The EDMM Modeling and Transformation System

Michael Wurster[1], Uwe Breitenbücher[1], Antonio Brogi[2], Ghareeb Falazi[1], Lukas Harzenetter[1], Frank Leymann[1], Jacopo Soldani[2], and Vladimir Yussupov[1]

[1] Institute of Architecture of Application Systems, University of Stuttgart, Germany
[lastname]@iaas.uni-stuttgart.de
[2] Department of Computer Science, University of Pisa, Pisa, Italy
[lastname]@di.unipi.it

Abstract. Since deployment automation technologies are heterogeneous regarding their supported features and modeling languages, selecting a concrete technology is difficult and can result in a lock-in. Therefore, we presented the Essential Deployment Metamodel (EDMM) in previous work that abstracts from concrete technologies and provides a normalized metamodel for creating technology-independent deployment models. In this demonstration, we present tool support for EDMM in the form of the EDMM Modeling and Transformation System, which enables (i) creating EDMM models graphically and (ii) automatically transforming them into models supported by concrete deployment automation technologies.

Keywords: Deployment Modeling, Automation, Transformation, Tool

1 Motivation: The Deployment Technology Lock-In

An integral aspect of efficient application deployment processes is that they must be highly automated: Manually deploying applications consisting of multiple components is complex, time-consuming, error-prone, and, moreover, requires immense technical expertise to execute the technical deployment tasks. Therefore, several *deployment automation technologies* have been developed in the past years that are actively used by industry and research. Deployment technologies are usually offered as a software system or service that can deploy applications fully automatically by processing so-called *deployment models*. Deployment models can be categorized into two types: (i) imperative models and (ii) declarative models [1]. The main idea of imperative models is to describe a detailed, executable process specifying all necessary technical tasks to be executed, their implementations, and their order. In contrast, declarative models only describe the components to be deployed, their configurations, and the relations between them, but hardly provide technical execution details. Declarative models, hence, need to be interpreted by a deployment automation technology that derives the technical deployment instructions while an imperative model can be directly executed as-is. Since our previous work [3] has shown that 13 of the most important deployment technologies are either declarative by nature or support declarative deployment modeling, we focus on declarative deployment models in this work.

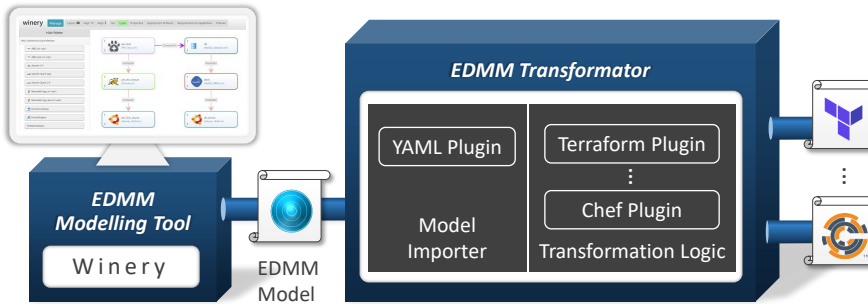

Fig. 1. EDMM Modeling and Transformation System Architecture.

However, the available deployment technologies are heterogeneous regarding their features and supported modeling languages. Thus, deciding for a specific technology quickly results in a *Deployment Technology Lock-In*, which means that it is hard to exchange the technology later. The main reasons for this lock-in result from (i) the deep technical expertise that needs to be acquired to work with such a technology and (ii) the need to rewrite all deployment models that are currently in use. Therefore, we introduced the *Essential Deployment Metamodel (EDMM)* in previous work [3], which abstracts from concrete technologies and provides a normalized metamodel that only supports commonalities of the 13 most important technologies. Thereby, it enables to create *deployment technology-agnostic EDMM models* that can be translated into each of the 13 technologies following the translation guidelines we presented in Wurster et al. [3]. However, this translation is currently a manual, time-consuming, and error-prone approach.

2 The EDMM Modeling and Transformation System

To tackle this issue, in this demonstration, we present the *EDMM Modeling and Transformation System* shown in Fig. 1, which consists of (i) the *EDMM Modeling Tool* and (ii) the *EDMM Transformation Framework*. Using the EDMM Modeling Tool, a user is able to graphically model the deployment of an application in the form of an EDMM model that describes the components to be deployed, their configurations, their implementations, and their relations. The resulting EDMM model is independent of concrete deployment technologies and can be exported as a file. This EDMM model file can be fed into the EDMM Transformation Framework, which offers a command-line interface (CLI) that can be either used directly by the user or integrated into any automation workflow. Using the CLI, the desired target deployment technology in which the EDMM model should be transformed can be selected. The output is an executable, technology-specific deployment model, which can be executed using the selected technology. Our prototype[3], as well as a video demonstrating the system, are available on GitHub.

