Towards Generating Multi-Tenant Applications

Julia Schroeter

Institute for Software- and Multimedia-Technology
Technische Universität Dresden
D-01062, Dresden, Germany
julia.schroeter@tu-dresden.de

Abstract. A multi-tenant application is a software delivery model in a Software as a Service (SaaS) context, where an application is deployed once and serves multiple customers (tenants). Every tenant has its own functional configuration of the application, but shares resources like memory, database and software assets with other tenants. From a software product line's (SPL) point of view, all tenant configurations of a multi-tenant application constitute a product family. We assume, that model-driven SPLs can be used to generate multi-tenant applications and customise tenant configurations. The abstraction of the system's architecture and variability as models will help us to analyze, deploy and undeploy configurations at run time. We will generate scalable and customisable multi-tenant applications. To achieve this, we will define a compositional application design, where only used functionality is included in an application, but can be extended on demand, e.g., when a new tenant configuration will be inserted. Thus, we will figure out a deployment strategy to find the best matching multi-tenant application in a distributed server environment to deploy a tenant configuration to.

Keywords: Software as a Service, On-Demand System, Dynamically Adaptive System, Deployment Optimisation, Semi-Dynamic SPL

1 Motivation

A multi-tenant application is a special form of a multi-user web application [1]. It is installed once and serves multiple tenants. Two types of users exist in the system: tenants and end-users. A tenant is an organisation that contracts the usage of the application, and to whom's demands the application is tailored. End-users belong to a certain organisation and work with the tailored application. Tenants share resources, like software assets and memory, but have their own data and usage context within the multi-tenant application [2]. To serve multiple tenants, two application designs can be distinguished *single-instance application* and *configurable single-instance application*. In the first case, the same functionality and workflow is offered to all tenants, but every tenant has its own data. In the second case, a tenant does not only have its own data, but the application's functionality can be configured due to customer needs. Since the latter case is more general, we focus on that one. To manage the variability of different tenant

configurations concepts from software product line (SPL) engineering could be applied [3]. Using model-driven SPLs raises the level of abstraction and enables the modeling of functionality as well as the system architecture. Those models can be used at design time as well as at run time for analysing, deploying and undeploying tenant configurations. Therefore designing a model-driven SPL for generating multi-tenant applications is needed *(Problem 1)*. To realise a configurable single-instance multi-tenant application, two architectural designs are possible, a *annotational application design* and a *compositional application design*. In the annotational design approach, the deployed application architecture contains all features of the system, tailored to the tenants at runtime according to their configuration. Because variability binding is done at runtime, all software artefacts need to be deployed. Much storage and performance is required to bind the variation points according to the tenants configuration, because all feature combinations must be checked at runtime. Second, the compositional design approach assumes, that the architecture of the final system only includes used components. This reduces the installation space. Less performance is needed to check the tenant configuration because less features are available. Thus, a compositional application design with support for deployment and undeployment of components at run time is needed *(Problem 2)*. The worst case in this scenaraio is, that all components must be installed as in the annotational approach. That happens only, when the configurations of multiple tenants differ highly from each other. The challenge is to prevent this by using algorithms for effienctly deploying and undeploying tenant configurations *(Problem 3)*.

2 Related Work

Our approach is related to work in the field of SPLs in general, run time adaptation of component based architectures in distributed environments and software architectures for multi-tenant applications.

In [4] the authors propose an approach to create multi-tenant web applications by combining SPL techniques with a multi-tenant service-oriented architecture (SOA). In contrast we use model-driven SPL techniques to model the variability in functionality as well as in the architecture of component-based multi-tenant application. Based on those models, we propose a strategy for efficiently deploying new tenant configurations into running multi-tenant applications. We use a compositional application design, where only used functionality is included in a running multi-tenant application but can be extended on demand.

An approach for dynamic run time adaptation of component-based architectures using feature models is presented in [5]. This approach focuses on self-configurable dynamic software product lines (DSPLs). The provided framework supports static binding of features at compile time and dynamic binding at load or run time. Dynamical binding of features is needed for self-configuration of the system. In contrast, we focus on the deployment algorithm for tenant configurations to find the best fitting multi-tenant application to deploy to, according to containt features and qualities, like server workload or distance.

3 Generating Multi-Tenant Applications

We develop a model-driven SPL to support the dynamic deployment and undeployment of tenant configurations in a distributed server environment (server farm). We call it a *semi-dynamic software product line (SDSPL)*, because selecting and generating a variant takes place at design time, whereas variant deployment takes place at run time *(Objective 1)*. Restarting a server would influence many tenants and is, thus, not desired.

Our approach will provide a three-staged customisation process *(Objective 2)*. First, the provider *configures* the multi-tenant application according to a tenant's needs in the SDSPL. Second, a tenant can *customise* its provided application using given extension mechanisms, e.g., for branding the application. The customisations are deployed in the multi-tenant application's architecture, but only visible to the tenant, who created them. Finally, an end-user can *personalise* the application, e.g., adjusting the look and feel. In the following, we focus on the first stage, the configuration and deployment of a tenant configuration.

