Position Papers of the Federated Conference on Computer Science and Information Systems pp. 73–80
DOI: 10.15439/2018F370
ISSN 2300-5963 ACSIS, Vol. 16

A Model-Driven Approach to Microservice Software Architecture Establishment

Branko Terzić, Vladimir Dimitrieski, Slavica Kordić, Ivan Luković
University of Novi Sad, Faculty of Technical Sciences,
Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia
Email: {branko.terzic, dimitrieski, slavica, ivan}@uns.ac.rs

***Abstract*—In this positional paper we propose a model-driven approach which addresses challenges related to modeling, development and deployment of software applications that follow the microservice architecture (MSA) design principles. We argue in favor of a model-driven tool which can be used to resolve challenges from the MSA establishment domain by providing a domain-specific language for MSA modeling and code generators for producing: (i) program and configuration code for MSA implementation; and (ii) program procedures for MSA building, packaging and installation. We give a brief description of two approaches to software application development which emerged in the last decade: the monolithic architecture approach and the MSA approach. We focus on challenges related to MSA establishment and argue that our model-driven approach could be suitable for their resolution. We also propose a plan of research activities aimed at improving our approach and which will lead to the final implementation of a model-driven tool to support such an approach**

I. Introduction

IN THE past decade two approaches to software application development became dominant among the majority of engineers: (i) software application that follows the Monolithic Software Architecture (MTA) design principles; and (ii) software application that follows the Microservice Software Architecture (MSA) design principles [1].

MTA is composed of software modules (SM) that mainly cannot exist and run independently from the core application they belong to [2]. Therefore, the whole business logic layer of the application typically runs within a single operating system process and all SMs execute within that process. Since all SMs are tightly coupled, development of individual SMs is hard to strictly divide between engineering teams. Accordingly, MTA software solutions are harder to develop, test and maintain [3]. Also, there is no possibility to choose different software technologies for individual SM development, so engineers are forced to make a final selection of technologies at the beginning of the development process. Sometimes, such decisions, which were made in the past, may turn out wrong after the years of development, leading to a great waste of time and even to the project failure. The configuration of MTA must be done at the level of the whole application rather than at the level of an individual SM. Therefore, there is a great possibility that an individual SM requires a usage of certain software libraries which are incompatible with the libraries in other SMs. Nevertheless, there are specialized modularization techniques and frameworks, for some programming languages, that can be used to overcome MTA configuration challenges. For example, Open Service Gateway Initiative (OSGi) [4] is the Java programming language framework which can be used for developing modular SMs within MTA. Horizontal scaling of an MTA must be done at the application level also, without the opportunity to scale individual SMs. Nevertheless, these types of applications are usually scaled vertically by increasing the infrastructure resources such as processing power and memory [5]. Thus, resources of an execution platform infrastructure cannot be adjusted in accordance with the requirements of individual SMs. Accordingly, engineers are forced to build "one size fits all" execution platforms, which result in irrational resource consumption and maintenance cost increase [6]. Deployment of MTA implies procedures for building, packaging and installation of the complete MTA, without a possibility to deploy individual SMs [7].

On the other hand, MSA was introduced as a suite of loosely coupled SMs, called microservices [8]. Each microservice exists and runs within a separate operating system process, independently from the other microservices. A microservice has well defined set of responsibilities and functionalities exposed through its application programming interface (API) [9]. Accordingly, engineers are able to group up into development teams in charge of developing different microservices, choosing technologies vendors and technical

The research presented in this paper was supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia, Grant III-44010

characteristics which are the most suitable for their needs. MSA testing comes down to testing of individual microservices, which eases locating bugs and bug fixing. The configuration of an MSA can be done separately for each microservice, eliminating possibilities for software library incompatibility. Horizontal scaling becomes a natural procedure to increase the MSA availability and it is done by running additional instances of a required microservices [10]. Since microservices are independent SMs, the adaptation of infrastructure resources to the needs of the individual microservices becomes easier and resource consumption becomes more rational [11]. MSA deployment procedure implies building, packaging and installation of the individual microservices, rather than the whole ecosystem. Thus, multiple versions of the same microservice can be run in order to compare them in production.

The MSA-specific infrastructure and the large number of microservices introduce several challenges to:

- MSA modeling, as there is a need for the MSA modeling framework which should provide a formal modeling technique and modeling tool which will ensure a higher abstraction viewpoint to engineers while specifying microservice business logic, microservice API, microservice configuration and inter-microservice communication patterns. The usage of such a modeling framework should decrease the ecosystem complexity in early phases of MSA specification, while enabling the usage of the MSA model specifications in the later development and deployment stages;
- MSA development, as there is a need for the implementation of mechanisms and infrastructure for: user request acceptance and routing, microservice auto-discovery and registry, microservice frontend and backend load-balancing, microservice fault-tolerance and health check; and
- MSA deployment, in regard to provisioning automated procedures for the MSA ecosystem building, packaging, monitoring, horizontal scaling and installation to the dedicated or cloud execution platforms.

