A DMN-based Approach for Dynamic Deployment Modelling of Cloud Applications

Frank Griesinger[1], Daniel Seybold[1], Jörg Domaschka[1], Kyriakos Kritikos[2], and Robert Woitsch[3]

[1] University of Ulm, Institute of Information Resource Management, Ulm, Germany
{firstname.lastname, joerg.domaschka}@uni-ulm.de
[2] ICS, FORTH, Heraklion, Greece
kritikos@ics.forth.gr
[3] BOC Asset Management, Vienna, Austria
robert.woitsch@boc-eu.com

Abstract. Cloud computing is well suited for applications with a distributed architecture and dynamic demand of resources. Yet, current approaches to model cloud application deployment do not cater for the application's dynamic nature and its rapidly changing business requirements. The static description of deployments results in a lack of reusability and also lacks an integrated way to adapt to the current context. To reuse and refine the deployment model, we introduce a *simple decision layer* on top of a cloud application description, which abstracts from the actual deployment language and allows assembling the deployment model from existing model fragments. Those fragments are chosen based on the input of the decision process. We define an architecture for the decision layer and sketch an implementation based on CAMEL, DMN, and ADOxx. The benefits of the decision layer are illustrated by two use cases. Our approach shifts the focus from a static to a dynamic and reusable modelling process, which also reduces the modeller's effort.

Keywords: DMN, DevOps, MDE, cloud, deployment modelling

1 Introduction

The cloud computing paradigm promises the unlimited offering of computational resources in a pay-as-you-go model. This helps organisations, especially SMEs, with unplannable or highly dynamic resource demands, to dynamically reserve IT resources as needed without having a huge upfront investment.

The benefits of cloud computing come along with an additional technical depth, which may hinder the migration to the cloud. Hence, industry and academia investigate approaches easing the cloud adoption. A well established approach to reduce technical complexity is model-driven engineering (MDE). Within MDE, domain specific languages (DSLs) for the cloud computing domain evolved, including TOSCA [4] and CAMEL [5]. Such DSLs ease the cloud adoption by enabling a complex cloud application deployment model on a higher level. A cloud orchestration tool (COT), such as Cloudiator [2] can then process this deployment model.

Still, the specification of a deployment requires a certain degree of technical knowledge to create the deployment model, which is static in nature. Current modelling approaches do not reflect the dynamic in changing business requirements that impact an implemented deployment model at run-time. As shown in the first lane of Fig. 1, any requirement change leads to the remodelling of the deployment model, which is error-prone and cost-intensive.

In this position paper, we propose a novel approach by adding *(i)* dynamic and *(ii)* reusability to cloud application modelling. We introduce a simple decision layer on top of the modelling, enabling the transformation of business requirements into a technical deployment model at both design- and run-time as shown in the second lane of Fig. 1. Based on two use cases, we demonstrate how our approach enhances the scope of cloud application modelling and therefore eases cloud adoption.

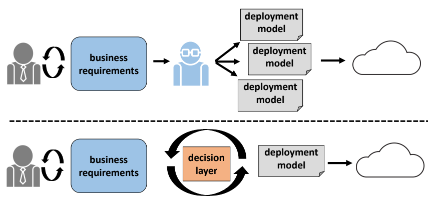

Fig. 1. Dynamic deployment modelling

The remainder of the paper is structured as follows: Section 2 analyses the problem. Section 3 presents our approach and sketches an implementation. The usage is presented on two use cases in Section 4. Section 5 discusses the approach, while Section 6 summarises the related work. Section 7 concludes the paper.

2 Problem Statement

Modelling cloud applications is technically challenging and therefore error-prone. In addition, most approaches have a steep learning curve. The process of current modelling approaches is a static sequence of the steps (cf. Fig 1): *(i)* business experts define high level business requirements, *(ii)* technical experts manually map these requirements to a technical deployment model using a cloud DSL, *(iii)* the deployment model is put into a COT. The shortcomings of this process are founded in the dynamic nature of cloud applications, and as soon as a requirement changes, the complete process has to start from the beginning. Due to its complexity, the repetition of all steps is cost-intensive and error-prone. This is also caused by the fact that current cloud modelling approaches do not focus on the automated reuse of model fragments and manual involvement increases the risk of failure. In addition, decisions are only implicitly integrated and hard-coupled into the deployment model. This hinders the employment of a feedback loop in this process to re-evaluate the business requirements when needed.

The following scenarios exemplify the shortcoming of a static procedure: *(i)* A customer-specific deployment model that requires minor adjustments, concerning the cloud provider or data location, will result in an independent model per customer. *(ii)* Update roll-outs are an important feature of DevOps tools. With the release of a version the deployment model changes. In cloud modelling approaches, there is no support to model the dependency of the version in respect to the deployment model. *(iii)* The service configuration, such as cloud provider or virtual machines specification, highly depends on business requirements, such as the available budget.

