Towards Blended Modeling and Simulation of DevOps Processes: The Keptn Case Study

Alessandro Colantoni
Luca Berardinelli
Antonio Garmendia
name.surname@jku.at
Institute of Business Informatics - Software Engineering
Johannes Kepler University
Linz, Austria

Johannes Bräuer
Dynatrace Gmbh
Linz, Austria
johannes.braeuer@dynatrace.com

ABSTRACT

DevOps and Model Driven Engineering (MDE) provide differently skilled IT stakeholders with methodologies and tools for organizing and automating continuous software engineering activities and using models as key engineering artifacts. JSON is a popular data format, and JSON Schema provides a general-purpose schema language for JSON. This paper presents our work in progress on blended modeling and scenario simulation of continuous delivery pipelines as executable JSON-based models. For this purpose, we show a case study based on Keptn, an open source tool for DevOps automation of cloud-native applications, and its language, Shipyard, a JSON-based process language for continuous delivery pipeline specification.

CCS CONCEPTS

• **Software and its engineering → Syntax**; **Semantics**; **Interpreters**; **Integrated and visual development environments**; **Domain specific languages**.

KEYWORDS

DevOps, MDE, blended modeling, simulation

ACM Reference Format:
Alessandro Colantoni, Luca Berardinelli, Antonio Garmendia, and Johannes Bräuer. 2022. Towards Blended Modeling and Simulation of DevOps Processes: The Keptn Case Study. In *ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS '22 Companion), October 23–28, 2022, Montreal, QC, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3550356.3561597

1 INTRODUCTION

Over the last decade, DevOps methods and tools have been successfully implemented and adopted by companies to boost automation and the efficiency of the engineering process.

The momentum of DevOps resulted in a flourishing of technological solutions to meet the huge market demands [1]. This rapid evolution resulted in a varied landscape of tools [11, 20] for supporting activities of *Continuous-software Engineering* (CSE) [30] processes. Consequently, the choice of the *right* set of DevOps tools and their integration into frameworks for supporting potentially arbitrary and customizable activities of an engineering process is a practical problem faced by DevOps engineers.

In this regard, Bordeleau et al. [8] investigated and elicited sets of requirements for DevOps frameworks. Among them, they also consider the need for adequate support for modeling DevOps engineering *processes*, of the *product* resulting from the process, i.e., the software system, and *resources* (e.g., tools) involved in the accomplishment of development and operations phases.

To contribute to the above requirements, we proposed in [15] DevOpsML, a tool-supported [43] model-driven approach to model DevOps frameworks as a combination of processes and tools. We chose SPEM [44] as standard and tool-independent process modeling language for process modeling.

However, due to the varied and continuously evolving landscape of DevOps solutions [11], it is likely that different DevOps tools may provide their approach defining processes to accomplish DevOps engineering tasks. In this paper, as a representative DevOps tool, we choose Keptn [32], an open-source control plane for orchestrating continuous delivery (CD) and operational processes of cloud-based applications. Thanks to ongoing collaborations [22, 29], Keptn is often chosen as the industrial use case for our research endeavors. In particular, Keptn provides a declarative and event-based integration approach to tame the complexity of DevOps tool integration. For this purpose, Keptn relies on a set of JSON documents [35] validated against a set of given JSON schemas [37].

In [16], we proposed a semi-automated approach, namely JSONSchemaDSL, for generating a modeling workbench for JSON-based domain-specific modeling languages (DSL) atop Eclipse Modeling Framework (EMF) and related technologies [23, 26–28, 31]. JSONSchemaDSL supports the (meta)modeling of JSON schemas and their instances. In [17], we adopted JSONSchemaDSL to realize a continuous consistency checking mechanism for JSON documents used in Keptn.

