Orthogonal Variability Modeling to Support Multi-Cloud Application Configuration

Pooyan Jamshidi, Claus Pahl

IC4 – the Irish Centre for Cloud Computing and Commerce,
Dublin City University, Dublin, Ireland

{pooyan.jamshidi,claus.pahl}@computing.dcu.ie

Abstract. Cloud service providers benefit from a vast majority of customers due to variability and making profit from commonalities between the cloud services that they provide. Recently, application configuration dimensions has been increased dramatically due to multi-tenant, multi-device and multi-cloud paradigm. This challenges the configuration and customization of cloud-based software that are typically offered as a service due to the intrinsic variability. In this paper, we present a model-driven approach based on variability models originating from the software product line community to handle such multi-dimensional variability in the cloud. We exploit orthogonal variability models to systematically manage and create tenant-specific configuration and customizations. We also demonstrate how such variability models can be utilized to take into account the already deployed application parts to enable harmonized deployments for new tenants in a multi-cloud setting. The approach considers application functional and non-functional requirements to provide a set of valid multi-cloud configurations. We illustrate our approach through a case study.

Keywords: Multi-cloud, Variability, Multi-tenancy, Cloud Architecture.

1 Introduction

In the delivery model of software systems, a frequently studied shift can be observed [1] in which software products are no longer delivered to customers as packages or deployed in-house (on-premises), however, they are deployed at a central or (potentially distributed) location(s) and offered to customers online (Software-as-a-Service; SaaS). In on-premise deployments, application users use only one instance of a software application that is developed in a software house and deployed on-premises. Obviously, such application can be tailored to behave in accordance with specific customer requirements in case the standard product functionalities do not perfectly fit with the very specific customer‘s requirements (in terms of business process support or particular non-functional requirements) in place [2]. However, this approach will not be profitable if the variations become unmanageable. To make this more cost-beneficial and yet to satisfy particular customer needs, a novel approach has been proposed many years ago within which different products, based on software product core containing general requirements shared by all customers, can be generated in a software product line [3].

The concept of software variability was first studied in the software product lines community [4]. Traditionally, variability only considers on-premise single-tenant software. This approach follows the idea that software products containing specific features have to be built before shipping the software (i.e., design-time binding). Software products that are delivered through online channels, however, only profits from runtime binding since it would be undesirable to restart or redeploy a version whenever changes are made (such case is a norm in design-time binding, for example). In software products, which are accessible online, the higher the configurability of a product is, the more variable a product would be. A variable product aims to provide customers with a set of options using a single code base in such a way that each tenant is able to have a unique configuration. Since the software instance is now deployed in one location, multiple customers can potentially use the same instance of the software. Sharing a software instance with multiple users, however, makes it impossible to have customized products for specific customers. In other words, multi-tenant software is able to fulfil all different customer requirements, while still taking advantage of shared resources [5]. In this setting, tenants are customers using the application and usually consisting of multiple users within the same organizational context or common interests [6].

The work reported in [7] use the concept of external runtime variability in software product line to represent the variability management in SaaS context. They refer to such type of variability as customer-driven variability. Such usage of external variability to represent tenant-based variability is also shown in several other research work, e.g., [5, 8–10]. One of the commonalities within such varieties of work is that they use a sort of variability model to represent very specific needs of tenants to make the online software product a configurable entity to allow for the varying requirements. However, with the new type of development in the accessibility options as well as deployment variability in the cloud, the application configuration and customization becomes a complex and unmanageable process due to the large variability in the solution space. The newly introduced dimensions are orthogonal to the tenant-specific configuration and, in this paper, we propose to use a multi-dimensional variability management to address the newly raised concerns that we discuss in the following.

Multi-cloud configuration. Multi-cloud deployment [6, 11] is particularly effective in dealing with the situations where users are widely distributed around multiple data centers, country regulations limit options for storing data in specific data centers, and circumstances where public clouds are used jointly with on-premises resources. The main difficulty is to deal with the wide range of deployment options at different layers on different cloud platforms [12]. Such dimensions are certainly orthogonal to the application architecture and make a large solution space for the deployment architecture. Consequently, there is a need to enable multi-cloud deployment specific configuration.

