Software Product Line Architectures for Cloud-based Applications

Graduate Students of Computer Science at Illinois Tech

Chen, Feiyu (A20349269),
Elango, Subakar Sundarababu (A20345012),
Han, Seungho (A20360115),
Moon, Sangwon (A20259651),
Tiwari, Saurabh (A20356308)

Abstract

The paper which we are writing here covers important topics of Software Product Lines Architecture for cloud based application. The paper involves reading of five of the most renowned papers published in ACM, Computer Org and IEEE in the field of cloud and merges the information to form a concrete information on different types of architecture that are used today. The architecture of Software Product Lines makes use of various variables that need to be taken care while designing a cloud application by Software Product Line architecture model. This includes variability study, the different requirements and different cloud services available.

Keywords

Software Product Lines (SPL), Cloud Computing, Cloud-Architecture, IaaS, PaaS, SaaS, Variability.

1. Introduction

In this paper we went through a series of papers published in world renowned journals and studied how Software Product Lines are used to develop cloud based application. This study not only includes the comparison of different architectures present but also a comprehensive architectural patterns and the comparison between each of the studied SPL Architectures. The term paper studies the following contents taken from various papers.

- Studying variability in cloud to develop an SPL for most of the requirements.
- Software product line and selection strategy.
- Architecting Cloud Applications using SPL techniques
- Applying SPL to create customizable SaaS application.

The variety of topics covered in this paper gives an in depth analysis on how would someone architect a cloud platform to include software product lines for developing a generic component that is common for any kind of application. The world and top industries have moved their focus from a self hosted servers to cloud services, which provide for the variety of requirements.

In the end, we review a paper which discussed about the successful implementation of SPL which are currently in practice. That includes a brief discussion on Microsoft Azure, Amazon AWS, and other cloud platforms. We also provide some of the case studies that are done by John D McGregor, who is a professor of Computer Science in Clemson University, and his team who have an intensive experience in cloud industry.

2. Motivation

The key motivation behind this paper is to find the advantages of using SPL techniques in architecting cloud platforms for different application. As we discuss the SPL techniques and advantages we also find the commonality and variability study in the papers which are used for designing SPL architectures for cloud based application.

3. Background

With the advancement of the Cloud Computing paradigm, new challenges in terms of model and tools for building and deploying complex software systems arise. In the heterogeneous scenario of this new paradigm, the development of applications using cloud services becomes hard, and the software product lines (SPL) approach is potentially promising for this context since specifications of the cloud platforms, such as services heterogeneity, pricing model, and other aspects can be catered as variabilities to core features.

3.1 *Software Product Lines (SPL)*

A software product line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a market segment or mission and that are developed from a common set of core assets in a prescribed way. The method epitomizes strategic and planned reuse. More than a new technology, it is a new way of doing business that can yield order-of-magnitude improvements in time to market, cost, and quality and other business drivers. Software product line engineering can also enable rapid market entry and flexible response and provide a capability for mass customization [1].

Cloud computing is a model for enabling ubiquitous, convenient on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [2]. According to the National Institute of Standards and Technology (NIST), these are the five specific qualities that define cloud computing:

- On-demand self-service
- Broad network access
- Resource pooling
- Rapid elasticity or expansion
- Measured service

3.2 Software as a Service (SaaS)
The consumer uses the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, except for limited user specific application configuration settings (Example: Google docs, Office 365. Enterprise software such as Salesforce and Business by Design by SAP)[2].

3.3 Platform as a Service (PaaS)
The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application hosting environment (Example: Force.com - focuses on development of additions to well-known Salesforce CRM solution. Borderline example is Google AppEngine) [2].

3.4 Infrastructure as a Service (IaaS)
The capability provided to the consumer is to provision processing, storage, networks and other fundamental computing resources where the consumer can deploy and run arbitrary software which can include operating systems and other applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage and deployed applications; possibly limited control of networking components (e.g., host firewalls). For example AWS, Windows Azure, OpenNebula are some of the IaaS available today in the market [2].

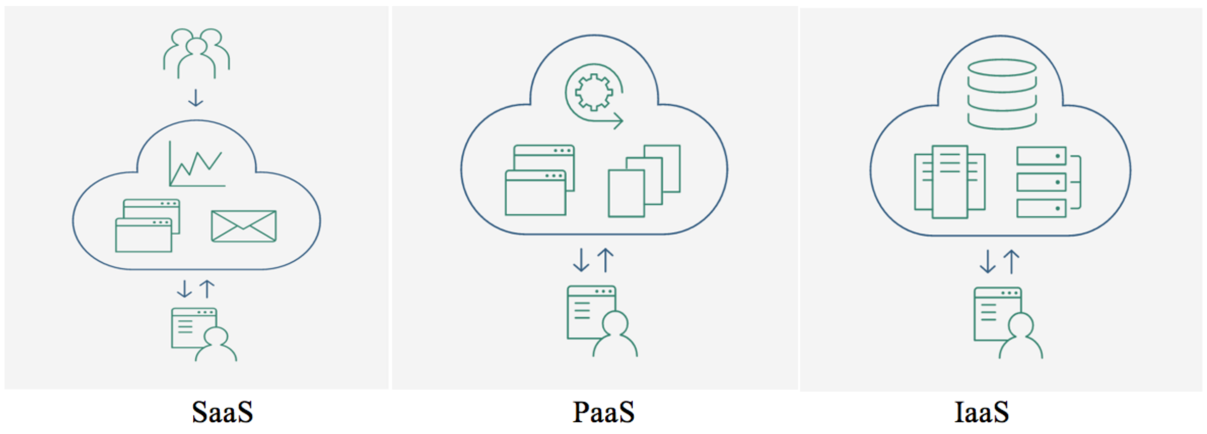


Fig. 1: Cloud Systems

4. SPL to Software Development Process

So far, we have gone through the cloud computing and details about software product line. In this phase, we present how SPL techniques can be applied to software development process and a real industry cloud tool with a target application, Installation Service.

