AjiL: A Graphical Modeling Language for the Development of Microservice Architectures

Jonas Sorgalla

University of Applied Sciences and Arts Dortmund
Institute for Digital Transformation of Application and Living Domains
44227 Dortmund, Germany
jonas.sorgalla@fh-dortmund.de

1 Introduction and Background

Microservice Architecture (MSA) represents an architectural style for distributed and service-based systems [3]. As such, MSA emphasizes loosely coupling between services whereby each service is realized as an independent process that can be designed and operated autonomously. Each individual service corresponds to a certain business capability. For communication, MSAs rely on smart endpoints and simple message-based protocols like RESTful HTTP. MSA is well suited to address the rising demand for cloud applications as well as shorter and more agile development cycles [3]. However, this comes at the cost of more complexity in certain areas, e.g. distributed data models or the need to prepare well-defined endpoints for each service to consider possible choreography-based calls from other services. Additionally, the development of an MSA is marked by writing a lot of boilerplate code to set up the necessary infrastructure, e.g. service discovery or an API gateway, rather than implementing actual business logic [5].

This paper aims to bridge this conceptual gap between the actual problem domain of an MSA and its infrastructure-centered implementation with the help of Model Driven Engineering (MDE). MDE leverages models as an abstraction of an actual software system and its components to tame its complexity [2]. Therefore, we propose the Aji Modeling Language (AjiL) as a supportive tool to describe and create MSAs. AjiL and its sources can be accessed on the AjiL GitHub repository[1].

Besides AjiL, there are various tools to ease the development of MSAs. For example, MAGMA [5] is a tool to create microservices based on Maven archetypes. Another tool is JHipster[2] which provides customized microservices based on a web dialog. However, to the best of our knowledge there is currently no graphical diagram tooling solely dedicated to MSA besides AjiL.

2 The Aji Modeling Language

As a modeling language, AjiL comprises three components: (i) Abstract syntax, (ii) Concrete syntax, and (iii) Semantics [1]. The abstract syntax was derived from several public MSA examples and is depicted as a Unified Modeling Language (UML) class diagram in Figure 1.

Starting from the system as the root element, an MSA consists of several microservices which can be classified according to their functional or infrastructural purpose. Each service consists of a domain model, which aggregates multiple entities, and one or more interfaces. In AjiL, interfaces can provide abilities, e.g. create or read, to manipulate entities of a service and

[1] https://github.com/SeelabFhdo/AjiL
[2] https://jhipster.github.io

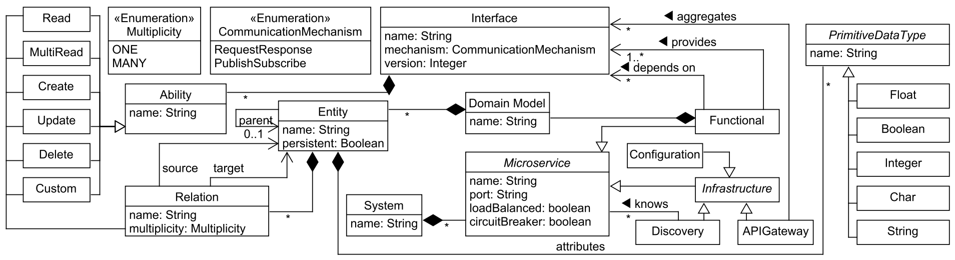

Figure 1: UML class diagram of the AjiL metamodel.

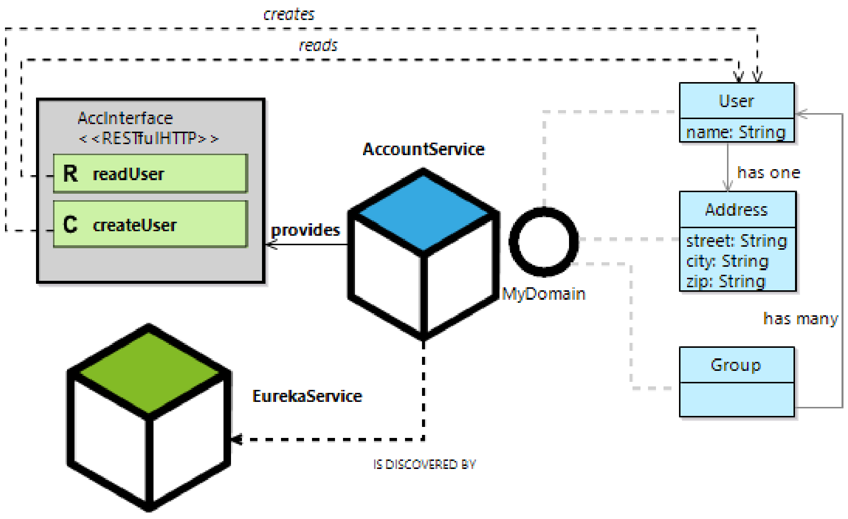

Figure 2: Example model of a functional service using the graphical AjiL notation.

thus are used to describe the endpoint of a service. In addition to the multiplicities shown in the figure, the abstract syntax comes with several constraints, e.g. to conceal entity relations to a single service. The constraints form AjiL's syntactical semantics and are formulated in the Object Constraint Language (OCL).

AjiL's concrete syntax, which is exemplified by the `AccountService` in figure 2, can be characterized as a Box-and-Line diagram type [1]. The notation differs significantly from the widespread UML. Rather than stereotypes in UML, the notation uses shapes and colors to distinguish between the different language elements. For creating such diagrams, AjiL comes with an editor which is realized using the Eclipse Sirius framework[3].

At last, AjiL as a tool comprises a template-based generator realized with the Epsilon Generation Language (EGL) [4]. The generator is able to convert persisted AjiL diagrams, created with the editor, into a runnable MSA. The MSA is based upon Java and the Spring framework.

[3] https://eclipse.org/sirius

3 Conclusion and Future Work

This paper presented AjiL as a graphical way to describe and generate MSAs. AjiL comes with an Eclipse-based editor and generator. In the future, we plan to extend the generator which currently does not utilize all information from an AjiL model. Additionally, we plan to further improve the graphical notation as well as validate the underlying metamodel based upon testing in the field.

References

[1] B. Combemale, R. France, J. Jézéquel, Bernhard Rumpe, J. Steel, and D. Vojtisek. *Engineering modeling languages.* Chapman & Hall/CRC innovations in software engineering and software development. Taylor & Francis, CRC Press, Boca Raton, 2016.

[2] Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering.* IEEE Computer Society, 2007.

[3] Paolo Francesco, Patricia Lago, and Ivano Malavolta. Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption. In *2017 IEEE Int. Conf. on Software Architecture*, 2017.

[4] Dimitrios Kolovos, Louis Rose, Richard Paige, and A Garcia-Dominguez. *The Epsilon Book.* 2010.

[5] Philip Wizenty, Jonas Sorgalla, Florian Rademacher, and Sabine Sachweh. Magma: Build management-based generation of microservice infrastructures. In *Proceedings of ECSA 2017*, 2017.