Since the MSA approach has become dominant in the past several years [12], [13], there are plenty of development frameworks introduced by the large software companies and the open-source software community [14]. These frameworks address the majority of the aforementioned development and deployment challenges by introducing well-defined software libraries which wrap-up the core functionality of a framework. On the other hand, the usage of the aforementioned frameworks requires redundant program and configuration code to be written for different layers of the MSA ecosystem. Since the MSA ecosystem usually comprises a lot of microservices, this can lead to mistakes as engineers unintentionally introduce errors to repetitive code constructs. Also, workflow procedures used for ecosystem deployment need to be written repeatedly for individual microservices within the MSA ecosystem. Such a procedure development is often harder for average engineers and needs to be done by engineers which are specialized for the MSA deployment tasks.

Therefore, the first goal of this research is to provide a formal procedure for MSA specification in order to address the MSA modeling challenges. The second goal is to address the development and deployment challenges by using the MSA model specification for generation of program, configuration and infrastructure program code constructs. In this way, first, we want to ease the usage of MSA development frameworks and to generate all the repetitive code constructs in order to eliminate potential errors. Second, we want to generate all required procedures for the MSA ecosystem deployment in order to ease this process, make it less dependent from the specialized engineering teams and therefore less time consuming.

In order to achieve the aforementioned goals, it could be beneficial for engineers to have a domain-specific language (DSL) which will provide a formal technique for MSA modeling, as well as a set of code generators which will generate all the aforementioned artifacts based on a MSA specification written in the DSL. In order to enable usage of a DSL and code generators, we plan to develop a model-driven software tool which will support this approach.

Apart from introduction and conclusion, this paper is divided into 3 sections. In Section 2, we discuss in detail all of the challenges caused by the large number of microservices within MSA and MSA-specific infrastructure. We also propose a model-driven approach as a possible solution to these challenges. In Section 3, we present our previous research efforts to the MSA establishment, alongside the plan of research activities that should lead to its improvement and implementation of a model-driven tool which will support the realization of such an approach. In Section 4, we give an overview of related works.

II. MSA Ecosystem Establishment Challenges

In this section, we present challenges that engineers typically face during modeling, development and deployment of the MSA ecosystem. We also propose a model-driven approach as a possible solution to these challenges.

Since MSA was introduced in 2011 [15], this software development style has become the fundamental for many engineers [16] as it overcomes the most of the challenges encountered in the monolithic software application development. Large companies have adopted MSA and developed many open-source frameworks which are constantly maintained and improved by the large MSA developer community. For example, An American media streaming service Netflix has developed a set of open-source frameworks called Netflix OSS [17]. Netflix OSS was used by Netflix to divide their monolithic software system into a MSA. Today, Netflix software system consists of over 900 microservices [18]. Currently, the MSA development frameworks are introduced in almost all mainstream programming languages [14], with a strong development and maintenance support by the community.

On the other hand, MSA have introduced the new challenges to software development process, particularly caused by the large number of microservices within the MSA ecosystem and MSA-specific infrastructure features which are required for the ecosystem establishment.

A. MSA Modeling Challenges

The first challenge is related to the MSA modeling process. Usually, at the beginning of the MSA development, it is hard for engineers to have the complete overview over the individual layers of the MSA ecosystem. The situation is even more complex if existing monolithic software is required to be migrated to MSA. Therefore, engineers are trying to decrease the MSA ecosystem complexity by specifying different types of MSA models. These models usually comprise microservice business entity models, microservice API models, inter-microservice communication pattern models and deployment strategy specification. Thus, MSA models are often specified by using an informal modelling techniques and, at the end, used just for documentation purposes. On the other hand, Model-Driven Software Engineering (MDSE) practitioners argue in favor of models as a formal way to describe the entities from the specific domain and use of such specifications as primary artifacts in the development process [19]. Domain entities, their attributes and relationships are described in a form of a meta-model which represents the abstract syntax of a DSL [20]. In order to use such a DSL for the specification of meta-model concept instances, called models, a concrete DSL syntax must be developed [20]. Therefore, in order to address MSA modeling challenges, the application of MDSE should introduce a DSL as a formal way for the MSA ecosystems modeling. The MSA DSL should provide an abstraction level which is high enough to decrease MSA modeling complexity, but which provides enough information that can be used for automation of the MSA development and deployment process. In order to use the MSA DSL in practice, a model-driven tool should be developed. Such a tool should provide a SM which will support the usage of the MSA DSL concrete syntax, used for the MSA model specification (MDM), and an appropriate file format for the MDM storage and representation.