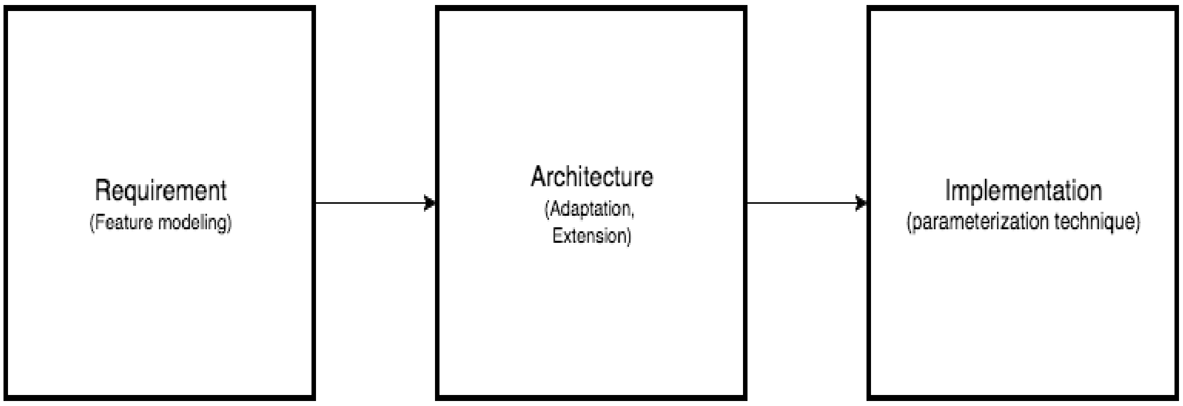


Fig. 2: SPL Development Cycle

4.1 Applying SPL technique to Requirements Phase

Now, we apply SPL technique from the beginning of software development process say requirement phase. The main goal of requirements engineering is the specification of common and variable domain requirements [16]. As a type of domain requirements, a method, called feature modeling, can be used to proceed requirement phase. This method which is originated from basic FORM (Feature-Oriented Reuse Method) is a feature modeling approach that encompasses activities for identifying and classifying features [15] approach helps to architect cloud computing system. An important thing is that recognizing features should be conducted by skillful domain manager with a set of documents including requirement document. Features are attributes those are corresponding to the entities in a system. It provides guidelines for capturing SPL shared parts and fluctuating parts. Feature models consist of features and communication among the features and it is represented by de-integration of a system. Features can be as follows.

- Mandatory, which are common to all products.
- Optional, which may or may not be selected for a particular product.
- Alternative, which results in one or more features being selected from a set.

Fig. 3 represents partial features and the relationships between the features on the Installation System. The features in the figure 7 have been allocated into following four domains using FORM access.

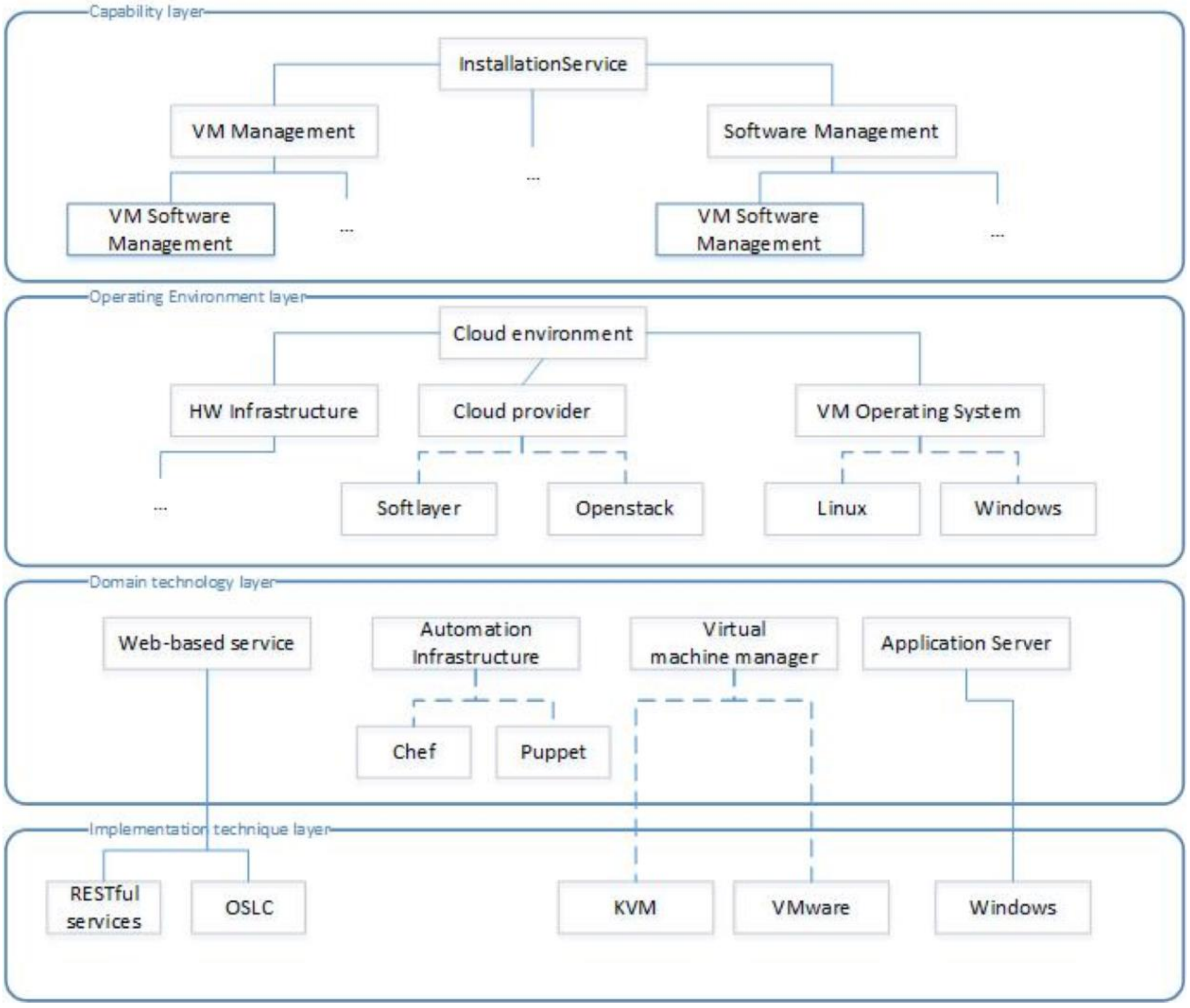


Fig. 3: Installation System Feature Relationship

- Capability layer shows the services such as virtual machine manager and software manager. Some features those are subject to these services are deeply affiliated to end users.
- Operating environment layer contains the attributes those are consumed by the operating system that is run by Installation Service. Some mandatory attributes like Cloud Provider or VM Operating System are included in this layer. Moreover, optional attributes are included since this Installation Service need to be put in the highest

position. Some attributes can be captured as feature models those accord with IaaS layer and the Installation Service itself relies on these optional services.