Multi-device configuration. With the raise of smartphones and devices in one hand and heterogeneity in the platforms, tenants are required to access to the cloud-based services on specific devices. Consequently, there is a need to enable multi-device and cross platform configuration, which is also orthogonal to both the application and deployment architectures.

To address these challenges, we propose $3OVM$, a model that integrates three variability models each addressing specific challenges regarding tenant-specific, multi-device and multi-cloud configuration. Cloud application variability can thus be described using $3OVM$. We use a video processing application as a running example to describe the approach. This application contains more than 10,000 different configurations. However, the approach proposed here is not specific to this application and can cover different cloud-enabled application domains that can benefit from multi-cloud deployment, such as process-aware applications [13].

2 Research Challenges

With the rise of multi-tenant cloud-based applications, the dimensions for configuring the software that is usually offered as a service has been also increased dramatically. The functionality and quality that individual tenants need from a software application are typically different from each other. As a consequence, in order to attract enough customers, cloud service providers are required [5, 7, 9, 10, 12]: **(i)** To cater for the varying requirements of potential tenants by providing tenant-specific configurations. **(ii)** To make sure that they can handle the varieties of the deployment options in terms of, for example, infrastructure offerings. **(iii)** To configure the tenant-specific configuration by considering the variability of the devices that each tenant needs to interact.

For example, SaaS applications allow users to customize the captions used in the application as well as adding and modifying business processes implemented in the systems. Such tenant-specific adaptations of a SaaS application affect all layers of the application. From functional requirements to business processes and all the way down to database schemas. Tenants do not only have various requirements with respect to functional features, but also require different non-functional requirements. While some tenants want an application to be highly available, other tenants are not so much interested in high-availability, but care more about let's say performance. Traditionally, this was handled through different pre-configured software packages with different prices. However, cloud allows a more robust delivery models in which software is licensed on a subscription basis and is hosted on cloud platforms by independent vendors or service providers. Therefore, cloud service providers face the following challenges when it comes to the configuration and customizations of their software applications:

Challenge1. *Tenant-specific configuration and customizations*. Cloud applications are typically multi-tenant and each tenant require its own specific functional and non-functional requirements.

Challenge2. *Multi-cloud configuration*. The main difficulty is to deal with the wide range of deployment options at different layers on different cloud platforms. Such dimensions are certainly orthogonal to the application architecture and make a large solution space for the deployment architecture.

Challenge3. *Multi-device and cross-platform configuration*. With the rise of smartphones and devices on the one hand and heterogeneity in the platforms that they support, tenants are required to access to the cloud-based services on specific devices.

We highlight these challenges through a case study in the next section.

3 Running Example

To highlight the challenges and to exemplify our approach, we introduce a video-processing application [10], which is a cloud-enabled system deployed on a multi-cloud environment. In this example, the video processing software is offered as a service, which has the capability to be customized for different tenants based on their required functions, quality and end-point devices. The application comprises a number of components as depicted in **Fig. 1**. The system is illustrated in three different architectural views, i.e., application architecture, deployment architecture and accessibility devices.

Tenants can choose different subsets of the components, considering that the architectural constraints are not violated. One of the constraint is that the configuration must include Player (VP), Decoder (Dec) and Data-Provider (DP) components, which form the core of the video processing (this represents the commonalities among the derived products). The functional variability of the video-processing application is as follows: Video-Manager (VM) component, which offers a graphical UI to add and remove videos. Icon component, which injects a tenant-specific logo into the videos before they are played. Subtitle (Sub) component, which introduces subtitles as overlays on videos.

The StreamProcessor (SP) is parent of the Decoder, Icon and Subtitle components. This abstraction enables the tenant to choose a combination of the three inheriting components. Note that in this setting different sub-architecture of the system can potentially be deployed on different cloud platforms. This is because all deployment of the same parts provide the same functionality, but differ in their non-functional properties. For example, the VideoPlayer-StreamProcessor binding is realized by the pipes and filters style on Windows Azure and Amazon AWS. Both deployments facilitate playing a video, but Azure deployment offers a different rendering resolution and performance than AWS does. Furthermore, the accessibility on the two deployments is different. For example, AWS deployment does not provide refrigerator end accessibility.

