi-tenant SaaS application for creating and managing online surveys. Tenants can create, publish surveys, and analyze results. Three different actors interact with the application: the application provider administrator, the tenant administrator, and the survey respondent. The application provider administrator manages all tenants and their surveys, whereas the tenant administrator manages its own surveys and survey results, and the survey respondent completes surveys.

Although multiple tenants use the same application instance with core functionalities and user interface layouts, each tenant can view and edit its own data. In addition, the application allows tenants to apply user interface customization by uploading their corporate logo, adding tenant name, welcome text, and contact details. Besides, tenants can customize the business process by choosing a standard or premium subscription type. With standard subscription, tenants can publish a limited number of surveys and cannot export their survey results. Premium subscription tenants can create and publish any number of surveys, export survey results for further analysis, and their requests are prioritized by the application.


Capability Layer
Surveys
View Profile
User Interface Customisation
Subscription
Standard
Premium
Create Surveys
Add Custom Fields
Browse Surveys
Publish Surveys
Manage Surveys
Delete Surveys
Complete Surveys
Manage Survey Results
Analyze Surveys
View Summary Statistics
Export Results
Legend
Mandatory
Optional
Alternative
OR
composed of
generalisation/ specialisation
implemented-by
Domain Technology Layer
URL Route
Maintaining Session State
Data Storage
Storage Type
Table Storage
Blob Storage
SQL Database
Data Architecture
Single-Instance, Multi-Tenant
Single-Instance, Single-Tenant
Multi-Instance, Multi-Tenant
Implementation Technique Layer
JavaScript/AJAX
Cache session
Default in-memory session state provider
Separate Table Per Tenant
Single Schema with Separate Tables
Single Table with Multiple Schemas
Shared Database with Separate Schema or Tables
Separate Database with Custom Schema
Single fixed schema with custom columns
Single fixed schema with separate tables for custom data

**Composition Rules**
Add Custom Fields ***requires*** Premium subscription.
Export Results ***requires*** Premium subscription.
SQL Database ***requires*** Premium subscription and Export Results.

**Figure 3. The feature model of the Surveys application.**

## 4.1 Applying our Approach

As a first step, we constructed a feature model to define commonalities and potential variabilities in the application. An excerpt of the feature model is illustrated in Figure 3. As mentioned in Section 2.3, features were identified and categorized into three layers. The capability layer comprises the functional and non-functional features that are available for tenants. The domain technology layer describes the way of implementing features from the capability layer, and the implementation technique layer represents generic techniques to implement features on a cloud infrastructure. Further, the features were classified as mandatory, optional, alternative features, and at-least-one-of (OR). The mandatory features are common features that represent core components of the application that will always be present in any evolution of the cloud application. Whereas the optional, alternative and at-least-one-of features are variable features that describe different possible implementations. Once the common and variable features are defined, the process (as defined in Figure 2) would come up with a core UML model from the common features and models of variant features from the variable features. As a next step, a composed UML model from the core UML model and selected models of variant features would be generated.

Figure 3 shows that various options were modeled in the domain technologies and implementation techniques for realizing certain features. These variability models are used to support evolution. For example, the application uses a single database instance shared by all tenants. However, as the number of users per tenant increases, a more isolated approach must be selected from variability models to meet user requirements. With the MATA language multi-tenant data architectures are modeled separately with their dependencies to the core model and can be easily reused. Hence, developers can select any other multi-tenant data architecture model at any time during the application evolution.

## 4.2 Evolution Scenarios

Over the application lifetime, the functionality and quality of service offered by the application must increase to meet tenants' requirements. In this section, we consider some evolution scenarios that affect design decisions in the application structure.

When architecting the application structure, we decided to use a single database instance shared by all tenants. However, over time the number of tenants increases. Therefore, the number of concurrent end users and amounts of data stored by each tenant increase as well. Moreover, some tenants may require a separate database due to privacy requirements. These scenarios require a more isolated data storage approach and entail model re-selection from models of variant features. Thus, developers select either a single database instance for each tenant or multiple database instances for multiple tenants from the available data architecture models (as depicted in Figure 3).

For maintaining a session state while creating a new survey, we suggest JavaScript/AJAX technologies. This approach is simple, easy to maintain, scalable, and secure compare to other available implementation techniques under the Maintaining Session State feature. However, it relies on client-side JavaScript that makes it the least robust solution among available techniques. In the future, to improve robustness and effectiveness, developers must decide between default in in-memory session state provider and cache session. This scenario also requires model re-selection from existing models of variant features.