[3] `https://github.com/UST-EDMM/transformation-framework`

EDMM Modeling Tool

The EDMM Modeling Tool has been developed by extending Eclipse Winery [2], which is a web-based environment to graphically model TOSCA-based application topologies. It includes (i) a *back-end* to manage component and relation types, their properties, and artifacts and (ii) a *Topology Modeler* that enables to graphically compose application components and specify configuration properties. Since EDMM can be mapped to TOSCA [3], Winery has been extended by providing an export plugin to transform its internal TOSCA-based data model to the YAML format defined by EDMM. The EDMM export functionality was developed for the Java back-end and is merged to Winery's official *master* branch. Further, an administration component in the Angular user-interface has been added to specify custom type mappings between the maintained TOSCA node types and the built-in EDMM types. The Topology Modeler itself did not need an extension as we fully rely on Winery's internal data model during modeling.

EDMM Transformation Framework

The EDMM Transformation Framework provides a CLI for transforming EDMM models into technology-specific deployment models. At this stage, the framework supports YAML files as input according to the published EDMM YAML specification[4]. All components, as well as their component types, must be provided in a single EDMM model file at the time of writing.

We designed the framework to employ a plugin architecture that supports integrating various deployment technologies in an extensible and pluggable way. Each plugin defines an identifier and a corresponding display name, e.g., the "`kubernetes`" plugin is implemented to transform EDMM-based models into "Kubernetes" resource files. The transformation can be started by using the `transform` command of the CLI: The user has to specify the EDMM model file and the identifier of the target deployment technology. For the framework, we use Java with Spring and Spring Boot to build the CLI as well as to load the plugins dynamically once they are registered in a configuration file. Each plugin must implement a `transform()` method to execute the required transformation logic. Further, a plugin may implement different lifecycle methods: (i) `checkModel()` to indicate whether a model can be transformed by a plugin, (ii) `prepare()` to execute preparation activities prior to the transformation, e.g., download external files, and (iii) `cleanup()` to execute clean up activities after the transformation.

The internal data model of the EDMM Transformation Framework is based on and represented as a graph using the Java library JGraphT. By employing a graph, also for the reason that the component structure in an EDMM-based model naturally forms a graph, the plugins are able to efficiently traverse the data model to apply the respective transformation logic. Plugins may apply arbitrary graph algorithms, e.g., topological sorting of components to traverse the graph in a certain way. Further, this also enables to make use of the visitor pattern to add or extend new plugin logic without modifying the graph structure.

[4] `https://github.com/UST-EDMM/spec-yaml`

Developed Plugins and Supported Component Types

Currently, the framework supports all 13 deployment technologies which were systematically selected and reviewed by Wurster et al. [3]. Details of the plugins' implementations and the transformation rules can be found in the documentation. Please note: In this demonstration, we only focus on deployments that are based on virtual compute resources, i.e., operating systems, virtual machines, or containers, and on the software that needs to be deployed on them including their configuration and orchestration[5]. Therefore, we introduce a couple of built-in EDMM component types as modeling baseline. The base of all supported deployments is represented by the *Compute* component type that permits modeling a virtual compute resource, which can be then transformed by a plugin into a virtual machine or container, respectively, depending on the target technology's capabilities. For example, a Compute component gets transformed into a virtual machine for OpenStack Heat, while it is transformed into a container for Kubernetes. We also defined several software component types that can be installed on Compute components, e.g., a MySQL database. To install such components, either the plugin (i) contains built-in logic to translate a certain component type into the corresponding modeling element in the target model or (ii) it uses *EDMM Operations*, which provide generic plug-points in EDMM models to specify installation scripts for components that can be injected into the target model by the plugin. Also, the orchestration of components is supported, e.g., to connect an application to its database (possibly hosted on different Compute components), plugins inject the properties of the target component, e.g., IP address, as environment variable into the source component, which enables using them, for example, in installation scripts. In future work, we plan to extend the plugins for other types of components, e.g., PaaS, FaaS, and other Cloud services.

Acknowledgments This work is partially funded by the European Union's Horizon 2020 research and innovation project *RADON* (825040), the DFG project *SustainLife* (379522012), and the projects *AMaCA* (POR-FSE) and *DECLware* (University of Pisa, PRA_2018_66).

References

1. Endres, C., et al.: Declarative vs. Imperative: Two Modeling Patterns for the Automated Deployment of Applications. In: Proceedings of the 9th International Conference on Pervasive Patterns and Applications. Xpert Publishing Services (Feb 2017)
2. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: Winery – A Modeling Tool for TOSCA-based Cloud Applications. In: ICSOC 2013. pp. 700–704. Springer (2013)
3. Wurster, M., et al.: The Essential Deployment Metamodel: A Systematic Review of Deployment Automation Technologies. SICS Software-Intensive Cyber-Physical Systems (Aug 2019)

[5] An example that is supported by all developed plugins can be found here: `https://github.com/UST-EDMM/getting-started`