B. MSA Development Challenges

The second challenge is related to the MSA development process. After the end of the MSA modelling process, usually begins the MSA development process which consists of: (i) development of the user-defined microservice (UMS) layer, i.e. microservices which implement the MSA ecosystem business logic; (ii) development of the infrastructure microservice (IMS) layer, i.e. microservices which ensure accessibility, availability, durability and monitoring of individual microservices within the UMS layer; and (iii) development of the inter-microservice communication patterns (MSC), i.e. selection and implementation of microservice communication patterns and the message distribution infrastructure.

The first step in the UMS layer development comprises configuration of individual microservices, including: (i) specification of the UMS API settings, such as microservice name, host name, port number and data persistence layer; and (ii) specification of a software library list, necessary for using the chosen software framework. Therefore, UMS which use the same technology stack, have common configuration properties with specific values for each UMS. As engineers try to reduce the development time, by copying repeatable configuration code to the different UMS specifications, they are unintentionally introducing errors by skipping values for common configuration parameters. For example, UMS name misconfiguration can cause microservice auto-discovery and registry inconsistent behavior within the IMS layer which is usually hard to understand and debug. Also, data persistence layer misconfiguration can cause the inconsistency and data collisions for UMS using the common database management systems. In this case, engineers usually forget to change database connection profile settings for database-specific object names. In such a situation, different UMS can try to use the same database objects, such as database tables, for storing different business model objects, or different UMS try to create their own database objects with the same name. Further, since there is a certain set of programming libraries which are required for the usage of a chosen MSA development framework, engineers easy forget some of them or misconfigure their versions. This type of the UMS misconfiguration leads to unintuitive error messages in runtime and results in a great waste of time. Accordingly, it can be beneficial for engineers if configuration and technology stack settings can be specified during the MSA modeling process, within a single MDM specification. In this way, first, engineers are able to write an in-place UMS configuration specifications without writing any boilerplate or redundant code. Second, engineers do not need to specify individual software libraries within the MDM. It is enough to specify which development framework they want to use and that is enough information which software libraries need to be included in the UMS configuration. Thus, such a MDM specification further can be used as an entry artifact for the generation of configuration code required for the individual UMS.

The second step in the UMS development is the specification of the UMS business layer (BL) which comprises the UMS business entity models and implementation of the UMS business logic. The UMS business logic functionalities are exposed to the end user or the other UMS in a form of the UMS API. The UMS API is typically developed applying the REST API design principles [21] and using the HTTP application protocol [22]. The UMS REST API method specifications have a similar structure, depending on the REST method type [21], but with parameters specific to the individual UMS. The "copy-paste" problem is even more conspicuous in this case, because developers usually forget to change microservice-specific API settings, such as the REST method name, type or HTTP request content type for example. Therefore, it can be beneficial if engineers could use a DSL in order to specify the BL API within the same MDM, avoiding the need for the repetitive code constructs and potential mistakes. Such a specification can be then used in order to generate the BL API program code templates for the chosen technology stack. The generated code templates then can be manually filled out with program code which implements the concrete business logic for the specific UMS API.

In order to resolve the aforementioned UMS development challenges in practice, a model-driven tool should provide a separate SMs, which can be used for the UMS configuration and business logic code generation, using a MDM as its input.

The IMS layer is the heart of the MSA ecosystem as it provides the following infrastructure features:

1. user request acceptance and routing, i.e. exposing a unified access interface and a single entry point to the whole MSA ecosystem,
2. microservice auto-discovery and registering, i.e. providing a single point for microservice instances monitoring and microservice name, host and port registry,
3. frontend load-balancing, i.e. providing an improvement of workload distribution during the inter-microservice communication,
4. backend load-balancing; i.e. providing an improvement of workload distribution for incoming user requests across the MSA ecosystem,
5. microservice fault tolerance and circuit-breaking, i.e. providing a mechanism for microservice failure resistance,
6. the MSA ecosystem monitoring, i.e. providing procedures for acquisition and presentation of the MSA ecosystem metrics of interest, and
7. the MSA ecosystem scaling, i.e. increasing the availability of the ecosystem by provisioning the additional microservice instances in the UMS layer.