- Domain technology layer contains both mandatory and optional detailed technical features for specific domains. Whereas Web-based service, Automation infrastructure, Virtual machine manager and Application server are mandatory, Automation Infrastructure is optional.
- Implementation technique layer also shows detailed technical features but the features are more concrete than Domain technology layer. The features RESTful services and OSLC are taken to implement Web-based Service in Domain technology layer.

4.2 Applying SPL technique to Architecture Phase

We continue the development process based on the feature model created in the requirement phase. By taking the advantage of variation mechanisms, consists of required elements and variable elements, given from SPL architecture, we can achieve variability from a lot of software products [17]. We can take an example of Application architectures (or product architectures) as specializations of SPL architecture. In this architecture phase, the FORM approach is also used to create structural and behavioral pattern for making an application architecture. FORM approach offers ground rules for creation of SPL architectures that are composed of following models:

- Subsystem model defines the overall structure by grouping functionalities into subsystems.
- Process model represents the dynamic behavior of each subsystem.
- Module model relies on features at all levels to identify components [15].

As a result, application architecture is derived from an architectural structure consists of a multiple components and lines between the components representing architectural structure [18].

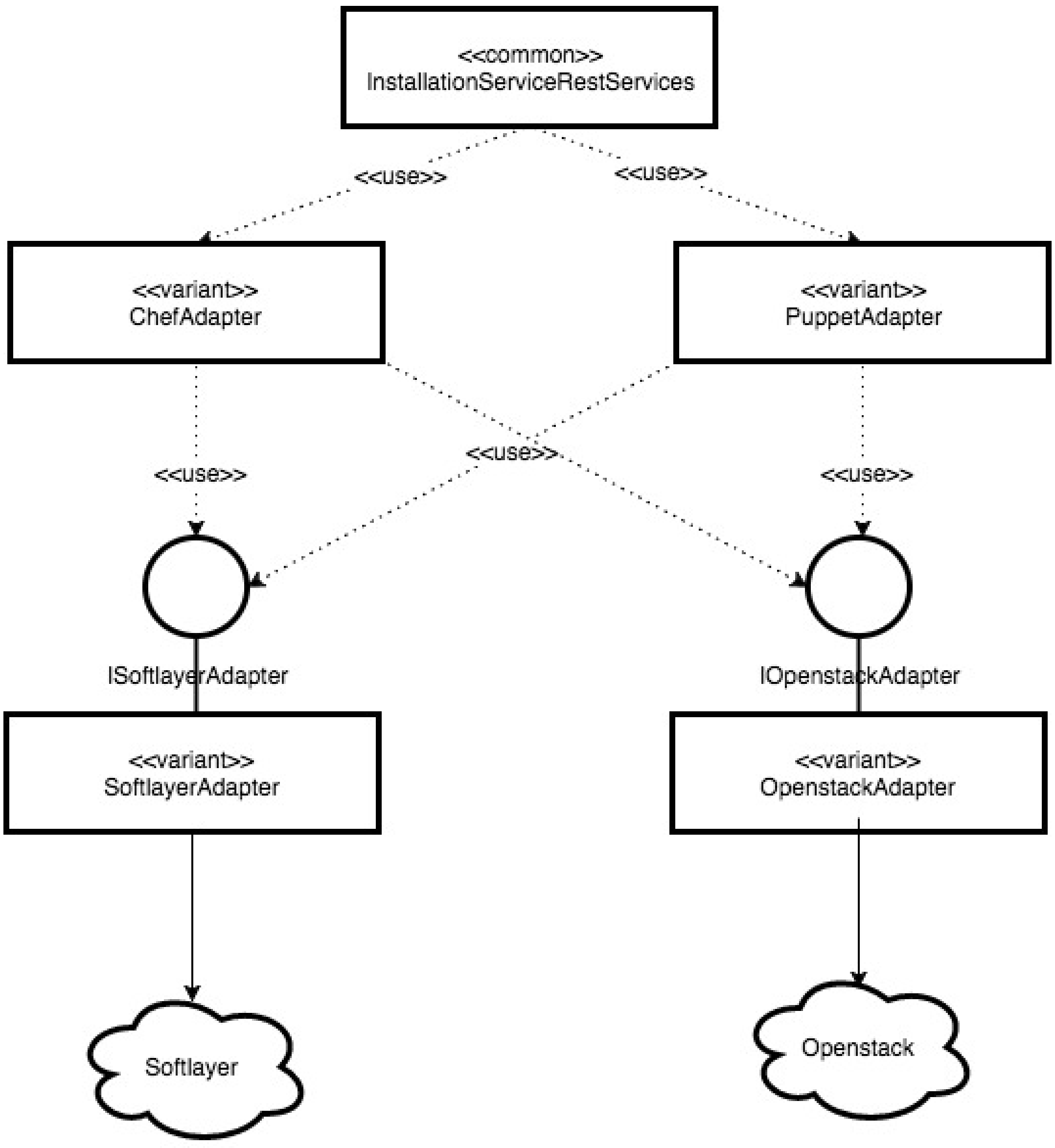
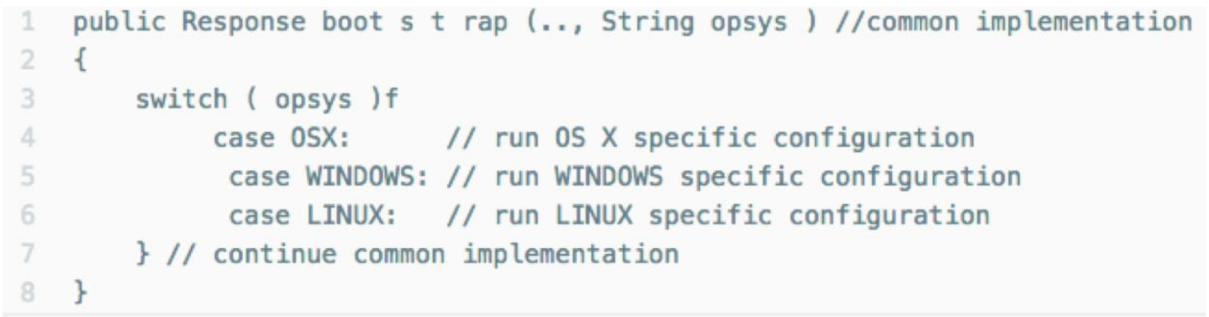


Fig. 4: Component-based software architecture

As seen in the figure 4 (redrawn from the original paper), the Installation Service architecture has been laid out using FORM approach. In this architecture phase, components have been used to model the SPL to achieve modularization and reusability. Using components also gives us advantages of utilization for achieving architectural variability using the techniques called adaptation and extension. For adaptation technique, a component provides an interface that is adaptable by other components those have the different interface from each other. The component "*InstallationServiceRestServices"* in the Figure 8. can be connected to both *Chef* and *Puppet* by providing each different interface to allow adapting from *Chef* and *Puppet*. Whereas adaptation technique requires a component to provide a changeable interface and extension technique requires a number of components to provide an interface for adding new features to itself. For instance, "*SoftlayerAdapter"* can be used as an adapter for a new component that will be added to the set of components.