MODELS '22 Companion, October 23–28, 2022, Montreal, QC, Canada

[1] https://www.gartner.com/en/newsroom/press-releases/2015-03-05-gartner-says-by-2016-devops-will-evolve-from-a-niche-to-a-mainstream-strategy-employed-by-25-percent-of-global-2000-organizations

In this paper, we further contribute to applying MDE techniques and practices for DevOps [7] using Keptn as a reference tool for our case study. The main contributions of this paper are

- Blended modeling [12, 18] of CD pipelines combining the native JSON textual notation and a dedicated graphical one. A new Sirius-based [26] graphical editor is under development on purpose
- Simulation of CD scenarios. An executable operational semantics [9] is implemented in the GEMOC Studio language workbench [31]. The events collected in the scenario model drive the simulation of DevOps stages and steps of CD pipelines as designed and executed in Keptn.

The contributions of this paper leverage the model-driven JSONSchemaDSL approach presented in [16] and combine it with the capabilities offered by the GEMOC Studio for designing JSON schema-based executable domain-specific modeling language (xDSL).

The rest of the paper is organized as follows. Section 2 sets the background. Section 3 introduces our bridging approach for blended modeling and simulation of DevOps processes. Section 4 presents a case study and a preliminary evaluation of our approach. Section 5 discusses related work, and finally, Section 6 concludes the paper.

2 BACKGROUND

This section briefly introduces MDE, blended modeling, JSON and JSON Schema, a.k.a. JSONware, our approach JSONSchemaDSL, and the Keptn tool.

2.1 Model-Driven Engineering

Model-driven engineering (MDE) [10] is an engineering methodology that relies on models as purposeful abstractions of complex (software) systems, which are manipulated during engineering activities and, possibly, throughout the whole system life-cycle for the sake of the highest possible process automation. *Model(ing)* and *metamodel(ing)* [9] are pillar concepts and activities in MDE. A *metamodel* defines modeling concepts and their relationships and provides the intentional description of all possible models, which, in turn, must conform to the associated metamodel. A metamodel represents the *abstract syntax* of a *modeling language* in a conceptual way, independently of any form of notation or *concrete syntax*, which assigns textual and/or graphical notation(s) to a modeling language.

Models are abstractions of reality for a given purpose. In MDE, they are prescriptive, machine-readable artifacts obtained at the end of a modeling activity. Models are connected (i.e., model elements may be linked beyond the boundary of one model) and dynamic (i.e., models may be analyzed and executed in some form) [4, 9]. *Executable models* are models provided with an operational semantics defining the meaning of the language by implementing an interpreter that directly executes the model behavior.

2.2 Blended Modeling

A formal definition of *blended modeling* has been introduced by Ciccozzi et al. [13]:

"*Blended modeling is the activity of interacting seamlessly with a single model through multiple notations, allowing a certain degree of temporary inconsistencies.*"

From a technical MDE perspective, blended modeling entails the adoption of a single metamodel and a mix of two or more graphical and/or textual concrete syntaxes. From a user perspective, a blended modeling environment should be able to cope with temporary inconsistencies across notations by detecting, solving, and then propagating consistent changes to the underlying model.

In [18], David et al. further characterized the notion of blended modeling by performing a systematic study of blended modeling in commercial and open-source model-driven software engineering tools. The collected tools are classified w.r.t. three main features of blended modeling, namely i) *multiple notations*, ii) *seamless interaction* among such notations, and iii) *flexible consistency management* to (temporarily) favor user experience and understandability over correctness.

In this paper, we contribute to the blended modeling of JSON-based artifacts by promoting a systematic integration of the native JSON textual notation with a graphical one.

2.3 JSONware: JSON and JSON Schema

JSON [35] initially emerged as lightweight and human-readable data serialization and messaging format for supporting information exchange. Two exemplary JSON artifacts are shown in Lst. 1 and Lst. 2. Data is stored in name/value pairs separated by commas, while curly braces hold objects and square brackets hold arrays. As a plain textual artifact, a JSON document can be manually edited in any text editor.

Listing 1: A JSON schema example.

```
{
    "$schema": "http://json-schema.org/draft-07/schema#"
    "type":"object",
    "properties":{
        "name":{"type":"string"},
        "surname":{"type":"string"}
    },
    "additionalProperties":true
}
```

Listing 2: A JSON schema instance, i.e., JSON document, example.

```
{
    "name": "Alessandro",
    "surname": "Colantoni",
    "affiliation": {
        "universityName": "Johannes Kepler University",
        "city":"Linz"
    }
}
```

The JSON Schema[2], promoted and maintained by an active open community [37], is published by IETF as a draft standard and defined as:

"*a JSON-based format for describing the structure of JSON data. JSON schema asserts what a JSON document must look like, ways to extract information from it, and how to interact with it.*" [37].