According to the aforementioned features, we can argue that implementation of the IMS layer is crucial to the MSA ecosystem establishment in practice. Depending on a chosen technology stack, there are different requirements which are not so trivial to fulfill and require repetitive procedures to be performed for each of the microservices from the UMS layer. Thus, configuration of the IMS layer depends on the configuration parameters of the individual microservices from the UMS layer, such as microservice names, host addresses and port numbers. Since the record about the aforementioned setting can be obtained from the MDM specification, automation of the IMS configuration and development can be achieved. Such and automation can reduce engineering efforts and radically decrease development time since the infrastructure microservice development requires knowledge of framework specifics, which depends on a chosen technology stack. For example, in order to enable microservice auto-discovery for the newly specified UMS, engineer can set one additional parameter within the existing MDM specification. This parameter can be a Boolean flag which determines if certain UMS should be added to the IMS auto-discovery settings. On the other hand, in order to achieve the same goal using the Netflix OSS framework, for example, engineer needs to write program and configuration code separately for all, the newly created UMS and the auto-discovery microservice from the IMS layer. Therefore, it could be beneficial if another code generator module could be built within a model-driven tool. This module should be dedicated to generation of configuration and program code for the IMS layer, so no significant manual and repeatable configuration or development is needed.

The development of the MSC layer implies specification and development of the communication patterns which enable inter-microservice communication and message exchange. Synchronous inter-microservice communication happens when microservice which initiates communication (client) consumes the functionality of the other microservice (server) using its API [23]. The client microservice API is blocked while it waits for the server microservice API to answer. On the other hand, asynchronous communication is done through messages sent to mediator (message provider) rather than directly to the server microservice [23]. The Client microservice API is not blocked while waiting for the server microservice API answer. In situations when a large number of microservices exist within the ecosystem, it is hard for engineers to have a clear overview on the individual microservice communication links, their type, message format and message content. This is especially pronounced in early phases of the MSA ecosystem establishment, when engineers have not developed individual microservice APIs yet, but they have to specify how microservices will communicate and which type of communication technique they will use in order to determine microservice roles and responsibilities. Therefore, the usage of a DSL can be beneficial in this situation since it can provide higher abstraction level for the specification of the inter-mircoservice communication templates, avoiding the need for the complete MSA API existence. Thus, such a specification can provide enough information that can be used to generate program code templates which implement the basic nutshell for communication infrastructure, business rules, and message format. The separate code generator module should be developed within a model-driven tool in order to generate required code templates using the MDM as its entry artifact. Later, as a MSA development moves on, these code templates should be filled-out with program code which implements required communication business roles and the message content.

C. MSA Deployment Challenges

The third challenge to MSA establishment is related to the MSA deployment process which comprise: (i) the MSA ecosystem building; (ii) the MSA ecosystem packaging; and (iii) the MSA ecosystem installation to the target execution platform.

The MSA building procedure differs depending on the chosen technology stack. Building procedure utilize a set of commands which need to be executed over the microservice program code, using program code build engine. For example, if Java and Netflix OSS are chosen, then Maven [24] or Gradle [25] build engines could be used for ecosystem building. Anyhow, the procedure is the same and repeatable for each microservice utilizing the same technology stack, no matter if it belongs to the UMS or the IMS layer.

The MSA packaging procedure depends on the chosen technology stack, as well as on the target execution platform type and configuration. For example, if Java and Netflix OSS are chosen, microservices could be packaged to JAR (Java Archive) [26] files, or could be packaged in a form of the Docker image in order to be run as isolated Docker containers [27]. However, the packaging procedure is also repeatable for microservices which share common packaging settings.

The MSA ecosystem installation to target execution platform comprise the specification of a blueprint which describes the structure of the MSA ecosystem and its desired state. For example, if an Amazon Web Service (AWS) and Docker packaging are chosen, the “Dockerrun.aws.json” file needs to be specified [28]. This file typically comprises specification of an individual microservice names, hosts, ports, storage volume settings, allocation of infrastructure resources and path to repository which keeps microservice Docker images.

Based on what was previously stated, it is obvious that an engineer needs to be familiar with many different fields of software engineering in order to complete the MSA deployment tasks. In practice, separate teams of engineers are dedicated to these tasks. However, it could be beneficial if the MSA deployment could be automated to certain extent. This can reduce the time needed for such a procedure development, and enable engineers from other teams to be less dependent on the deployment team. This is particularly important in situations when the MSA ecosystem, or some parts of it, needs to be deployed on different type of execution platforms [29]. In this case, it is crucial for the deployment procedure to be flexible and adaptive in order to provide rapid MSA migration and reduce the time needed for its customization.