4.3 Applying SPL Technique to Implementation Phase

Implementation using SPL technique is to implement a system based on SPL architecture [20] [21]. Till now, architectural aspects and its interfaces have been implemented for achieving the variability represented in the feature model. However, some few variability is implemented in the architecture phase even though major purpose of the architecture phase is to establish the variability. In other words, multiple variable features can be generated from architectural phase. In this implementation phase, binding times for compiling, loading, and run-time are decided according to the variability. For example, the Installation Service Tool provides same core services for a variety of nodes. However, the system should be able to run on diverse operating systems since they use different bootstrap configuration respectively. This variability issue can be solved by using a bootstrap method that has a parameter that represents the type of operating system as we can see in the figure 5. This technique is called *common parameterization technique* that is a dressed way of adaptation technique in the Architectural design phase.



```
public Response boot s t rap (.., String opsys ) //common implementation
{
    switch ( opsys )f
         case OSX:      // run OS X specific configuration
          case WINDOWS: // run WINDOWS specific configuration
          case LINUX:   // run LINUX specific configuration
    } // continue common implementation
}
```

Fig. 5: A Java example of implementation of the parameterization techniques

5. Architecting cloud applications using Software Product Lines

5.1 SPL - A formal definition

The formal definition of Software Product Lines as given by P. Clements and L. Northrop in their book. Software product line is a set of software-intensive system sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way. Organizations adopting product development strategy include a software product line have achieved impressive results, reducing product cycle time and increase productivity by an order of magnitude [13]. The software product line requires a commonality and variability analysis so that the host organization can characterise successful product line organizations.

Different Product lines use different production methods. Some software product line organizations have successfully used traditional programming languages and largely manual development techniques, while others have automated portions of the code generation for their products using model-driven techniques.[14] Various aspects of agile development methods have been integrated into SPL practices. Successful product line organizations intentionally select techniques and tools, models and processes that match their goals.[10]

5.2 Requirement for SPL based Cloud Architecture

In creating SPL based cloud architecture, we need to study commonality and variability. The paper written by Mohammad Abu Matar et. al [6] first discusses about various different cloud variability and based on that it talks about the different architectures. The paper talks about cloud based systems which are evolving to respond to changing client requirements. Cloud based applications can be modelled as SaaS families similar to the SPL products. As SPL development techniques rely on feature models to describe the commonality and variability of family member applications, such techniques can be used to model variability in SaaS.[6]

The above discussed paper is primarily a description of a unified and systematic framework for modelling cloud service in a vendor neutral manner. In addition the paper also demonstrates the applicability of the variability framework for building and customizing SaaS multi-tenant application. The approach is based on meta model that formalizes the multiple views of Service Oriented architecture SaaS.[10] The proof of concept (POC) which was done by the professor Matar and his team and they automatically generated the multitenant applications using the auto-generate scripts for writing application. The approach performed by the team of researchers facilitates the development of cloud SaaS families in a systematic, consistent and platform independent way. [6]

5.3 Studying Cloud Variability:

To achieve SPL based architectures for cloud applications, one needs to account the variability study which is thoroughly done in the paper[14]. A brief modelling of variability in the paper is as follows:

1. Application Variability - SaaS applications have varying functional and behavioural requirements in addition to the core SaaS application. Furthermore, tenants users may require further customization in the structure and look and feel of the UI
2. Business Process Variability where the business workflow may vary based on specific tenants' business requirements.
3. Platform variability- based on the applications Operating System and the programming language.
4. Provisioning Variability- This variability deals in the hardware networking, virtual machines and elasticity, generally observed in IaaS.
5. Deployment variability- dependency on where we are deploying the application, i.e. Public, private, community and hybrid clouds.
6. Provider Variability- which makes the application dependant upon whether it is specific to AWS, Google App Engine or SalesForce etc.