We argue that these shortcoming can be removed by adding *(i) dynamic* and *(ii) reusability* to the modelling process.

3 Dynamic Cloud Modelling

This sections is structured as follows. First, we present a solution for adding dynamic to the modelling process. Followed by a realisation sketch of this solution.

3.1 Introducing a Decision Layer

We propose a novel approach to ease the process of cloud application modelling by adding a *simple decision layer* on top of existing cloud DSLs. The proposed decision layer operates between higher level *(business) requirements* and low level concrete *model fragments*. Business requirements are integrated as influencing factors of a decision process which maps them to concrete model fragments. These model fragments are used to assemble the deployment model.

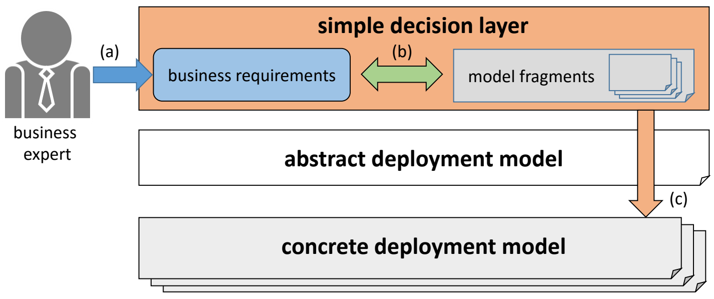

Fig. 2. Simple decision layer on top of deployment model.

As shown on Fig. 2, the simple decision layer handles business requirements and concrete model fragments. The decision layer abstracts from the actual language, by using an *abstract deployment model* that comprises anchor points for a decision process to concretise the deployment model by the output values of an evaluated decision. In contrast to the *concrete deployment model*, it is not completely described and therefore not executable in a COT. The business requirements are fed into the layer by business experts (Fig. 2 (a)). The mapping of requirements to model fragments is done based on a business knowledge model

(Fig. 2 (b)). When requirements arrive or change, the decision layer executes a decision process. After evaluating the business requirements, appropriate model fragments are selected and used to enhance the abstract deployment model and create a concrete deployment model that is executable by a COT (Fig. 2 (c)).

Based on the requirements, the decision layer operates on an abstract deployment model and a *decision set* that is defined just once, to reuse existing model fragments in order to create multiple concrete deployment models.

While in current approaches the concrete deployment model has to be modelled per business requirement set, this approach requires to model one abstract deployment model and define its business knowledge model for arbitrary requirement sets. In order to extend available business requirements, additional mapping decision can be added to the business knowledge model. The simple decision layer is then able to reuse those model fragments for new incoming business requirements, as well as for other abstract deployment models.

In order to deal with changing requirements at run-time, the decision layer will reevaluate the decisions and update the respective model fragments. Thus, the proposed decision layer enables dynamic redefinition of the required model fragments based on the predefined decision set.

3.2 Realisation Sketch

We propose a realisation of our approach based on the Decision Model and Notation (DMN) [3] as decision layer and CAMEL [5] as the cloud modelling DSL.

The DMN standard provides a human-readable common notation for modelling and automating decisions. We choose decision tables (DTs) to represent decisions as these are well known to business experts. An example of a DT is shown in Table 1. A DT consists of three column types: *(i)* a hit policy, *(ii)* an input variable set, and *(iii)* an output variable set. The hit policy defines the selection over overlapping decisions with policies like **U**nique, i.e., only a single decision will be selected or **C**ollect, i.e., all decisions can be selected. Each input variable can potentially map to a respective output variable of a sub-decision table. Hence, there is a possible cascade of decisions leading to hierarchical decision tables. Any DT is associated with a business knowledge model (BKM) defining the decision logic, i.e., the mapping between the input and output parameters. DMN is chosen as it is an impact gaining standard and it is already well adopted on the business level.

CAMEL models encompasses all technical details to deploy an application in the cloud, including specific cloud resources such as virtual machines and deployment structure. A concrete deployment model can be transformed into a set of cloud-provider specific deployment actions. We favoured CAMEL over other cloud application modelling languages like TOSCA as CAMEL supports the specification of a provider-independent deployment model, as well as an instance model.