In practice, MSA building, packaging and deployment procedures usually comprise well defined set of steps which mutually stem from one another. For example, if the MSA ecosystem is developed using the Java programming language and the Netflix OSS framework, then Maven can be used for a MSA building, Docker containers can be used for a MSA packaging and AWS can be used as target execution platform. In order to develop deployment procedure which supports the aforementioned technology stack and target execution platform, engineers need general microservice settings such as microservice name, host name, port number, desired number of microservice replicas, amount of memory that needs to be reserved for the microservice and so on. All these settings then need to be packed within the Dockerrun.aws.json file, so MSA is able to be installed to the AWS instance and to work correctly. Thus, engineers need to be familiar with specific format and individual settings of target execution platform blueprint. Therefore, there is an opportunity to build deployment templates, for different execution platforms, which consists of common configuration parameters with specific values for individual microservices. Further, using a DSL engineers do not need to be familiar with all the configuration parameters from specific deployment templates. Engineers just need to specify build engine, packaging strategy and target execution platform names as individual parameter within the same MDM specification. Accordingly, code generators can use the aforementioned general microservice settings from the MDM in order to fill-out the appropriate deployment script and blueprint setting parameters. Thus, changes in technology stack or target execution platform type require minor interventions in the MDM specification, re-generation and re-execution of deployment procedures in order to apply these changes in production.

In order to support the aforementioned MSA deployment requirements in practice, a separate code generator module within a model-driven tool can be developed. This module should use the MDM specification as its input and generate all required deployment procedures on the output.

III. Model-Driven Tool Prototype And Research Activity Plan

During our previous research [30] we have developed MicroBuilder, a model-driven tool for the specification of software applications that follow Representational State Transfer (REST) microservice software architecture design principles. MicroBuilder comprises two modules: (i) MicroDSL, a module that provides a DSL used for the specification of the REST microservice software architecture, and (ii) MicroGenerator, a module which comprises a set of code generators which implement series of model-to-text transformations (M2T). The M2T transformations are used to generate executable program and configuration code based on the model specification made using MicroDSL. We have supported generation of the Java program code for the implementation of the UMS layer and REST-based synchronous inter-microservice communication. We also generate the UMS configuration code with no manual configuration needed. Talking about UMS business logic code generation, we generate the Java program code for: (i) implementation of the UMS business models; and (ii) create, update and delete (CRUD) operations for data manipulation over a business models. For generation of custom business logic, we generate API templates which should be manually filled with program code by engineers. We have also supported the generation of the Java program code which applies the Netflix OSS framework in implementation of the IMS layer. For the MSA ecosystem monitoring, we have used Netflix Turbine [31], for acquisition of the MSA ecosystem metrics, and Spring Cloud Dashboard [32] for metric visualization.

We have also presented a detailed case study where we have used the MicroBuilder tool in order to establish the web shop MSA. The structure of generated Java code is also discussed in order to explain all benefits of the MicroDSL language usage. We have compared the number of lines of code needed to specify the web shop MSA using MicroDSL to the number of manually written lines of code needed to specify the same MSA.

In order to understand MicroBuilder strengths and shortcomings, we have performed the evaluation of the MicroBuilder tool. We have applied two types of evaluation approach: (i) evaluation by example in which have used the MicroBuilder tool in order to specify various real-world examples of the microservice software architectures in order to iteratively improve the MicroDSL language and code generators; and (ii) evaluation by questionnaire in which we were using a series of questions in order to perform an objective assessment of the MicroBuilder tool. Based on the obtained results we have concluded that MicroDSL satisfies the following DSL quality characteristics: functional stability, usability, reliability, expressiveness, and productivity.

To develop the MicroBuilder tool, we have used Eclipse Modelling Framework (EMF) [33]. The MicroDSL abstract syntax concepts conform to Ecore meta-meta-model [34]. The MicroDSL textual concrete syntax was developed using the Xtext framework [35], while graphical concrete syntax was developed using the Sirius framework [36]. Individual code generators within the MicroGenerator module were developed using the Xtend framework [37].

Since challenges related to the MSA deployment and asynchronous inter-microservice communication were not considered during the aforementioned research, in the research proposed in this paper we plan to: (i) extend the MicroDSL meta-model in order to enable specification of missing MSA concepts and settings; (ii) implement the new code generators which will generate required programcode; and (iii) improve the existing code generators in order to support the additional MSA development languages and frameworkds

In order to support the asynchronous inter-microservice communication, we plan to extend the MicroDSL meta-model by adding concepts and attributes which will be used for the specification of MSA events and event messages. Events will comprise a list of event messages used for relevant data exchange between the microservices.

In order to support automated deployment procedures, we plan to add concepts describing basic building, packaging and installation strategies within the MSA core concept. We also plan to add the container concept, as additional microservice resource type, in order to support the MSA container configuration and packaging.

In Figure 1 we present the architecture of the model-driven tool prototype which comprises two main modules: the MsaDSL module and the MsaCodeGen module. The MsaDSL module will provide a DSL which will be the improved version of the MicroDSL language. The new version of a DSL should provide additional concepts for specification of: (i) asynchronous inter-microservice communication patterns; and (ii) building, packaging and installation settings.