For each tenant an own configuration/deployment of the application can be created. For our example, we assume three tenants: the first tenant (T1) does not use any optional component, whereas the second tenant (T2) has decided to use the Subtitle component to enhance the application. The third tenant (T3) has decided to have all the functionalities. Besides the functionality that each tenant is required, they require their own specific non-functional requirements (NFRs) (c.f. Figure 2), which characterize how the provided functionality can be fulfilled. Besides, each tenant requires to access the functions in different devices, see **Table 1**.

Table 1. Tenants configuration and Requirements.

Tenants	**Components**	**NFRs**	**Accessibility**
T1	VP, Dec, DP	Bandwidth: 1000MB/s Availability: Standard, Latency: 1s	PC, Mobile (Android), Mobile (iOS), Refrigerator
T2	VP, Dec, Sub, DP	Bandwidth: 1000MB/s Availability: Standard, Latency: 0.1s	All
T3	VP, Dec, Sub, Icon, DP, VM	Bandwidth: 10MB/s Availability: Super, Latency: 0.01s	PC (Mac OS), Car

The architectural style of the video-processing application is pipes and filters. Such architectural style when realized on cloud platforms allows for the on-demand provisioning of multi-part job processing. It can be used for instantaneous or delayed deployment of a heterogeneous, scalable "grid" of worker nodes that can quickly crunch through large batch processing tasks in parallel [6, 14–16]. Numerous batch-oriented applications are in place that can leverage such on-demand processing including video transcoding. The video-processing application behaves as follows:

1. Users interact with the end-points, which is deployed on a cloud platform. This component controls the process of video management and playing.
2. Raw video data is transformed to a cloud storage, a highly available and persistent data store. The transcoding tasks are inserted by an elastic queue.
3. Worker nodes are cloud instances that can be scaled. Worker nodes pick up tasks from the input queue and perform single tasks that are part of the list of batch processing steps. Interim results from worker nodes can be stored in a storage.
4. Progress information and statistics are stored on the storage as well. This component can be either a cloud storage or a relational database.

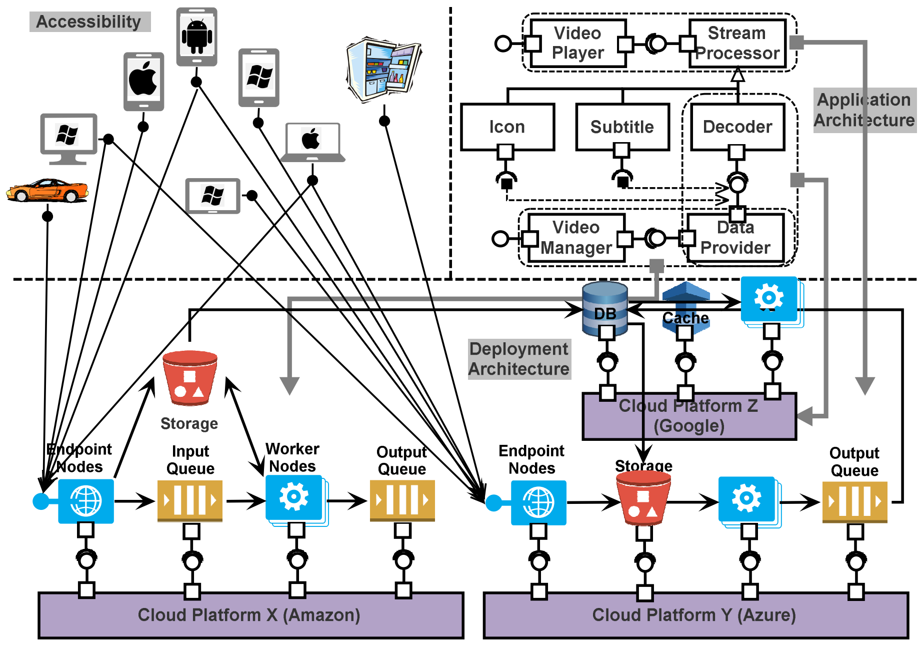

Fig. 1. Application and deployment architecture of video processing system.