Our proposed realisation is depicted in Fig. 3 implementing the decision layer as a hierarchical set of DTs enabling the dynamic CAMEL modelling. The DTs

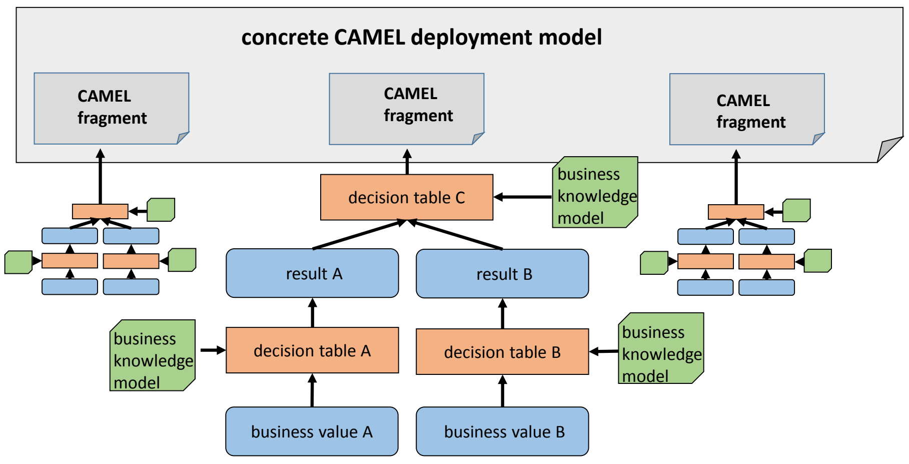

Fig. 3. Dynamic CAMEL modelling

are specified by BKM fragments, which define the actual decisions in the DTs. We distinguish between two different types of output values, DMN results, used as input for other DTs, and derived CAMEL fragments.

Both languages are integrated into the meta-modelling platform ADOxx[4], providing a modelling tool for dynamically generating CAMEL models via DMN. ADOxx is able to provide a modelling user interface and the integration of algorithms to implement the usage of meta models. The main scenarios taken into consideration are *(i)* the specification of DMN via a graphical user interface, and *(ii)* the support of the execution of DMN to generate the CAMEL model.

4 Use Cases

We present use cases from the areas of DevOps and business-IT-alignment. We exemplify achieving their requirements by integrating our approach.

4.1 Customer-specific and Continuous Deployment

A common requirement in DevOps environments is having a *customer-specific deployment* that differs slightly due to customers' specific favors, and *continuous deployment* on version updates. Introducing a decision process enables in this case the reusability of the cloud application model. The application is only modelled once, but the concrete deployment model is generated dynamically for different requirements of the customers and of the application version.

A sample excerpt of a DT is shown in Table 1. The input of this DT are the trusted cloud provider and the privacy level. The output is the VM image and region for the model to be used for the service deployment.

[4] `https://www.adoxx.org/`

Table 1. Image and Region DT

Hit Policy	**Input**		**Output**	
C	Privacy Level String	Provider String	VM Image String	Region String
1	low	Provider X	Image X	US
2	low	Provider Y	Image Y	Europe
...	...	...	...	...

4.2 Business Process as a Service

CloudSocket[5] introduces the concept of *Business Process as a Service (BPaaS)* by modelling business processes (BPs) on the highest level and semi-automatically align the BPs to the technical description of the required cloud services for the BP execution [8]. The BPaaS approach comprises a sequence of mappings from higher level business descriptions to low level technical descriptions. This chain contains points where decisions are made to create models of different levels of detail. The proposed layer caters for this mapping as it allows to integrate the business requirements to the model creation process. The decisions that have to be made in the BPaaS approach define, *(i)* which service to use, *(ii)* which configuration, such as cloud provider and hardware, the service will have and *(iii)* the service's behaviour at run-time.

5 Discussion

The presented approach has a major impact on evolving the current state of the art with respect to managing cloud applications through COTs and also for the features supported by COTs. As the input parameters for decision processes may change during run-time, COTs have to be able to update a deployed application on-the-fly according to the changes in the model. To implement this, a COT will need to create a change set between old model and new model and apply actions that implement the changes. This will involve adaptation actions currently not supported by any COT such as the migration of components onto different clouds.

In current modelling environments, the modeller directly interacts with a DSL or a direct (graphical) editor. Our approach shifts this view for modellers to a paradigm above the actual DSL. She will outline the deployment model by specifying the decisions that lead to the actual deployment. This will increase the reusability of cloud description fragments and cater for the dynamic nature of cloud-based applications and lower the learning curve for decision makers.

Although the paper motivates the importance of integrating business requirements in the decision process of cloud modelling, the presented approach is able to involve any kind of requirements, e.g. technical requirements, as the decision process is agnostic to the type of requirement. Also the rules described in the

[5] `https://www.cloudsocket.eu/`

decision tables of the introduced layer can be translated into adaptation rules in the DSL. Obviously, this would demand the specification of the correlation between business requirements and e.g., the number of component instances in the case of scaling rules.