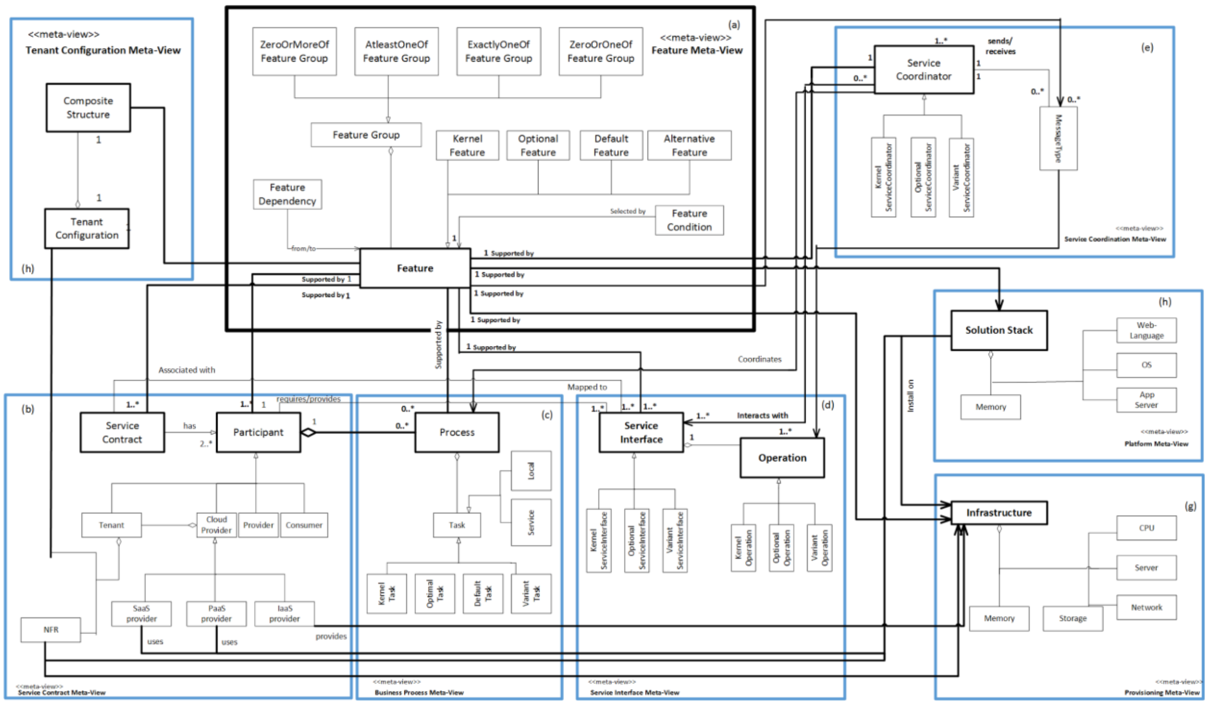


Fig. 6: Multiple view cloud variability Meta-Model

The Fig. 6 above is taken from the paper from Mohammad Matar et. al. (the figure has been redrawn) and it discussed about the Multiple View of Cloud Variability Meta Model. In the diagram drawn we could identify the various Meta-Views which correspond to a certain level of variability with respect to the Views. For example if you look at the Fig. 7 (zoomed view from the diagram) and focus on the provisioning meta-view you will find the details about infrastructure. In the infrastructure meta view we have a variability based on CPU, Server, Network, Memory, Storage etc. Similarly, in the platform Meta-view for analysing variability you can see the variable options as Web-Language, OS, AppServer, Database.[15]

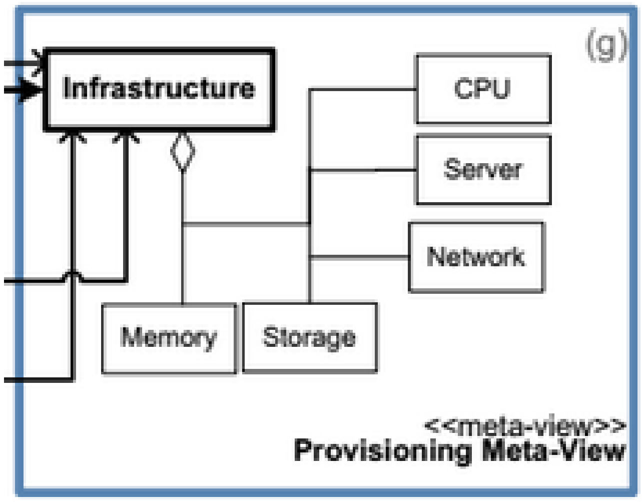


Fig. 7: Provisioning Meta-View

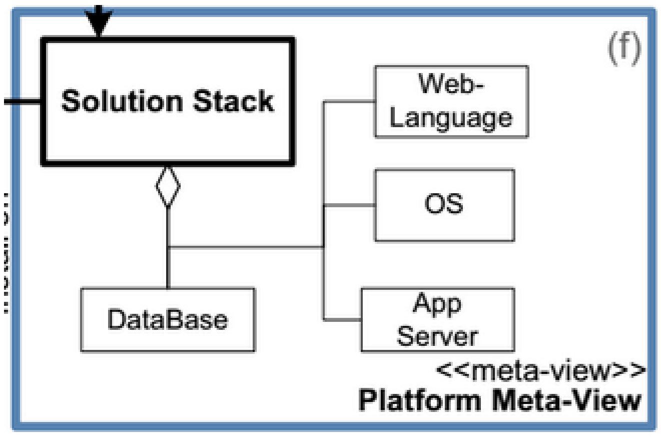


Fig. 8: Platform Meta-View

A variability analysis of feature, business process, service contract, service coordination is done accordingly and can be seen in the bigger picture in fig. 6.
When discussing feature variability it restructures all the variability issues as features and combine them in one and discusses them as features instead of separate meta views.

5.4 Combining Variability Views

We can combine all the variability views and present them in the form of a Feature which will result in a feature-meta variability and it can be explained as in the following way that manages commonality[14]:

- Capability Feature - functional variability in the domain.
- Platform Feature- Like, providing ASP, JSP, PHP, Windows Server, Apache Linux Server, LAMP, etc. e-commerce platforms can use various services like SimpleDB or RDS database service.
- Provisioning Feature- The infrastructure feature like CPU Memory, Storage and Network etc. For example an e-commerce SaaS provider may choose to build its application on a bare-metal dedicated infrastructure, or a virtualized infrastructure shared by other application tenants.
- Deployment Feature- Applications may be deployed on public, private, hybrid or community clouds.

5.5 Understanding Software Multitenancy

Multitenancy is an important term to understand for further reading of this paper, as the proof of concept developed by the group of researchers we use multitenant application instead of the same application running on a different instance itself. The term software multitenancy refers to software architecture in which a single instance of software runs on a server and serves multiple tenants. A tenant is a group of users who share a common access with specific privileges to the software instance. With a multitenant architecture, a software application is designed to provide every tenant a dedicated share of the instance including its data, configuration, user management, tenant individual functionality and non functional properties. It is to be noted that Multitenancy is different from running application on multiple instances[12].

5.6 SPL based Multitenant SaaS Application using SPL architecture

The group of researchers Mohammad Abu Matar et. al. developed a concept tool demonstrate using SPL in a multitenant SaaS application. To perform their experiment they developed a sample e-commerce application. Their experiment consisted of three major phases which includes:

1. SaaS kernel development
2. New Tenants Onboarding
3. Existing Tenants Customization

5.6.1 SaaS Kernel Development

This is the most important phase of the cloud architecture for Software Product Line based cloud application, the researchers developed Platform Independent Models(PIM) and then later on transformed to a Platform Specific Models (PSM) based on the desired target technologies. It is to be noted that in this kind of architecture, the kernel is the lowest level of layer which will interact with the application onboarding. And the kernel would be a part of all the future SaaS as well. So in this way the kernel instance serves as a seed for the all the multi tenant applications.[6]

5.6.2 New Tenants Onboarding

In this part of the architecture, the kernel modules are loaded for the Multitenant applications which have different features, for example if one application requires more CPU cycles and the other requires more network bandwidth as a part of their feature they are installed on the same kernel modules. Later on the PIMs of the different tenants is merged to generate a multitenant PSM, during this merging phase of the application onboarding the anomaly is detected and recorded in the report so that it can be added later in the phase.[11]

5.6.3 Existing Tenant Customization:

In this section the already boarded application is customized. It can be very well explained by a business requirement, for example, let’s say there was an e-commerce business and it required only 500 CPU cycles to process in the past but now the customers have increased and it not only requires more CPU cycles but also requires more bandwidth. So the customization part deals with change of configuration of already onboarded application. In most modern cloud architectures, this strategy is used. This above discussion can be well explained from figure 9.