[2]By convention, we use JSON Schema (uppercase S) to refer to the standard and JSON schema (lowercase S), or simply schema, to refer to a particular schema document.

The JSON Schema defines a so-called *metaschema*, i.e., a self-descriptive schema, validated against itself, which is used to validate any JSON schema. As a *schema* defines the data structure of JSON documents, the document is referred to as an *instance* (e.g., Lst. 2) of a given *schema* (e.g., Lst. 1). With JSON Schema, users are provided with a language to define constraints on JSON documents and tools for checking their conformance between schema instances and the corresponding schema [47–49].

2.4 JSONSchemaDSL: Bridging JSONware and MDE

From a MDE perspective, the metaschema by JSON Schema is a good candidate for playing a *de-facto* metalanguage role in an emerging JSONware technical space [38, 39]. In [40], Maiwald et al. provide a manual classification of JSON schemas collected in a growing and curated collection of JSON schemas called Schemastore [36]. This work testifies how JSON schemas are not only used for data definition and exchange but also for scripting tasks for configuring application services.

In [16], we elicited the JSONware technical space [38] (JSONware TS, see Fig.2) and presented the conceptual architecture, workflow, and technical implementation of *JSONSchemaDSL*, a model-driven approach to bridge the JSONware to the EMF TS [38] while preserving the native JSON concrete syntax.

The JSONSchemaDSL is an approach for developing DSLs based on JSON schema specifications, built atop Eclipse EMF and Xtext. In [16], given a JSON schema as input, the approach can semi-automatically generate the Ecore-based metamodel and an Xtext grammar. The former defines the in-memory representation of a JSON document as a EMF model and is transparent to domain experts. The latter is designed to replicate the native JSON textual concrete syntax and, as such, does not require any MDE-specific knowledge and is suitable for any JSON users. A configuration model specifies variations of the generated grammar (e.g., choosing between JSON arrays or JSON objects).

JSONSchemaDSL reuses the generation capabilities of Xtext. The generated Xtext-based editors, possibly extended with OCL constraints, support editing, parsing, and validation of JSON schemas and JSON instance documents. From a MDE perspective, JSONSchemaDSL enables the (meta)modeling of JSON documents.

As many DevOps approaches are based on JSON, such as Keptn, discussed next, we have a basis for applying MDE for DevOps artifacts.

2.5 Keptn

Keptn is an open-source control plane for orchestrating continuous delivery (CD) and operational processes of cloud-based applications. It started in January 2019 by the company Dynatrace and then donated to the Cloud-Native Computing Foundation (CNCF) in 2020.

Keptn supports a declarative approach to building scalable automation routines for continuous delivery and operations. Keptn can invoke services from external DevOps tools and consumes the generated events while executing continuous delivery. At the time of writing, the integrated tools support testing, observability, deployment activities, and web-hooks to web applications[3].

To configure the provisioning of services, Keptn relies on declarative artifacts or *specifications* [14].

In [16, 17], we applied the JSONSchemaDSL approach to three Keptn specifications, namely Shipyard and SLI/SLO pair, enabling textual modeling and continuous consistency checking capabilities for corresponding JSON-based artifacts. In particular:

- *Shipyard*: The *Shipyard* specification declares a multi-stage delivery workflow by defining what needs to be done. A delivery workflow is based on multiple stages, each with different task sequences. A task sequence is a set of actions for a specific delivery or operational process. Following this declarative approach, there is no need to write imperative pipeline code.
- *SLI* and *SLO*: A *service-level indicator* (SLI) is a "carefully defined quantitative measure of some aspect of the level of service that is provided" [3]. A *service-level objective* (SLO) is "a target value or range of values for a service level that is measured by an SLI [3]. Together, the SLI/SLO specifications declare a quality gate for a given service. This quality gate can be leveraged in the delivery or operational process to measure the defined quality criteria.