The video-processing application is a multi-tenant SaaS application and as we indicated in **Table 1**, each tenant requires its own specific functional and non-functional features accessible on specific platform. The main challenge here is how to deal with the wide range of deployment options and this situation becomes more challenging when we consider multi-cloud. In this work, we address the complexity of multi-cloud application configuration through a model that we introduce in the next section.

4 Approach

In order to build highly scalable applications, *multi-cloud deployment* is appropriate [11, 17]. The objective of this work is to provide a model-based approach that facilitates tenant-specific configuration and customizations for applications that run on multiple independent clouds.

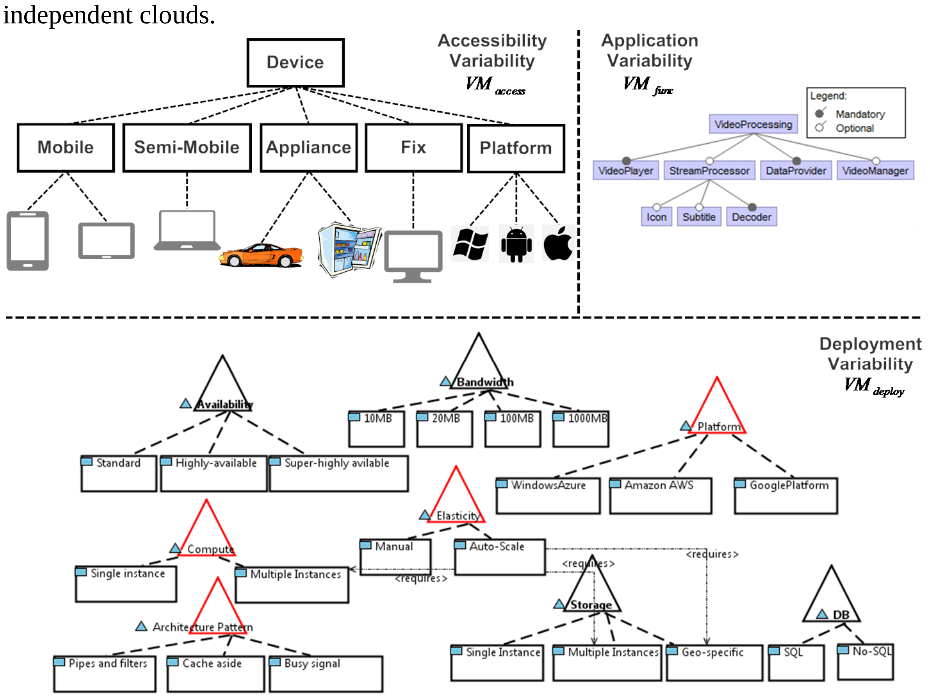

Fig. 2. Orthogonal variability model showing the accessibility-driven, application-driven and deployment-driven variability.

Multi-cloud denotes the usage of multiple, independent clouds by a client or a service. A multi-cloud environment is capable of processing user demand and distributing work to resources deployed across multiple clouds [18]. A multi-cloud is different from *federation* where, a set of cloud providers voluntarily interconnect their infrastructures to allow sharing of resources among each other [18]. *Hybrid deployment* can be considered as a special case of multi-cloud where an application is deployed in both on premise infrastructure as well as cloud platform(s). Such type of deployment model is essential in cases where critical data need to be kept in house in corporate data centers. We have reviewed different application types and specific requirements of them that necessitate multi-cloud deployment – see the supplementary materials here [15].

In a multi-cloud configuration perspective, parts of the application can be deployed at either PaaS, IaaS or both levels [17, 19], see **Fig. 1**. The wide range of cloud providers likely to host the application makes the choice difficult. To fit these requirements and dimensions, we find 15 possibilities in terms of patterns, reported in [15]. The key reasons behind such multi-cloud migration are as follows:

- Users are widely distributed where they are located around multiple data centers.
- Country regulations limit options for storing data in specific data centers, e.g., EU.
- Circumstances where public clouds are used jointly with on-premises resources.
- Cloud-based application must be resilient to the loss of a single data center.