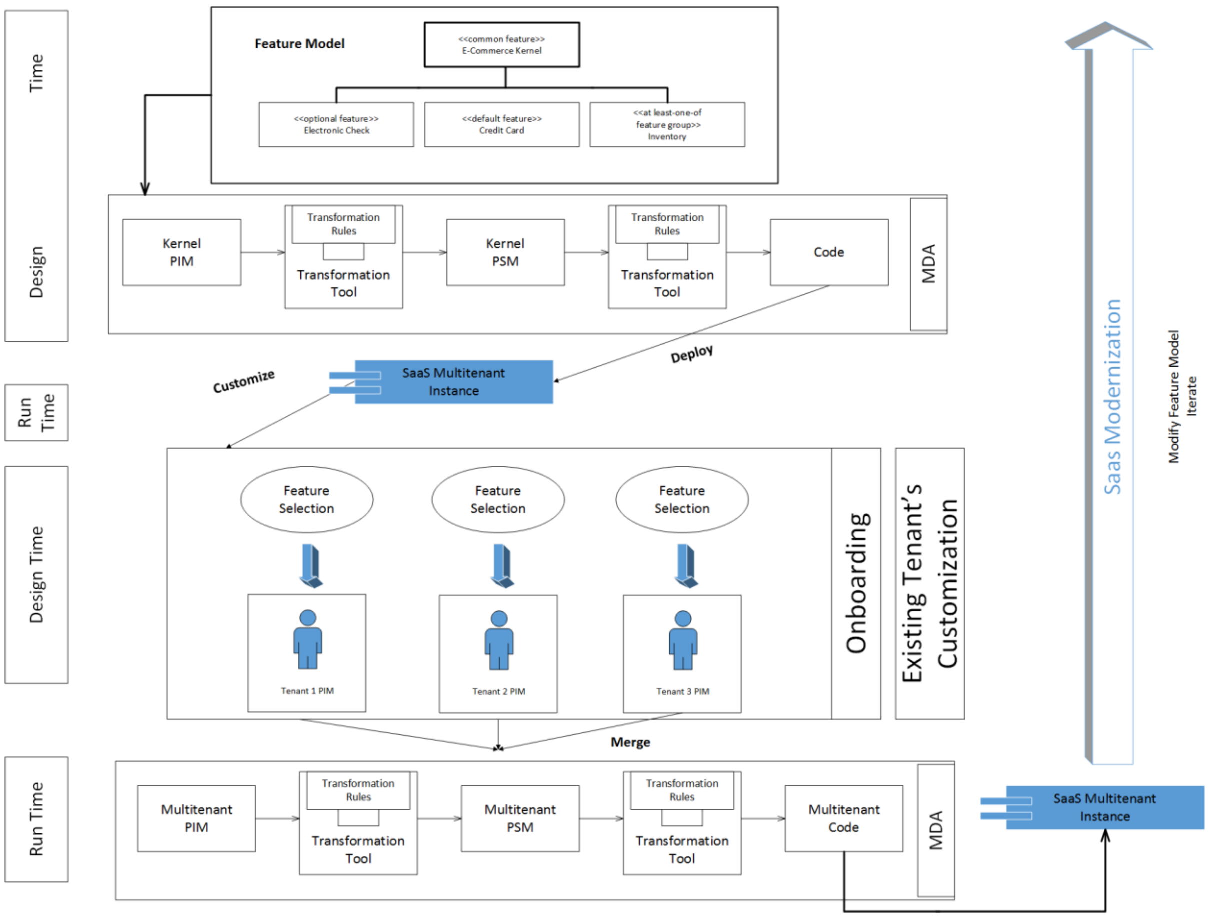


Fig. 9: Tenant evolution architecture concept model

The researchers developed a proof of concept tool in which they created a kernel module on SaaS and then onboarded the applications with different features. The experiment thus successfully proved that different applications can be onboarded using the same kernel instance. They even customized the MTA requirement and changed the resources allocated to the different applications.

6. Case Study - SaaS Application Customization Using SPL

In this part, a specific model of SaaS with SPL is brought to show how it works and enhances the multitenancy's using experience. SaaS is the short of Software-as-a-Service, which is a new structure of delivery. It means customer can customize their application on the internet[22]. This paper meanly analysis the possibility of implementing SPL(Software Product Line) into SaaS, in order to improve the customizability of SaaS applications. In traditional way, customers who want to use a application have to buy the specific license or hire their own technical people to development one. Whereas, based on SaaS, the only thing need to do for a customer, who want to use the application recourse, is paying for the application provider, then this application will be accessible for this customer. From this part we can see the issue customers always have different requests for the application they want, single provided application cannot fulfill every customer. In that situation, if the provider want the customers could share the application, it will require the flexibility.

There are two type of flexibility of applications. To take IS(Information System) as example[23], first one is Flexibility-to-Use, it means the application's capability of requests without the major change. Second one is Flexibility-to-Change which refer to the extensibility and the describes an application's extensibility and the works need to be done for a major change.

To compare the performance of two type of systems, here we discuss the two structures reliability of flexibility[24]. In the Single-tenancy model, sometimes the single-tenancy application need to be shared by multiple users, but this application is actually designed for one customer. In this case, the application's flexibility will depend on the main users requirements and compatibility. If the customer need any other changes after the adaptation is done, the only method is doing the major change. So the main flexibility of a Single-tenancy application is Flexibility-to-Change. Whereas, for the SaaS applications require higher Flexibility-to-Use. Because in this model, the application will no longer be designed for a single customer. So this application needs enough compatibility for every users in the group. Moreover, the SaaS application still have some Flexibility-to-Change, that is because the application cannot guarantee all the customers will be covered, so if customers require some functions which are out of boundary, providers can still make major change to fulfill the requirements.