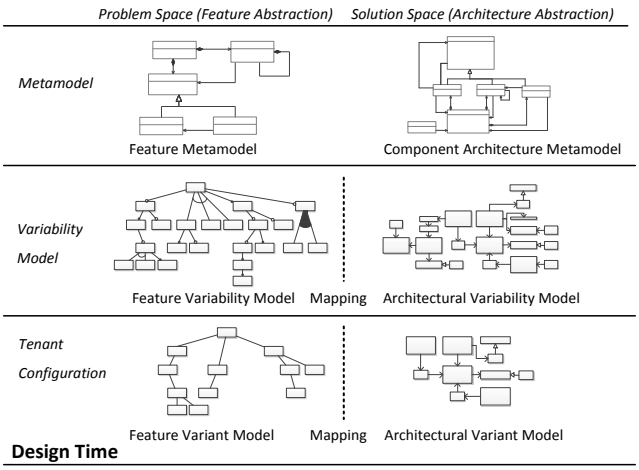

Fig. 1. A Tenant Configuration Represented as Models at Design Time.

In the SDSPL, as shown in Fig. 1, a feature metamodel is used to describe variability in functionality (e.g., alternatives, mandatory functions, optional features, constraints and dependencies between features) in the problem space [6]. Analogously, variability in the application's architecture is described by an architecture metamodel [7] in the solution space. A mapping between both spaces specifies how application functionality is realised technically in the application's architecture. The steps to create a valid tenant configuration at design time are equal to the steps to select a product configuration in a model-driven SPL [8], [9]. We will specify a compositional component-based design for multi-tenant

applications *(Objective 3)*, where models of different tenant configurations form a multi-tenant application as shown in Fig. 2.

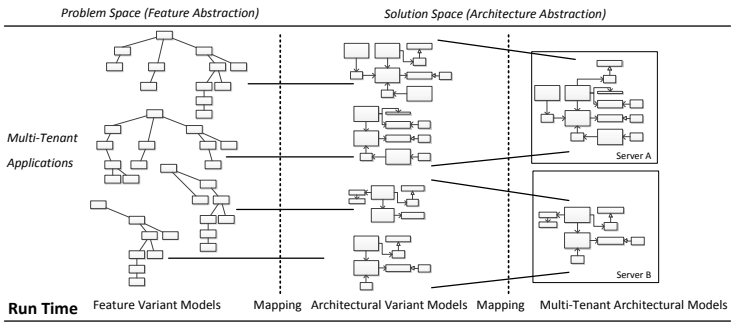

Fig. 2. Tenant Configurations Forming Multi-Tenant Applications at Run Time.

New tenant configurations will be integrated into existing multi-tenant applications by adding new components to the already deployed architecture without influencing other configurations. In contrast to instantiating a single application in the classical SPL way, a new tenant configuration will be integrated into an existing multi-tenant application's architecture at run time. We will define an algorithm to identify the delta between the new tenant configuration and the deployed application *(Objective 4)*. The algorithm takes into account non-functional qualities, like workload or server distance. When a tenant's contract ends, its configuration must be undeployed from the running system *(Objective 5)*. Therefore its configuration model is compared to the remaining ones of other tenants. Unused components are removed and probably a reconfiguration of the remaining components takes place. The tenants configuration and data cannot just be deleted but must be archived.

4 Research Method

We will formalize the given objectives and implement them in an integrated tooling for SDSPL. The environment will support creation and deployment of tenant configurations. The deployment and undeployment algorithms will be integrated in the tooling. As a key factor, our approach will be evaluated on industrial scale application examples. Furthermore, it must be approved, that components at runtime form valid configurations in the multi-tenant application.

Acknowledgements

This research is encouraged and supervised by Prof. Uwe Aßmann and SAP AG. It is co-funded by the European Social Fund, Federal State of Saxony and SAP AG within project #080949335.

References

1. Guo, C.J., Sun, W., Huang, Y., Wang, Z.H., Gao, B.: A Framework for Native Multi-Tenancy Application Development and Management. In: Proceedings of the 9th IEEE International Conference on E-Commerce Technology / 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services. CEC-EEE '07, Los Alamitos, CA, USA, IEEE Computer Society (2007) 551–558
2. Bezemer, C.P., Zaidman, A.: Multi-tenant SaaS applications: maintenance dream or nightmare? In: Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE). IWPSE-EVOL '10, New York, NY, USA, ACM (2010) 88–92
3. Muthig, D., Atkinson, C.: Model-Driven Product Line Architectures. In: SPLC '02: Proceedings of the Second International Conference on Software Product Lines, London, UK, Springer-Verlag (2002) 110–129
4. Mietzner, R., Metzger, A., Leymann, F., Pohl, K.: Variability modeling to support customization and deployment of multi-tenant-aware Software as a Service applications. In: Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems. PESOS '09, Washington, DC, USA, IEEE Computer Society (2009) 18–25
5. Rosenmüller, M., Siegmund, N., Pukall, M., Apel, S.: Combining Runtime Adaptation and Static Binding in Dynamic Software Product Lines. Technical Report 02, School of Computer Science, University of Magdeburg (February 2011)
6. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie Mellon University Pittsburgh, Software Engineering Institute (1990)
7. Moon, M., Chae, H., Yeom, K.: A Metamodel Approach to Architecture Variability in a Product Line. In Morisio, M., ed.: Reuse of Off-the-Shelf Components. Volume 4039 of Lecture Notes in Computer Science., Berlin, Heidelberg, Springer-Verlag (2006) 115–126
8. Czarnecki, K., Antkiewicz, M., Kim, C.H.P., Lau, S., Pietroszek, K.: Model-driven software product lines. In: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. OOPSLA '05, New York, NY, USA, ACM (2005) 126–127
9. Pohl, K., Böckle, G., van der Linden, F.: Software product line engineering - foundations, principles, and techniques. Springer (2005)