The MicroGenerator module [30] will be transformed to the MsaCodeGen module and divided into to several submodules:

1. the MsaUMS submodule, used for the generation of program and configuration code which implements the UMS layer. MsaUMS supports the generation of the Java executable program code,
2. the MsaIMS submodule, used for the generation of program and configuration code which implements the IMS layer. MsaIMS supports the generation of the Java executable code which utilizes the Netflix OSS framework,
3. the MsaIMC submodule, used for the generation of program code which implements the synchronous and asynchronous inter-microservice communication patterns. For synchronous communication patterns, the Java program code which uses the Hypertext Transfer Protocol (HTTP) communication protocol is generated. For asynchronous communication MsaIMC will support generation of the Java program code which implements the Apache Kafka message provider [38], and
4. the MsaDPY submodule which will provide a set of code generators for generation of program code which implements the MSA deployment procedures. We plan to support generation of provisioning scripts for the IBM Cloud Container services [39] and the AWS EC2 Multicontainer Docker Environments [40]. We also plan to use the Netflix Spinnaker [41] platform to support the MSA ecosystem continuous integration and continuous

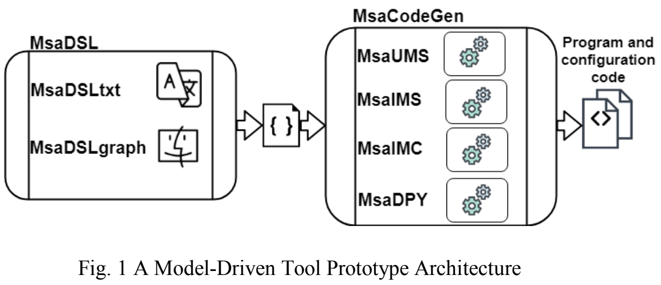

Fig. 1 A Model-Driven Tool Prototype Architecture

delivery.

IV. Related Work

While surveying the state-of-the-art literature in this area, we have found several research papers that deal with the specification of different MSA layers, using the MDSE approach. In the rest of the section we discuss the individual approaches and compare them with our approach.

In [42], the authors present an automated approach for the selection and configuration of cloud providers for multi-cloud microservices-based applications. They have developed a DSL which can be used for the specification of the application's multi-cloud requirements. Authors also provide a systematic method for obtaining proper configurations that comply with the application's requirements and the cloud providers' constraints. Comparing to our approach, authors were focused just on one aspect of the MSA ecosystem deployment, which refers to specification of installation settings for different cloud providers. On the other hand, their approach provides an opportunity for more fine-grained specifications, since they have developed a DSL that is used just for this particular use-case.

In [43], the authors try to answer the question if and to what extent MSA might build upon existing findings of Service-Oriented Architecture (SOA) research. They try to find the answer to the aforementioned question in the area of Model-driven Development (MDD), whose application to SOA has been intensively studied. The presented meta-model is divided into the three viewpoints Data, Service and Operation, each of which encapsulates concepts related to a certain aspect of MSA. The meta-model aims to support DevOps-based MSA development and automatic transformation of meta-model instances into MSA implementations. Comparing to our approach, the authors were focused on the MSA meta-model development by utilizing the deduction procedure based on the several SOA modeling approaches with the goal to identify the modeling concepts which can be used for MSA specification. Therefore, the main goal of the aforementioned research is more related on the MSA meta-model specification procedure, rather than to the MSA ecosystem establishment in practice. Nevertheless, the presented meta-model and approach seems to be still a work-in-progress towards a tool which can be used in practice.

In [44], the authors present the Aji Modeling Language (AjiL) which can be used for the MSA ecosystem specification. The AjiL abstract syntax was derived from several public MSA examples and is depicted as a Unified Modeling Language (UML) class diagram. The AjiL graphical concrete syntax was developed using the Sirius framework. Comparing to our approach the aforementioned authors have developed a DSL which can be used for the basic specification of MSA. There is still no support for specification of inter-microservice communication patterns and deployment settings. Nevertheless, we have utilized the similar set of techniques and technologies for the specification and development of a DSL concrete syntax.

V. Conclusion

In this paper we argue in favor of MDSE utilization in resolution of challenges related to the MSA ecosystem establishment in practice. We propose a DSL as a formal technique for the MSA ecosystem modeling in order to: (i) decrease the system complexity in early phases of the MSA ecosystem development; and (ii) use such a formal specification as entry artifact to process of the MSA program code generation.

Our goal is to improve our model-driven approach established during previous research efforts [30] in order to address all remaining challenges related to MSA modeling, development and deployment.