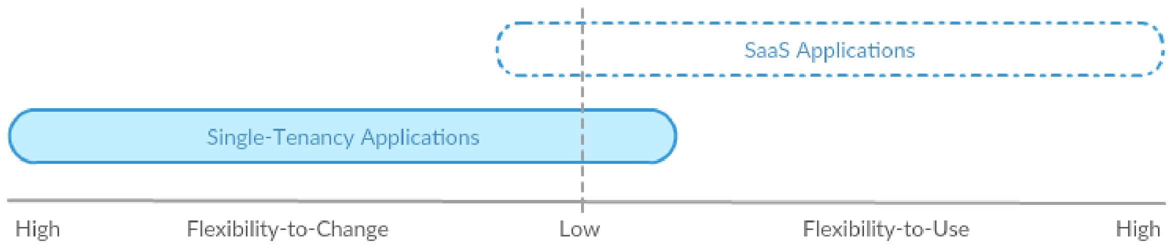


Fig. 10: Need for Flexibility

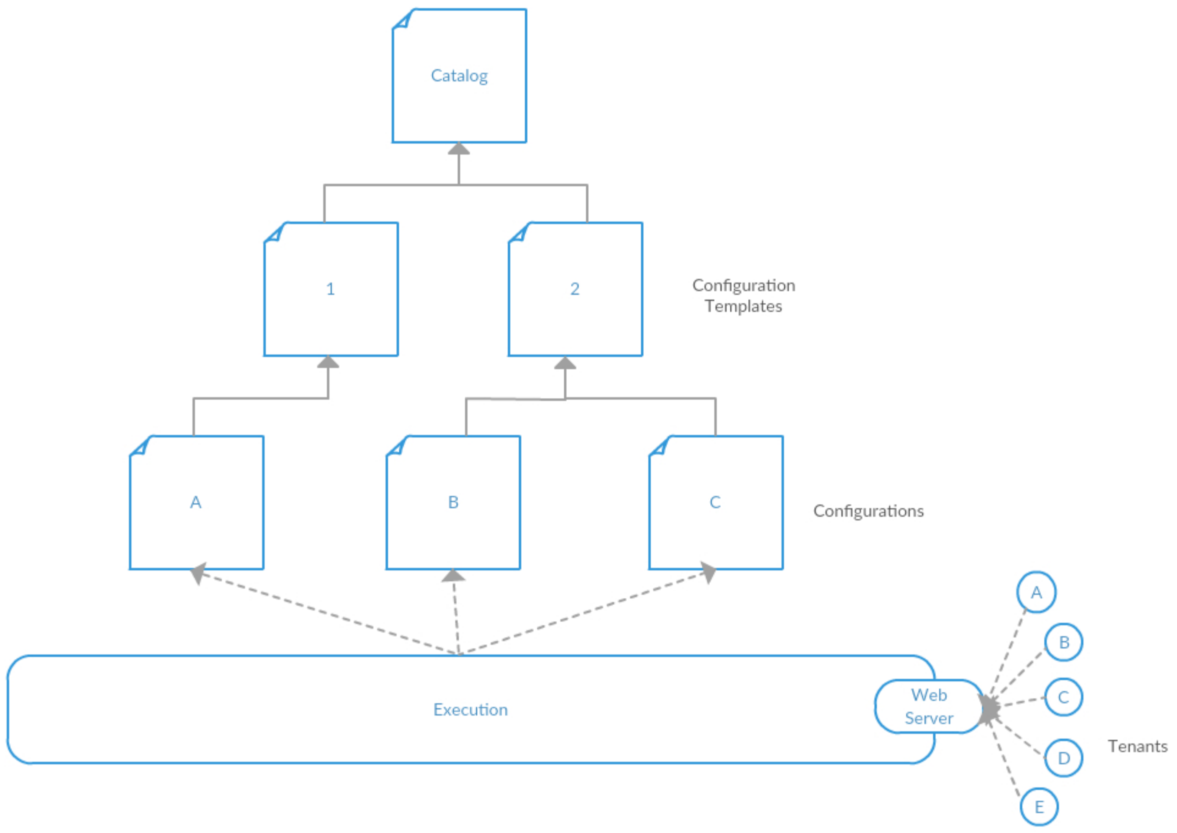


Fig. 11: Overview of the model

From the figure 11 (redrawn from the original paper) we can see the general structure of the new model. The tenants stand for customers, each of them could have multiple sub-points(Users). Every customer could achieve a application by communicating with Web server, the application is unique and customized by the customer. The server will allocate the components, which are needed by customers, to form the new application.

After having the structure of SaaS, next step is about the SPL. SPL is a kind of product line, which means manufacture the specific product to customer massively, the only difference is the products turn to be software. Because the products in the SPL are tailored for the customers, so it can be applied to the adaptation part of SaaS application to make it more efficient. And for reducing the problems, applying SPL needs to switch the product line's approach to flexibility-to-use[25].

The development of SPL can be separated into two aspects[4]: Domain Engineering and Application Engineering. Domain Engineering will create the necessary components of application, and also a Variability Model to monitor the state of the product line. Application Engineering will create the components which customer really needed as the customer's requirements. Thus, with this two Engineering, the new component of the application could be created anytime the customer needed. Then is the variability of the application. Because the customers need to build their own application from the components. So this part need a function to describe each components(help to form the application). Whereas, as we all know, the components of an application have very complicated relationships, so form a application is not a simply combination of components. Just like the product line, the SPL also have the ability to combine the different components with a complex rule. It will analysis the potential form of the application, so SPL will solve this problem easily. To make sure the application running well, when a custom start form a application, the used components need to bound with this custom, and store it in memory. Thus the customer could continue using this application. There are multiple technicals could be used from SPL to solve this problem, whereas based on some limitation, only 6 of them can be used in SaaS application. Those are[19]

1. Binary Replacement Directives
2. Binary Linker Directives
3. Infrastructure- Centered Architecture
4. Run-time Variant Components Specifications
5. Variant Component Implementations
6. Condition on Variable

Comparative Study

So far, we have gone through the papers deeply related to SPL architectures in addition to the basic concepts of software product line and cloud computing.