At the time of writing, Keptn provides two other JSON-based *specifications*, namely *CloudEvents* and *Remediation*, to define a contract for tool integration and a list of remediation actions for a given service, respectively [14].

3 APPROACH

Fig. 1 depicts the use cases covered by the proposed approach, extending the use cases (in gray) played by language engineers, tool providers, and domain experts presented in [16]. In particular:

- The *Language Engineer* creates and manipulates languages. With JSONSchemaDSL, she can define the abstract syntax of JSON-based DSLs. By construction, the native JSON notation is chosen as textual concrete syntax. The language engineer is expected to provide at least two concrete syntaxes for blended modeling, and it is a common choice [18] to mix textual and graphical ones. An operational semantics specification can also be specified, obtaining a JSON-based xDSL
- The *Domain Expert* creates and manipulates JSON documents. In JSONware, these artifacts are schema instances conforming to JSON schemas. Due to JSON, a domain expert is expected to work with textual artifacts. However, if additional concrete syntaxes and supporting tools are available (e.g., a graphical one), she can benefit from a blended modeling experience. The JSON documents can also be executed if an operational semantics and interpreter are provided.
- The *Tool Provider* implements tools to support activities. Essential tools are textual and graphical editors for blended modeling and interpreter for execution via operational semantics

[3]The up-to-date list is available at https://keptn.sh/docs/integrations/

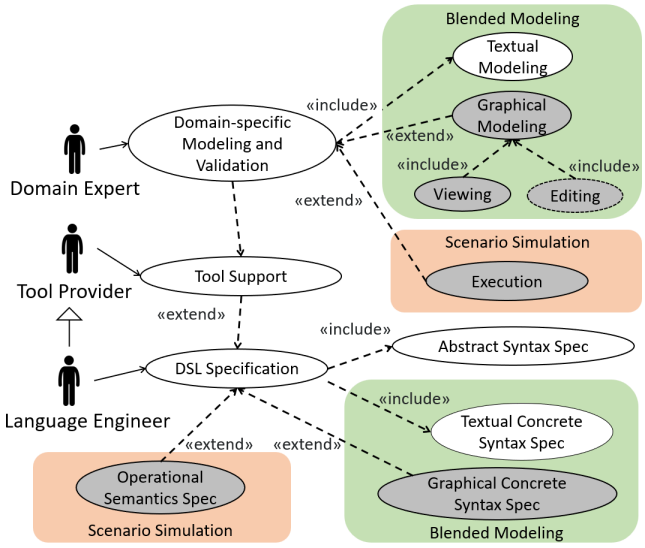

Figure 1: Actors and use cases of our field of investigation.

In this paper, we build atop the outcome of [16] and enrich the use cases in Fig. 1 with two new capabilities, i.e., blended modeling and scenario simulation of CD pipelines specified in Shipyard by Keptn [32]. A bird's eye view of the technical realization of these two new capabilities is given in Fig. 2[4].

A JSON schema is taken as input by JSONSchemaDSL (1), which generates (2), via a semi-automated process [16], the corresponding Ecore metamodel and Xtext grammar. Thanks to the native capabilities of EMF and Xtext, tree-based and textual editors for JSON instance documents are automatically generated, enriched by JSON schema-specific OCL constraints. The textual editor adopts the native JSON notation and is meant to be used by domain experts already acquainted with the JSON notation for their particular engineering activity. In addition, MDE experts can also further inspect or edit the actual structure of the in-memory representation of the JSON artifact in EMF using the generated tree-based editor.

A graphical concrete syntax for the given JSON schema is realized via Sirius [26], an Eclipse project which allows the generation of a graphical modeling workbench for EMF-based models. Language engineers and domain experts are expected to agree on the graphical notation. A graphical editor is then generated for JSON instance documents. The Sirius-based editor offers the possibility to visualize and edit the content of a JSON instance document.