By applying our approach, application modeller can create abstract models and distribute them in a marketplace-like manner. Companies can choose from those abstract deployment models, customise them, and create concrete deployment models by the means of a company's specific business requirements. In contrast to similar application libraries of current COTs, our approach does not suffer from static models that needs low level adjustments to customise it.

6 Related Work

Besides DMN, there are numerous approaches for decision engines like Gandalf[6] or the IBM Operational Decision Management[7] that also apply decision tables to define business rules. Those can also be used to run the decision layer.

The usage of interconnected ordered decision tables as a selection method for cloud services can be categorised as a multi-criteria decision-making (MCDM) process [7]. In contrast to optimisation-based approaches realising MCDM decisions using utility functions, we are convinced that our approach is more user-friendly and -intuitive, due to the use of human-readable tables as interfaces.

Cloud orchestration tools mainly use DSLs to specify the deployment models [1]. However, they do not automatically create a set of differences to integrate modifications due to a decision process. This becomes necessary, when business requirements are evaluated on run-time and a feedback loop is integrated.

DevOps tools like Puppet[8] support updating an application at run-time on the basis of the differences between the latest and the currently active configuration. However, those DevOps tools operate on the level of the single component. They do not consider the overall view on the cloud-based application. The autonomous provisioning of infrastructure or platform resources is also out of scope of such tools. Our model-driven approach integrates the update functionality by using the application version as input for the decision process.

ToscaMart [6] introduces the idea of reusing model fragments for modelling by employing a marketplace of predefined application components to be used by the modellers to assemble their applications. This approach lacks a general decision-making process but instead relies on lowest technical requirements of the application to be assembled.

7 Summary

Cloud deployment models are described in domain specific languages (DSLs). Current DSLs are static and the creation comes along with the complexity of

[6] `https://gndf.io/`
[7] `http://www-03.ibm.com/software/products/de/odm`
[8] `https://puppet.com/`

many technical details. Whereas modelling decisions are taken at design-time, influencing factors, such as technical or business requirements require to update fragments of the deployment model at run-time. Current modelling approaches only cater for updates of the complete deployment model and do not consider the reusability of constant model fragments.

In this position paper, we proposed a simple decision layer residing above current DSLs. This decision layer enhances the modelling scope by considering decisions affecting the deployment model at design and run-time.

We sketch a realisation based on the decision model and notation (DMN) as decision layer. DMN enables the semi-automatic creation of the deployment model from the results of DMN decision tables. Our realisation proposes the usage of DMN with the cloud DSL CAMEL in the ADOxx modelling environment. The suitability of the approach is discussed based on two use cases types.

Future work will encompass a prototype implementation of the presented decision layer based on the outlined technologies. Based on the prototype an evaluation on the impact of model creation and execution will be performed.

Acknowledgements The research leading to these results has received funding from the EC's Framework Programme FP7/2007-2013 under grant agreement number 317715 (PaaSage) and the EC's Framework Programme HORIZON 2020 (ICT-07-2014) under grant agreement number 644690 (CloudSocket).

References

1. Baur, D., Seybold, D., Griesinger, F., Tsitsipas, A., Hauser, C.B., et al.: Cloud orchestration features: Are tools fit for purpose? In: 2015 IEEE/ACM 8th Int. Conf. on Utility and Cloud Computing (UCC). pp. 95–101. IEEE (2015)
2. Domaschka, J., Baur, D., Seybold, D., Griesinger, F.: Cloudiator: A Cross-Cloud, Multi-Tenant Deployment and Runtime Engine. In: 9th SummerSoC (2015)
3. Group, O.M.: Decision model and notation. Tech. rep., OMG, `http://www.omg.org/spec/DMN/1.1/` (2015)
4. OASIS: Topology and Orchestration Specification for Cloud Applications Version 1.0 Committee Specification Draft 08 (2013)
5. Rossini, A.: Cloud Application Modelling and Execution Language (CAMEL) and the PaaSage Workflow. In: Adv. in Service-Oriented and Cloud Comp.—Workshops of ESOCC 2015. CCIS, vol. 567, pp. 437–439. Springer (2016)
6. Soldani, J., Binz, T., Breitenbcher, U., Leymann, F., Brogi, A.: Toscamart: A method for adapting and reusing cloud applications. Journal of Systems and Software 113, 395 – 406 (2016)
7. Sun, L., Dong, H., Hussain, F.K., Hussain, O.K., Chang, E.: Cloud service selection: State-of-the-art and future research directions. Journal of Network and Computer Applications 45, 134 – 150 (2014)
8. Woitsch, R., Utz, W.: Business process as a service: Model based business and it cloud alignment as a cloud offering. In: 2015 Int. Conf. on Ent. Sys. (ES). pp. 121–130. IEEE (2015)