Another typical scenario is adding new features. For example, tenants may want to perform complex analysis on survey results. Currently, the application stores survey answers in blob storage. To provide the new feature, an SQL database (from different models under Storage Type) is the best solution for applying complex queries and join query. When adding a new feature, developers must identify whether the new feature is common or specific to certain clients. If the feature is common, the core UML model will be updated. If the feature is variable, the core UML model will remain the same and a model of variant feature for this variable feature will be generated. At this point, the MATA language detects relations and dependencies of the new feature to other features. The SQL Database also needs partitioning to support multi-tenancy. Thus, the developers must select one of the different partitioning models for SQL databases. Moreover, a new interface must be implemented to view and analyze survey data.

## 5. CONCLUSION

In this paper, we have proposed an integrated SPL and MDE modeling approach to address design decision variability and evolution concerns in multi-tenant SaaS cloud applications. We have applied feature modeling concepts to identify variability in implementation. The MATA language has been suggested to manage variability, and to support customization and evolution. Thus, the proposed approach allows features to be modeled independently. Furthermore, conflicts in the application structure and dependencies between models are detected. However, it requires improvements to enable cloud application development and multi-tenancy.

In our future work, we plan to enhance our approach by making the MATA language applicable for multi-tenant SaaS cloud applications and by developing a model to code transformation prototype to transform composed models to source code. A case study will be carried out to illustrate and evaluate the implemented tool. Moreover, we will compare our approach with other tools to identify benefits and drawbacks.

## 6. REFERENCES

[1] P. Mell, et al., "The NIST Definition of Cloud Computing", National Institute of Standards and Technology, Special Publication 800-145, Bethesda, Maryland, 2011

[2] D. Betts, et al., "Developing Multi-Tenant Applications for the Cloud on Windows Azure", Microsoft Patterns and Practices, 2013

[3] J. Guo, et al., "A framework for native multi-tenancy application development and management", Proceedings of the 9th IEEE Conference on E-Commerce Technology and the 4th IEEE Conference on Enterprise Computing, E-Commerce and E-Services, pp. 551–558, 2007.

[4] M. Abu-Matar, et al., "Towards Software Product Lines Based Cloud Architectures", Proceedings of the IEEE Conference on Cloud Engineering, 2014

[5] H. Gomaa, "Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures", Addison-Wesley Professional, 2004

[6] I. Sommerville, "Software Engineering", Pearson, 2010

[7] K. Lee, et al., "Concepts and guidelines of feature modeling for product line software engineering", Proceedings of the 7th Conference on Software Reuse: Methods, Techniques, and Tools, pp. 62–77, 2002.

[8] R. Mietzner, et al., "Variability modeling to support customization and deployment of multi-tenant-aware Software as a Service applications", Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems, pp. 18-25, 2009

[9] S. Walraven, et al., "Efficient Customization of Multi-tenant Software-as-a-Service Applications with Service Lines", Journal of Systems and Software, Vol. 91, pp. 48-62, 2014.

[10] A. Shahin, A "Variability Modeling for Customizable SaaS Applications", International Journal of Computer Science and Information Technology, 6(5), pp. 39-49, 2014.

[11] I. Kumara, et al., "Sharing with a difference: Realizing service-based SaaS applications with run-time sharing and variation in dynamic software product lines", IEEE Conference on Services Computing, pp. 567–574, 2013

[12] A. Bergmayr, et al., "The Evolution of CloudML and its Manifestations", Proceeding of the 3rd Workshop on CloudMDE, 2015

[13] A. Bergmayr, et al., "UML-Based Cloud Application Modeling with Libraries, Profiles and Templates", Proceedings of the 2nd Workshop on CloudMDE, 2014.

[14] G. S. Silva, et al., "Cloud DSL: A Language for Supporting CloudPortability by Describing Cloud Entities", Proceedings of the 2nd Workshop on CloudMDE, 2014.

[15] E. Cavalcante, et al., "Exploiting Software Product Lines to Develop Cloud Computing Applications," the 16th Software Product Line Conference, 2012.

[16] P. Jayaraman, et al., "Model Composition in Product Lines and Feature Interaction Detection Using Critical Pair Analysis", Conference on Model Driven Engineering Languages and Systems, 2007

[17] F. Mohamed, et al., "SaaS Dynamic Evolution Based on Model-Driven Software Product Lines", Proceedings of the IEEE 6th Conference on Cloud Computing Technology and Science, 2014