In order to achieve this goal, we plan to improve MicroBuilder, a model-driven tool which we have developed during our previous research [30]. MicroBuilder has addressed the majority of challenges related to MSA modeling and development, including the automation of the UMS and the IMS layer development and the REST-based synchronous inter-microservice communication specification. In order to address the challenges related to the MSA deployment and asynchronous inter-microservice communication, first, we plan to extend the MicroDSL meta-model with additional concepts, attributes and constraints. We also need to update textual and graphical concrete syntax specifications in order to support the new concepts. Second, we plan to improve existing code generators and build new ones in order to support the generation of missing program code constructs. In this way, we want to fulfill all the prerequisites, so the new version of the MicroBuilder tool can be used for MSA establishment in practice.

Since we have supported the generation of the Java program code which uses the Netflix OSS framework, in our future research we plan to extend technology stack by implementing addition code generators for other programming languages and frameworks. Since there is an effort [45] in development and improvement of the Netflix OSS framework for the Node.js language [46], we plan to develop code generators which will support the Node.js code generation. Also, we plan to support the usage of Zookeeper [47] as an alternative for the Netflix Eureka [48].

After the completion of the model-driven tool, which we propose in this research, we expect it to be used by software engineers in real-world projects. The tool can be used in order to develop the MSA ecosystem from scratch and deploy it to different production environments. On the other hand, the tool can be also used in situations when existing MTA should be migrated to MSA. Anyhow, the usage of the proposed model-driven tool should ease the process of the MSA establishment in production and significantly reduce development time and engineering effort.

REFERENCES

[1] A. Balalaie, H. Abbas, and J. Pooyan. "Migrating to cloud-native architectures using microservices: an experience report," In European Conference on Service-Oriented and Cloud Computing, pp. 201-215. Springer, Cham, 2015.

[2] A. Levcovitz, R. Terra, and M. Tulio Valente. "Towards a technique for extracting microservices from monolithic enterprise systems," arXiv preprint arXiv:1605.03175 (2016).

[3] J.P. Gouigoux, and D. Tamzalit. "From Monolith to Microservices: Lessons Learned on an Industrial Migration to a Web Oriented Architecture," In Software Architecture Workshops (ICSAW), 2017 IEEE International Conference on, pp. 62-65. IEEE, 2017.

[4] "Open Services Gateway initiative" [Online], Available: https://en.wikipedia.org/wiki/OSGi [Accessed: 27-Jun-2018].

[5] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil. "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud," In Computing Colombian Conference (10CCC), 2015 10th, pp. 583-590. IEEE, 2015.

[6] M. Villamizar, O. Garces, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas. "Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures," In Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on, pp. 179-182. IEEE, 2016.

[7] E. Daniel, D. Cárdenas, R. Amarillo, E. Castro, K. Garcés, C. Parra, and R. Casallas. "Towards the understanding and evolution of monolithic applications as microservices," In Computing Conference (CLEI), 2016 XLII Latin American, pp. 1-11. IEEE, 2016.

[8] N. Dragoni, G. Saverio, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina. "Microservices: yesterday, today, and tomorrow," In Present and Ulterior Software Engineering, pp. 195-216. Springer, Cham, 2017.

[9] T. Johannes. "Microservices." IEEE Software 32, no. 1 (2015): 116-116.

[10] N. Dragoni, I. Lanese, S. T. Larsen, M. Mazzara, R. Mustafin, and L. Safina. "Microservices: How to make your application scale," In International Andrei Ershov Memorial Conference on Perspectives of System Informatics, pp. 95-104. Springer, Cham, 2017.

[11] D. S. Linthicum. "Practical use of microservices in moving workloads to the cloud." IEEE Cloud Computing 3, no. 5 (2016): 6-9.

[12] K. Bakshi. "Microservices-based software architecture and approaches," In Aerospace Conference, 2017 IEEE, pp. 1-8. IEEE, 2017.

[13] J. Bogner, and A. Zimmermann. "Towards integrating microservices with adaptable enterprise architecture," In Enterprise Distributed Object Computing Workshop (EDOCW), 2016 IEEE 20th International, pp. 1-6. IEEE, 2016.

[14] "Awesome Microservices" [Online], Available: https://github.com/mfornos/awesome-microservices. [Accessed: 03-Jun-2018].

[15] "Microservices Martin Fowler and James Levis" [Online], Available: https://www.martinfowler.com/articles/microservices.html [Accessed: 03-Jun-2018].

[16] G. Kecskemeti, A. C. Marosi, and A. Kertesz. "The ENTICE approach to decompose monolithic services into microservices." In High Performance Computing & Simulation (HPCS), 2016 International Conference on, pp. 591-596. IEEE, 2016.

[17] "Netflix OSS" [Online], Available: https://netflix.github.io/ [Accessed: 03-Jun-2018].

[18] "Adopting Microservices at Nerflix" [Online], Available: https://netflix.github.io/ [Accessed: 03-Jun-2018].