It is worth noting that all the editors mentioned above, i.e., tree-based, textual, and graphical, share the same in-memory representation based on EMF and can be opened in the same Eclipse IDE. This capability is necessary to enable a blended modeling experience of JSON artifacts (4). Indeed, domain experts can seamlessly choose among three different editors to manipulate the same JSON Instance document, which, in turn, conforms both to the original JSON schema and the generated Ecore metamodel and Xtext grammar. Moreover, by preserving the native JSON notation, the domain experts can continue using any existing JSON document editor.

Finally, if a JSON schema is provided with executable operational semantics, any compliant JSON instance documents can be executed (5). We implemented an executable operational semantics for Shipyard using GEMOC Studio. It provides generic components through Eclipse technologies developing, integrating, and using of heterogeneous xDSLs via a so-called *language workbench*. In particular, the GEMOC Studio integrates Kermeta 3 (K3) [21], an action language used to implement the execution semantics of Ecore metamodels, as the ones generated by the JSONSchemaDSL. Kermeta 3 is built on top of the Xtend, a dialect of Java, which compiles into readable Java-compatible source code. As a result, a Java-compliant interpreter can be generated for a given JSON schema. The availability of executable semantics and interpreters for JSON-based xDSLs may help (i) domain experts in performing activities and (ii) tool providers to augment their tools with MDE technologies whenever JSON schemas are used in model-based DevOps processes [30].

It is worth noting that the JSONSchemaDSL is not a "one size fits all" approach. It covers domain-independent steps, like representing the JSON metaschema in Ecore, hidden on purpose from Fig. 2, and domain-specific ones. The latter must be replicated for each JSON schema.

The extended capabilities of the JSONSchemaDSL approach (see Fig.1) benefit both domain experts and tool providers. The former can choose among a richer set of tools to manipulate JSON artifacts, in addition to and compatible with any existing JSON-based tool. The latter can leverage the generation capabilities provided by model-driven language workbenches, like GEMOC Studio, to give a richer, domain-specific tool set for JSON artifacts.

4 BLENDED MODELING AND SIMULATION OF CONTINUOUS DELIVERY PIPELINES

This section describes the blended modeling and scenario simulation for CD pipelines specified in Shipyard. The case study extends the one presented in [16] to show an application of the approach in Fig. 2, combining JSONSchemaDSL and GEMOC Studio.

Thanks to JSONSchemaDSL, each Keptn specification can be engineered as a fully-fledged DSL. Subsequently, the corresponding JSON document can be edited and visualized by the generated textual and graphical editors.

In [16, 17], we applied the JSONSchemaDSL to Shipyard and SLI/SLO specifications obtaining the following artifacts and tools:

- Ecore metamodels and Xtext grammars for Shipyard and SLI/SLO specifications
- Xtext-based textual editors, with additional OCL constraints, for Shipyard and SLI/SLO
- A continuous consistency checking mechanism between explicitly linked Shipyard and SLI/SLO models, based on Viatra, an open-source model query, validation, and transformation framework supporting the efficient evaluation of model queries on EMF models [50]

In this paper, we further extend the case study for Shipyard, obtaining the following additional artifacts and tools:

- Sirius-based Shipyard graphical editor

[4]We invite the reader to refer to [16] for an in-depth explanation of the language engineering capabilities supported by JSONSchemaDSL.