To address the challenges identified in this paper, we define an orthogonal variability model used to **(i)** capture cloud providers visible options for tenants-specific deployment and **(ii)** enable tenants to configure functional and non-functional aspects of the application, **(iii)** enable tenants to choose their preferences of accessibility options.

In this work, we combine three different variability models at different levels of abstractions regarding the three above-mentioned concerns into a single model (i.e., $3OVM$) that gives strong support to all tenants involved in cloud deployment. As illustrated in **Fig. 2**, the approach allows users to **(i)** define a functional specification of the system through choosing the options in the application variability model (VM_{func}). Also, it allows users **(ii)** to select alternatives for realizing and deploying the application on a (multi-)cloud environment as well as selecting the non-functional preferences of the system in the deployment variability model (VM_{deploy}). Such aspects themselves affect some internal non-visible aspects of the system (red triangles in **Fig. 2**), which are only visible for the development team of the cloud-based application. This will be realized by combining several valid cloud configurations to fit the requirements. **(iii)** To choose the accessibility options that are required by the users comprising multi-device and multi-platform capabilities in the accessibility variability model (VM_{access}).

Therefore, $3OVM$ consists of three variability models at different levels regarding different concerns of multi-cloud application deployments. VM_{func} is a fully fledged variability model that represents both functional commonalities and variabilities of the cloud-based software products. However, VM_{deploy} and VM_{access} represent only variabilities that determines the non-functional aspects of the cloud-based products. They represent, in other words, a reference point to where different variants regarding the deployment options or accessibility can be attached. The variants manifest a concrete variability in terms of deployment or accessibility. In this model, all variation points in VM_{deploy} and VM_{access} are related to at least one functional variant in VM_{func}, and all variations in VM_{func} are related to at least one variation point in either VM_{deploy} or VM_{access}. This reduces the complexity of the variability model and therefore enhances the readability of the model facilitating a robust application customization.

For defining VM_{func} and VM_{access}, we use feature model defined in [20], while for specifying VM_{deploy}, we employ the OVM (Orthogonal Variability Model) introduced in [3]. The reason behind such choice for VM_{deploy} is that the OVMs are smaller and less complex since they only model variability and not the commonalities. This is useful in the context of multi-cloud environments since for modeling the deployment space we only need to consider different variability that each platform may offer and not thinking about their commonalities.

The $3OVM$ model distinguishes between different roles: application developers (Dev), cloud experts (Ops) and the tenants (Ten). Application developers provides the functional variability and commonality points of the system, resulting in the corresponding variability model (i.e., VM_{func}). Cloud experts are involved in the platform specific descriptions. They describe cloud platform variability and commonality points, thus providing the corresponding variability model (i.e., VM_{deploy}) to the architecture. Cloud experts are also responsible for providing the accessibility variability model (i.e., VM_{access}). Tenants are all user groups involved in externally visible option (bold triangles in **Fig. 2**) selection through such orthogonal variability models. Using such an approach only requires having necessary knowledge to properly configure the cloud application and to properly cooperate to develop such knowledge to the system. We will describe the usage process of our approach in detail in the next section.

5 Multi-Cloud Deployment Support

Having specified the variability of cloud applications, the three variability models can be used to further support the customization and deployment lifecycle. In this section, we describe a process to perform such deployment by utilizing $3OVM$ via the example.

In order to customize a cloud-based application, **(i)** *tenants need to decide which variants*, whether they are functional or non-functional or accessibility aspects, *should become part of their application*. Therefore, tenants need to have the capability to choose among the potential configuration options in order to bind the variability of the cloud application. In the product line engineering, several approaches are existed to realize such a customization support [7].

After binding the tenant-specific variability of a cloud-based application, **(ii)** *the deployment actions should be accomplished* in order to prepare the application for users belonging to that tenant. The deployment actions depend on the binding of the deployment variability, which itself depends on the cloud platform and the information about already bounded variability, i.e. deployed features of the application. Therefore, to bind the deployment variability, the cloud platform needs to consider the tenant-specific variability as well as the binding information about the parts of the application that have already been deployed for other tenants.

