On Viewpoint-specific Microservice Modeling

Philip Wizenty

Institute for the Digital Transformation of Application and Living Domains,
University of Applied Sciences and Arts Dortmund,
Otto-Hahn-Straße 23, 44227 Dortmund, Germany,
philipnils.wizenty@fh-dortmund.de

1 Introduction

A microservice is typically designed, developed and deployed by a single team, with the potential use of the DevOps paradigm [4]. DevOps teams typically consist of service developers, operators and possibly domain experts, i.e., the roles of teams' members are heterogeneous [5]. Thus, microservice team members have different viewpoints on the system regarding their tasks, e.g., the design and implementation of a microservice as a service developer.

Model-driven Development (MDD) [1] is an approach to software engineering, that abstracts from implementation details and is in particular beneficial to engineer complex and distributed systems [3]. The benefits, provided by the usage of MDD, are model validation to test systems' correctness, simulations to predict systems' behavior and code generation to increase development productivity and reduce the number of errors. Furthermore, MDD enables a viewpoint-specific abstraction of the software system by providing specialized models that address the viewpoints of different stakeholders, e.g., domain experts create domain models for domain concepts, while service developers create models that focus on microservice design.

In the following, we present three metamodels for viewpoint-specific modeling languages, that can be used by DevOps-based microservice teams to create models for exploiting the mentioned benefits of MDD in the context of Microservice Architecture (MSA) engineering.

2 Viewpoint-specific Metamodels for MSA Modeling

The metamodel for domain experts (cf. Figure 1a) provides concepts to build models, which capture domain-specific information for microservices. The metamodel supports `PrimitiveType` and `ComplexType` data types. `DataStructure` and `ListType` are specializations of the complex data types, which can be used to build data structures and lists of them. Furthermore, the `Context` and `Version` concepts support domain model organization. Semantically, the `Context` concept corresponds to the Bounded Context pattern of Domain-driven Design [2].

The metamodel for service developers (cf. Figure 1b) is built around the `Microservice` concept. `Endpoints`, `Interfaces`, `Operations` and `Protocols` of microservices can be modeled to specify essential service components. For the assignment of data types to operation parameters in a service model, the metamodel defines the `Import` concept. It provides the possibility to import complex types from domain models based on the metamodel in Figure 1a. To model dependencies between microservices, the service metamodel also allows the import of service models.

The metamodel for service operators in DevOps-based microservice teams is depicted in Figure 1c. Operators can create models to specify aspects of microservice deployment and operation. For instance, the employed `OperationTechnology`, e.g., Docker[1], and

[1] https://www.docker.com

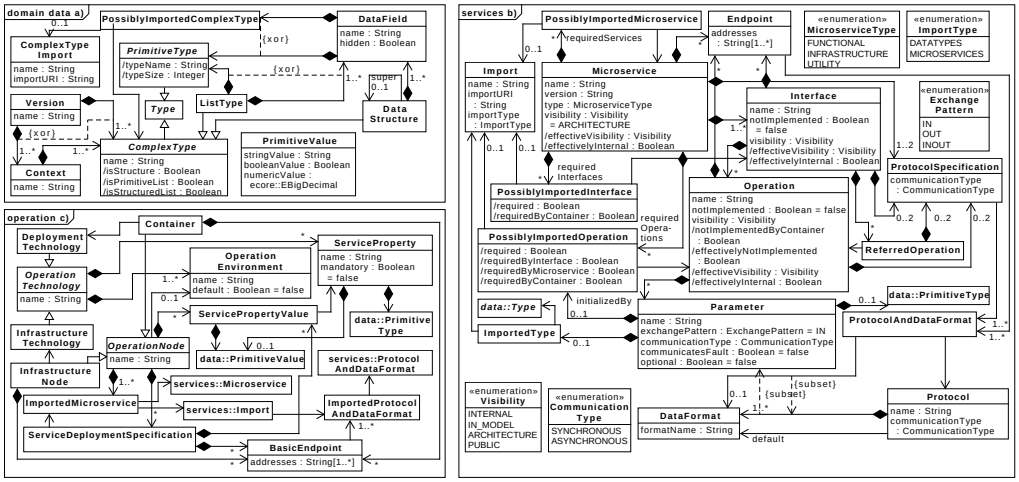

Figure 1: Domain Data, Service and Operation Viewpoint Metamodel

`OperationEnvironment`, e.g., a Docker image, can be expressed. `OperationNode` is the central concept of the metamodel. It represents either a `Container` or `InfrastructureNode`, and references a microservice being imported from a service model (cf. Figure 1b) to specify its deployment and related characteristics like its `BasicEndpoints`.

3 Conclusion

In our paper, we introduced metamodels for different viewpoints in DevOps-based microservice teams to enable viewpoint-specific MDD of MSA. Currently, we are working on code generators to derive executable service code as well as deployment descriptors from service and operation models being based on the metamodels.

In our talk, we focus on the practical usage and benefits of MDD for MSA by showing the usage of concrete modeling languages for the metamodels, which are realized as Eclipse[2] plugins and is published on Github[3]. Therefore, we will create domain, service and operation models for a case study microservice architecture from the electromobility domain.

References

[1] Benoit Combemale, Robert B. France, Jean-Marc Jézéquel, Bernhard Rumpe, Jim Steel, and Didier Vojtisek. *Engineering Modeling Languages*. CRC Press, 2017.

[2] Eric Evans. *Domain-Driven Design*. Addison-Wesley, 2004.

[3] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. *Proc. of the 2007 Workshop on Future of Software Engineering (FOSE)*, 2007.

[4] Hui Kang, Michael Le, and Shu Tao. Container and microservice driven design for cloud infrastructure devops. In *Proc. of the Int. Conf. on Cloud Engineering*, pages 202–211. IEEE, 2016.

[5] Irakli Nadareishvili, Ronnie Mitra, Matt Mclarty, and Mike Amundsen. *Microservice Architecture*. O'Reilly Media, 2016.

[2] http://www.eclipse.org
[3] https://github.com/SeelabFhdo/ddmm