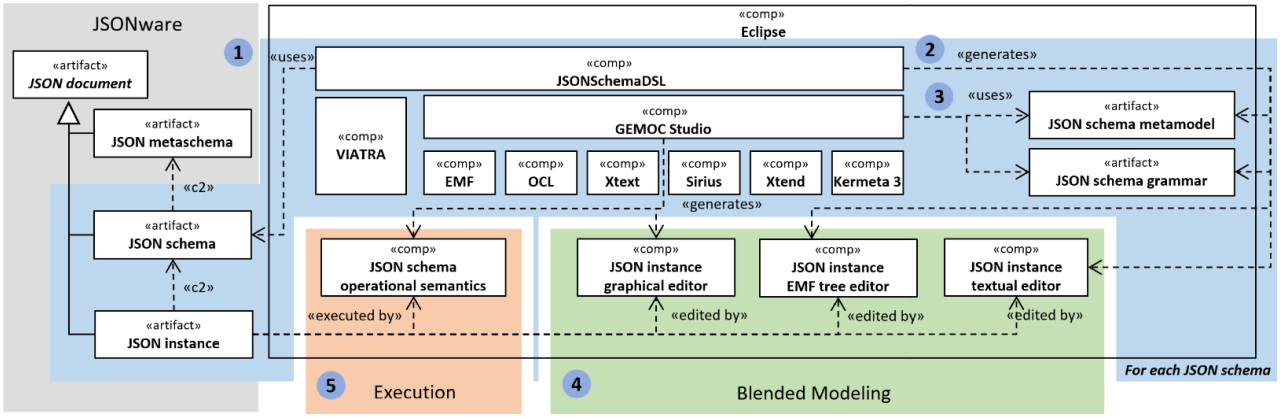

Figure 2: JSONSchemaDSL and GEMOC Studio for blended modeling and execution of JSON-based (x)DSLs.

- Shipyard executable operational semantics and interpreter based on Xtend and Kermeta 3 [21]

As a result, Keptn users as domain experts can benefit from a blended modeling experience and can simulate CD scenarios. We further detail these two new capabilities in the following.

4.1 Blended Modeling for Shipyard

Fig. 3 shows a running GEMOC Modeling Workbench, i.e., a xDSL-specific Eclipse instance used by domain designers, in our case Shipyard experts, to create, and execute Shipyard models. In Fig. 3, the view is suitably arranged to show the textual and graphical editors of a Shipyard model.

A *shipyard-sockshop* Shipyard model is available from the official Keptn documentation[5]. The *shipyard-sockshop* consists of three consecutive stages, i.e., development, hardening, and production. Development and hardening stages contain a delivery sequence of four tasks: deployment, test, evaluation, and release. The next production stage contains a delivery sequence including deployment and release tasks, a rollback sequence containing a single homonym task, and a remediation sequence, including remediation and evaluation tasks. If the delivery sequence of the dev stage (dev.delivery) is activated by the user, and its execution is successful, the delivery sequence of the hardening stage (hardening.delivery) is triggered. During the hardening stage, if the delivery sequence is successful, then the delivery sequence is also triggered in the production stage. If the latter fails, the rollback sequence is started. In this *shipyard-sockshop* example, a remediation sequence, including remediation and evaluation tasks, is available at the production stage. The above-described workflow does not trigger the remediation sequence and, as such, appears disconnected from it. Therefore, if started by the user, it is executed independently from other stages, sequences, or tasks.

Fig. 3a shows the *shipyard-sockshop* CD pipeline as an instance of the Shipyard DSL in the generated Xtext-based editor, including highlighting of Shipyard keywords. Fig. 3b shows the same *shipyard-sockshop* CD pipeline in a Sirius-based graphical editor.

Since the Shipyard JSON schema defines a continuous delivery pipeline as a hierarchical workflow of stages, sequences, and tasks, the graphical concrete syntax has been designed to resemble a workflow. It is worth noting that, for the sake of blended modeling (single model, multiple notations, see Section 2), we did not consider as viable solutions the adoption of existing software process languages like SPEM [44], BPMN [45], or UML activities [46], that require the integration (e.g., via model transformations [9]) of additional metamodels.

Designing the Shipyard continuous delivery workflow directly in JSON (Fig. 3a) is the typical choice for Keptn users since Keptn is provided as a MDE-agnostic tool and does not provide Shipyard-specific graphical editors. Specifying complex Shipyard CD pipelines in JSON can be verbose and then cumbersome. The current implementation of the Shipyard graphical editor works as a viewer for Shipyard CD pipelines. The depicted workflow is automatically synchronized when the Shipyard model is saved. We are working on offering a full blended modeling experience to Keptn users, which requires extending the graphical editor with editing capabilities, following the Sirius implementation guidelines[6].

Table 1 provides a preliminary overview of the blended modeling capabilities for Shipyard, according to the classification framework proposed by David et al. in [18].