Some variation points must be bound because of dependencies to variants that have been chosen by the binding of the tenant-specific variability. For each open variation point, the cloud service provider is aware of the possible variants because it already knows the variability model via $3OVM$. Now the most appropriate variant for the provider must be chosen. This can be done through **(iii)** *annotating individual variants* with, for example, a cost parameter and using optimization algorithms to find the least expensive variant combination. This, however, is beyond the scope of this paper.

The deployment actions do not only depend on the status but also on the services that are used for a particular tenant. For the services that are used in a single instance mode, the cloud service provider must make sure that enough resources are available to run the new tenant on the instance of the service that is already deployed. In multi-cloud setting, this however can manage through live migration from one platform to another one if the resources are not sufficient in one particular platform. For the services in multiple instances mode, the appropriate infrastructure must be provisioned. In sum, the purpose of this step is **(iv)** *to perform the required reconfiguration* to adjust the platform for proper deployment of the application.

We now describe the above–mentioned customization and deployment process through our running example. **Table 2** shows three possible configurations for the video processing application. The table shows the bounded alternatives for each variation point (here we only concentrated on VM_{deploy}). Config. 1 and Config. 2 are the same in terms of external variability points, since every variation point is bound with the same variant. Note that they differ in the platform variation point; however, platform variation point is internal and is not visible to the tenants. Therefore, these two configurations are externally equivalent and the same in the view of tenants. The result of such situation is that two tenants that bind all external variability points in a similar way can end up with two different solutions, as one solution for example is deployed on a cloud platform while the other is deployed on another platform. Such configuration, although might have the same functional behaviour, they may show different non-functional behaviour due to heterogeneous cloud platforms.

Table 2. Exemplary configurations of the video processing system.

Variation point		**Config. 1**	**Config. 2**	**Config. 3**
External	**Availability**	Standard	Standard	Super available
	Bandwidth	1000	1000	10
	Storage	Multiple-instance	Multiple-instance	Geo-specific, single instance
	DB	SQL	SQL	No-SQL
Internal	**Platform**	Azure	AWS	Azure/AWS/Google
	Compute	Multiple-instance	Multiple-instance	Multiple-instance
	Elasticity	Auto-scale	Auto-scale	Auto-scale
	Pattern	Pipes and filters	Pipes and filters	Cache-aside, Pipes and filters

The deployment for tenant 3 (Config. 3) must provision resources for the storage service as this tenant chose to store the data in specific location and cannot share its data with the other tenants due to security concerns. Therefore, the service must also be deployed in single instances mode, as privacy is important to that tenant. As the tenant also selected the option of super availability, the application is deployed on multiple cloud platforms. Now a situation arises where tenant 3 requires the resources on Google cloud platform in which the functionality of the video processing application is not deployed since the other two tenants are on Azure and AWS platforms (cf. **Fig. 1**).

Once a tenant decides to unsubscribe from a cloud application, it must be undeployed from the system [7]. This might be as simple as removing a line in a configuration setting. However, in case a tenant has used services that are deployed on multiple cloud platforms, these services must be undeployed from each specific platform by issuing platform specific undeployment commands. Nevertheless, it is important to know which combination of functionality, accessibility and deployment has been used by a tenant to perform the necessary steps to undeploy the services.

In addition to provisioning and de-provisioning scripts, "tenant transfer" scripts can be generated that describe how a tenant is transferred from one configuration to another one or from one specific platform to another platform or hybrid or even multiple platforms.

6 Related Work

Multi-cloud application configuration. Quinton et al. [21] present a model-driven approach based on feature models and ontology to handle heterogeneity in cloud variability and managing cloud configurations. The approach considers technical as well as non-functional requirements to provide a set of valid configurations. Their focus is mostly on managing the heterogeneity in multi-cloud environments via a mapping from an ontology model to platform specific feature models. This work is the closest existing work to our approach. However, our main concern is to manage functional and accessibility aspects in accordance with multi-cloud deployment aspects in a homogenous variability model that different roles in developments and operations (known as DevOps) can cooperate to build up the model and provide a unified model that tenants can choose their preferences through it without requiring specific technical knowledge.

Sampaio et al. [22] propose an approach that facilitates modeling, deployment and configuration of software applications over multiple heterogeneous IaaS clouds. In this approach, the application to be deployed is specified using open standards to be run in VMs while our approach is based on the variability model that enable tenants to choose from three different aspects of the system considering its tenant-specific requirements.