1. Towards Software product Lines Based Cloud Architectures
    a. A multiple-view modeling approach which deals with service-oriented cloud systems variability with regards to a combined manner.
    b. Incorporating for SPL concepts and variability analysis techniques to model variability of cloud systems.
2. Architecting Cloud Tools using Software Product Line Techniques: an Exploratory Study
    a. Applying SPL techniques to a software development process of practical cloud tools that control their own variabilities.
    b. Using feature modeling to capture the variabilities and synchronize it into the cloud architecture.
3. Applying Software Product Lines to create Customizable Software-as-a-Service Applications
    a. Making use of SPL techniques to implement customizable SaaS applications.
    b. Creating applications based on generic model designed by SPL techniques

Challenges in SPL with Cloud

Today cloud usage is increasing and many organizations are trying to use cloud to deploy their applications. But many of these organizations does not have much knowledge on cloud infrastructure and its configuration. Therefore obtaining the right balance of configuration will be difficult and might be time consuming work. So this leads to under or over utilization of cloud resources which will in turn bring the application down or cost overrun will happen. Mapping features with cloud configuration is a very difficult task since feature are high level concepts about the applications. But today we have many tools which will convert the selected features to the required cloud environment configurations. For example if there are two features whereas one requires network intensive nodes and the other requires processor intensive nodes the tool will define that. Another problem with cloud infrastructures is that it might be difficult to configure large features in the cloud.

One of the basic requirement of the cloud infrastructure is that it has to customizable according to our requirement. This is highly feasible with the SPL architecture techniques, there are some techniques in the SPL architecture to make this customizability highly possible. Several techniques in the Service Oriented architecture SOA and SPL has provision for dynamic reconfiguring of the configuration. These techniques can be adopted to the cloud configurable in multi tenant scenario.

Conclusion

Through this paper we learnt a comprehensive cloud architecture models, and we understood the background behind creating an SPL approach in creating cloud architecture. We learned about various study models behind creating Software Product lines based cloud architecture. The variability study model was common in most of the research papers that we used as a reference. By doing a thorough study of the papers we were able to comprehend how variability study helps build a common architecture. The common architecture consists of a common kernel in the multitenant application. The multitenants application then make use of the kernel resources as per the way they are programmed. This design model makes the architecture independent of specific requirement and thus creates a universal Software Product Line which can be used by other applications in the model. The paper that talks about SPL to software development process gives an insight on the various phases that lead to a concrete architecture of the cloud based application. In the end we would like to conclude that SPL based cloud architecture is heavily dependant upon studying the variability and merging them in one so that they can be used as product line to provide variety of services.

From the paper published by Professor Mohammad Matar et. al we quote their direct words and belief [6]:

That their approach has several benefits:

- Meta-modeling of the cloud computing ecosystem.
- A multiple-view meta-model for describing cloud-based software product lines.
- Treatment of cloud systems variability concerns in a unified, systematic, platform-independent manner.
- The use of established SPL Feature Modeling to manage variability in the Cloud.

References

[1] Overview and Benefits of Software product lines: http://www.sei.cmu.edu/productlines/

[2] NIST: Definition of cloud computing and recommended standards prescribed by national institute of standards and technology: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[3] Cloud computing architecture overview and definition of IaaS, PaaS, SaaS, http://cloudacademy.com/blog/cloud-computing-architecture-an-overview/

[4] Stefan T. Ruehi, Urs Andelfinger, “Applying Software product Lines to create Customizable Software-as-a-Service Applications,” SPLC’ 11 Proceeding of the 15th International Software Product Line Conference, Volume 2 Article No. 16, 2011.

[5] Explaination on Software Product Lines and various components associated with it: https://en.wikipedia.org/wiki/Software_product_line

[6] Towards SPL based cloud architecture: Mohammed Abu Matar et. al. IEEE 2014 International conference on cloud computing.

[7] Abu-Matar, M., Gomaa, H.,"An Automated Framework for Variability Management of Service-Oriented Software Product Lines," Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on, 25-28 March 2013.

[8] K. Schmid and A. Rummler, “Cloud-based software product lines,” in Proceedings of the 16th International Software Product Line Conference - Volume 2, New York, NY, USA, 2012, pp. 164–170.

[9] Service Component Architecture (SCA), http://oasis-opencsa.org/sca, May 2013.

[10] A. Kleppe, J. Warmer, and W. Bast, MDA Explained: The Model Driven Architecture(TM): Practice and Promise, 1st ed. Addison-Wesley Professional, 2003.

[11] Schmid, K.; Rummler, A., "Cloud-based Software Product Lines," Software Product Line Conference (SPLC), 2012 16th International, vol. 2, pp.164-170, 22-26 Aug. 2012.

[12] Abu-Matar, M.; Gomaa, H., "Variability Modeling for Service Oriented Product Line Architectures," Software Product Line Conference(SPLC 2011), , 22-26 Aug. 2011.

[13] P. Clements and L. Northrop, Software Product Lines: Practices and Patterns, Addison-Wesley, 2001.

[14] S.G. Decker and J. Dager, “Software Product Lines Beyond Software Development,” Proc. 11th Int’l Software Product Line Conf. (SPLC 07), IEEE CS Press, 2007, pp. 275–280.

[15] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, \FORM: A feature-oriented reuse method with domain-specific reference architectures," Annals of Software Engineering, vol. 5, no. 1, pp. 143-168, 1998.

[16] K. Pohl, G. B• ockle, and F. Van Der Linden, Software product line engineering. Springer, 2005, vol. 10.

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

[18] N. Medvidovic and R. N. Taylor, A classification and comparison framework for software architecture description languages," Software Engineering, IEEE Transactions on, vol. 26, no. 1, pp. 70-93, 2000.

[19] M. Svahnberg, J. van Gurp, and J. Bosch. A taxonomy of variability realization techniques: Research articles. Software—Practice & Experience, 35:705–754, July 2005. ACM ID: 1070905.

[20] C. Gacek and M. Anastasopoules, Implementing product line variabilities," in Proceedings of the Symposium on Software Reusability: Putting Software Reuse in Context. NY, USA: ACM, 2001, pp. 109-117.
[21] F. J. van der Linden, K. Schmid, and E. Rommes, Software product lines in action. Springer, 2007.
[22] W. Sun, X. Zhang, C. J. Guo, P. Sun, and H. Su. Software as a service: Configuration and customization perspectives. In Services Part II, IEEE Congress on, volume 0, pages 18–25, Los Alamitos, CA, USA, 2008. IEEE Computer Society.
[23] J. Gebauer and F. Schober. Information system flexibility and the cost efficiency of business processes. Journal of the Association for Information Systems, 7(3):122–147, 2006.
[24] Nitu. Configurability in SaaS (software as a service) applications. In Proceedings of the 2nd India software engineering conference, ISEC ’09, page 19–26, New York, NY, USA, 2009. ACM. ACM ID: 1506221.
[25] K. Pohl, G. Bo ̈ckle, and F. J. van der Linden. Software Product Line Engineering: Foundations, Principles and Techniques. Springer, 1st edition. edition, Nov. 2010.