[19] I. Lukovic, S. Ristic, S. Aleksic, A. Popovic. "An application of the MDSE principles in IIS* Case," Model Driven Software Engineering-Transformations and Tools (2008): 85.AS

[20] J. Porubän, M. Sabo, J. Kollár, and M. Mernik. "Abstract syntax driven language development: Defining language semantics through aspects." In Proceedings of the International Workshop on Formalization of Modeling Languages, p. 2. ACM, 2010.

[21] A. Rodriguez. "Restful web services: The basics. IBM developerWorks". 2008 Nov 6:33.

[22] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. "Hypertext transfer protocol--HTTP/1.1". No. RFC 2616. 1999.

[23] "Communication between the microservices" [Online], Available: https://dzone.com/articles/communicating-between-microservices [Accessed: 03-Jun-2018].

[24] "Maven" [Online], Available: https://maven.apache.org/ [Accessed: 03-Jun-2018].

[25] "Gradle" [Online], Available: https://gradle.org/ [Accessed: 03-Jun-2018].

[26] "JAR file" [Online], Available: https://docs.oracle.com/javase/tutorial/deployment/jar/basicsindex.html [Accessed: 03-Jun-2018].

[27] "Docker" [Online], Available: https://www.docker.com/what-docker [Accessed: 03-Jun-2018].

[28] "Aws Multicontainer Docker Configuration" [Online], Available: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_docker_v2config.html [Accessed: 03-Jun-2018].

[29] C. Esposito, A. Castiglione, and K.-K Raymond Choo. "Challenges in delivering software in the cloud as microservices." IEEE Cloud Computing 3, no. 5 (2016): 10-14.

[30] B. Terzić, V. Dimitrieski, S. Kordić, G. Milosavljević, and I. Luković. "Development and evaluation of MicroBuilder: A Model-Driven tool for the specification of REST Microservice Software Architectures," Enterprise Information Systems (2018): 1-24.

[31] "Netflix Turbine" [Online], Available: https://github.com/Netflix/Turbine [Accessed: 03-Jun-2018].

[32] "Spring Cloud Dashobard" [Online], Available: https://github.com/VanRoy/spring-cloud-dashboard [Accessed: 03-Jun-2018].

[33] "EMF" [Online], Available: http://www.eclipse.org/modeling/emf/ [Accessed: 03-Jun-2018].

[34] "Ecore" [Online], Available: http://download.eclipse.org/modeling/emf/emf/javadoc/2.7.0/org/eclipse/emf/ecore/package-summary.html [Accessed: 03-Jun-2018].

[35] Eysholdt, Moritz, and Heiko Behrens. "Xtext: implement your language faster than the quick and dirty way." In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, pp. 307-309. ACM, 2010.

[36] "Sirius" [Online], Available: https://www.eclipse.org/sirius/ [Accessed: 03-Jun-2018].

[37] "Xtend" [Online], Available: http://www.eclipse.org/xtend/ [Accessed: 03-Jun-2018].

[38] "Apache Kafka" [Online], Available: https://kafka.apache.org/ [Accessed: 03-Jun-2018].

[39] "IBM Container Service" [Online], Available : https://www.ibm.com/cloud/container-service [Accessed: 03-Jun-2018].

[40] "Aws Multicontainer Docker Configuration" [Online], Available: https://www.ibm.com/cloud/ [Accessed: 03-Jun-2018].

[41] "Netflix Spinnaker" [Online], Available: https://www.spinnaker.io/ [Accessed: 03-Jun-2018].

[42] G. Sousa, W. Rudametkin, and L. Duchien. "Automated setup of multi-cloud environments for microservices applications," In Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on, pp. 327-334. IEEE, 2016.

[43] F. Rademacher, J. Sorgalla, S. Sachweh, and A. Zündorf. "Towards a Viewpoint-specific Metamodel for Model-driven Development of Microservice Architecture," arXiv preprint arXiv:1804.09948 (2018).

[44] J. Sorgalla. "Ajil: A graphical modeling language for the development of microservice architectures," In Extended Abstracts of the Microservices 2017 Conference. 2017.

[45] "Slaying Monoliths at Netflix with Node.js" [Online], Available: https://www.linux.com/news/event/nodejs/2017/3/slaying-monoliths-netflix-nodejs/ [Accessed: 03-Jun-2018].

[46] "Node.js" [Online], Available: https://nodejs.org/en/ [Accessed: 03-Jun-2018].

[47] "Apache Zookeeper" [Online], Available: https://nodejs.org/en/ [Accessed: 03-Jun-2018].

[48] "Nerflix Eureka" [Online], Available: https://github.com/Netflix/eureka [Accessed: 03-Jun-2018].