Brandtzæg et al. [23] propose a component-based approach that leverages the existing deployment descriptors into a domain-specific language (DSL). The DSL is used to model the deployment and an interpreter is provided to identify which resources have to be used in the platform to fulfill requirements. Although this work facilitates a semi-automated deployment, they do not consider the cloud offer heterogeneity.

Paraiso et al. [24] present a multi-cloud PaaS infrastructure deployed on existing IaaS/PaaS. This infrastructure is based on an open service model. Contrarily to our approach, they do not need to consider the capability to configure the multi-cloud platform since they use the same service model for both SaaS and PaaS.

Multi-tenant application configuration. Mietzner et al. [7] show how variability modeling techniques from software product line can support cloud service providers to manage the variability of cloud-based applications and tenant specific configurations. They propose using explicit variability models to systematically derive customization and deployment information for individual tenants. Gaddar et al. [9] introduce the new concept of Variability as a Service (VaaS) model to relieve cloud providers from developing expensive variability models by decreasing the variability management complexity.

More recently, Quinton et al. [8] propose an automated approach to face the configuration of cloud-based applications. Their approach automates the deployment of such configurations through the generation of executable scripts. Schroeter et al. [10] identify requirements for runtime architecture addressing the individual interests of tenants in multi-tenant architectures. They show how dynamic architectures can be extended for the development of multi-tenant applications. This work, as opposed to the other approaches, concentrates on the variability at the architecture level.

These above mentioned work although inspiring, they are only applicable for single platform SaaS based applications as opposed to our approach, which targeted multi-cloud environments. In fact, in terms of technical contribution, all of the multi-cloud configuration management approaches, are an extension of [7].

7 Conclusion and Outlook

We presented how orthogonal variability models that we borrowed from software product line engineering, can be used to model variability in 3 different yet important aspects of cloud applications in the multi-cloud environments. We have applied the concepts of configuration management from software product line engineering to the problem of deployment support for multi-cloud applications and have demonstrated the benefits of our approach by means of a case study. The key benefit of the proposed model is to manage different interrelated deployment aspects through a unified model consists of models at different levels of abstractions.

In our future work, we plan to automate this approach by developing a tool that enables the creation of the $3OVM$ variability models facilitating the automated deployment of multi-cloud application at runtime. We plan to employ the approach in the context of a multi-cloud runtime adaptation mechanism, a similar approach as have been pursued in projects like MODAClouds [19], CloudMF [25], mOSAIC [26] and OPTIMIS [27], however, targeting different concerns: (**i**) uncertainty handling in elastic systems, (**ii**) multi-cloud auto-scaling, (**iii**) auto-scaling in data-intensive applications and (**iv**) the application of control theory in auto-scaling [28].

Acknowledgments. The research work described in this paper was supported by the Irish Centre for Cloud Computing and Commerce (IC4), an Irish national Technology Centre funded by Enterprise Ireland and the Irish Industrial Development Authority.

References

1. Jamshidi, P., Ahmad, A., Pahl, C.: Cloud Migration Research: A Systematic Review. IEEE Trans. Cloud Comput. 1, 142–157 (2013).
2. Sun, W., Zhang, X., Guo, C.J., Sun, P., Su, H.: Software as a Service: Configuration and Customization Perspectives. 2008 IEEE Congress on Services Part II (services-2 2008). pp. 18–25. IEEE (2008).
3. Pohl, K., Böckle, G., Linden, F. Van Der: Software product line engineering. (2005).
4. Svahnberg, M., van Gurp, J., Bosch, J.: A taxonomy of variability realization techniques. Softw. Pract. Exp. 35, 705–754 (2005).
5. Kabbedijk, J., Jansen, S.: Variability in multi-tenant environments: architectural design patterns from industry. Adv. Concept. Model. Recent Dev. New Dir. (2011).
6. Wilder, B.: Cloud Architecture Patterns: Using Microsoft Azure. (2012).
7. Mietzner, R., Metzger, A., Leymann, F., Pohl, K.: Variability modeling to support customization and deployment of multi-tenant-aware Software as a Service applications. 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems. IEEE (2009).
8. Quinton, C., Romero, D., Duchien, L.: Automated Selection and Configuration of Cloud Environments Using Software Product Lines Principles. IEEE CLOUD 2014. (2014).
9. Ghaddar, A., Tamzalit, D., Assaf, A., Bitar, A.: Variability as a service: outsourcing variability management in multi-tenant saas applications. Adv. Inf. Syst. Eng. (2012).
10. Schroeter, J., Cech, S., Götz, S., Wilke, C., Aßmann, U.: Towards modeling a variable architecture for multi-tenant SaaS-applications. Proceedings of the Sixth International

Workshop on Variability Modeling of Software-Intensive Systems - VaMoS '12. pp. 111–120. ACM Press, New York, New York, USA (2012).

11. Petcu, D.: Multi-Cloud. Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds - MultiCloud '13. p. 1. ACM Press, New York, New York, USA (2013).
12. Koetter, F., Kochanowski, M., Renner, T., Fehling, C., Leymann, F.: Unifying Compliance Management in Adaptive Environments through Variability Descriptors. 2013 IEEE 6th International Conference on Service-Oriented Computing and Applications. IEEE (2013).
13. Jrad, F., Tao, J., Streit, A.: A broker-based framework for multi-cloud workflows. Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds - MultiCloud '13. p. 61. ACM Press, New York, New York, USA (2013).
14. Homer, A., Sharp, J., Brader, L., Narumoto, M., Swanson, T.: Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications. Microsoft (2014).
15. Jamshidi, P., Pahl, C.: Cloud Migration Patterns - Supplementary Materials, http://www.computing.dcu.ie/~pjamshidi/Materials/CMP.html.
16. Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P.: Cloud Computing Patterns. Springer Vienna, Vienna (2014).
17. Petcu, D., et al.: Experiences in building a mOSAIC of clouds. J. Cloud Comput. Adv. Syst. Appl. 2, 12 (2013).
18. Grozev, N., Buyya, R.: Inter-Cloud architectures and application brokering: taxonomy and survey. Softw. Pract. Exp. 44, 369–390 (2014).
19. Nitto, E. Di, Silva, M.A.A. da, Ardagna, D., Casale, G., Craciun, C.D., Ferry, N., Muntes, V., Solberg, A.: Supporting the Development and Operation of Multi-cloud Applications: The MODAClouds Approach. 2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. pp. 417–423. IEEE (2013).
20. Kang, K., Lee, J., Donohoe, P.: Feature-oriented product line engineering. IEEE Softw. (2002).
21. Quinton, C., Haderer, N., Rouvoy, R., Duchien, L.: Towards multi-cloud configurations using feature models and ontologies. Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds - MultiCloud '13. p. 21. ACM Press, New York, New York, USA (2013).
22. Sampaio, A., Mendonça, N.: Uni4Cloud: an approach based on open standards for deployment and management of multi-cloud applications. Proceeding of the 2nd international workshop on Software engineering for cloud computing - SECLOUD '11. p. 15. ACM Press, New York, New York, USA (2011).
23. Brandtzæg, E., Mohagheghi, P., Mosser, S.: Towards a domain-specific language to deploy applications in the clouds. CLOUD Comput. 2012. (2012).
24. Paraiso, F., et al.: A Federated Multi-cloud PaaS Infrastructure. 2012 IEEE Fifth International Conference on Cloud Computing. pp. 392–399. IEEE (2012).
25. Ferry, N., Chauvel, F., Rossini, A., Morin, B., Solberg, A.: Managing multi-cloud systems with CloudMF. Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies - NordiCloud '13. ACM Press, New York, New York, USA (2013).
26. mOSAIC EU project, http://www.mosaic-project.eu/.
27. Ferrer, A.J., et al.: OPTIMIS: A holistic approach to cloud service provisioning. Futur. Gener. Comput. Syst. 28, 66–77 (2012).
28. Jamshidi, P., Ahmad, A., Pahl, C.: Autonomic resource provisioning for cloud-based software. Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems - SEAMS 2014. pp. 95–104. ACM Press, New York, New York, USA (2014).