JSONSchemaDSL and the generated tool support for Shipyard are open-source tools [5, 6]. The tool support for Shipyard xDSL is available as standalone Eclipse desktop applications. A web-based version is left as future work. Candidate technologies to be considered in Fig. 2 are Eclipse Theia [24], EMF Cloud [25], and GEMOC headless [31]. Collaboration can be supported in Keptn via GitOps [51], as discussed in our previous work [17].

Concerning user-oriented features, two notations, i.e., the JSON native textual notation (Fig. 3a) and graphical one (Fig. 3b), are provided to domain experts for Shipyard. These two notations are used in separated editors, and their overlap is complete: all the information in the JSON document can be depicted on the diagram or, if not, shown in property views associated with the model element selected on the diagram. Tree-based notations (and editors) are not

[5]https://github.com/keptn/spec/blob/master/shipyard.md. The example is provided in YAML. Any YAML document can be automatically converted into JSON format.

[6]See https://wiki.eclipse.org/Sirius/Tutorials/StarterTutorial for a tutorial example.

Table 1: Preliminary classification of the blended modeling capabilities of the Shipyard model editor based on [18].

Generic	**Tool**	Open-source	YES
		Web-based	NO
		Motivation	Blended modeling and simulation of contiuous delivery pipelines specified in Shipyard, a JSON schema-based DSL used in the Keptn open source tool.
		Collaboration	YES
User-oriented	**Notation**	Notation types	TEXTUAL, GRAPHICAL
		Notation instances (number of)	2
		Embedded languages	NO
		Overlap	PARTIAL
	Visualization and Navigation	Visualize multiple notations	YES
		Synchronous navigation	NO
		Navigation among notations	NO
Realization-oriented	**Mapping and Platform**	Mapping	PARSER-BASE
		Platform	ECLIPSE
	Change Propagation and Traceability	Change propagation	SEQUENTIAL
		Traceability	NO
	Inconsistency Management	Inconsistency visualization	NO
		Inconsistency management type	NO
		Inconsistency management automation	NO

considered by [18]. As shown in Fig. 3, it is possible to visualize both the textual and graphical versions of the model. However, the generated editors do not yet provide advanced synchronization or navigation routines (e.g., simultaneous selection of the same model elements across notations or customized contextual menus).

Realization-oriented features capture technicalities. JSONSchema-DSL is based on Xtext [28] that generates a parser for a given JSON schema. Change propagation across notations happens when the textual editor saves a valid Shipyard model.

Mechanisms for traceability and inconsistency management across notations are not yet explicitly supported and are left for future work.

4.2 Simulation of Continuous Delivery Scenarios

A Keptn user can decide the starting sequence of a Shipyard CD pipeline, uniquely identified by its stage and sequence name (e.g., dev.delivery). At the time of writing, Keptn users can only execute and test Shipyard CD pipelines by uploading and running them on a Keptn runtime via CLI[7]. Keptn does not provide any means for simulating the CD pipeline outside the Keptn runtime.

To overcome this limitation, we are extending Shipyard with an operational semantics implemented in Xtend/Kermeta 3 (Fig.2, label 5), following the GEMOC Studio guidelines[8], obtaining a Shipyard xDSL. The goal is to provide an approach for early validation by simulation of CD scenarios defined in Shipyard, before their actual executions by the Keptn engine. For this purpose, we defined a minimal Scenario metamodel in Ecore (see Fig.4a). A top-level *ShipyardExecutionSuite* refers to the *ShipyardRoot*, i.e., the top-level metaclass of the Shipyard schema metamodel as generated by JSONSchemaDSL. A ShipyardExecutionSuite is composed of many *Scenario*s. Each scenario consists of two possibly empty collections of *SequenceEvent*s and *TaskEvent*s, triggering the Sequences and Tasks, respectively (Fig.4b). A simulation scenario starts from an arbitrary initial Sequence. Each sequence or task can be correctly executed, fail, or cause a warning, generating a corresponding passed, failed, or warning event. Fig.3 shows a running debug session of the *shipyard-sockshop* driven by the scenario named Rollback of the associated ShipyardExecutionSuite model depicted in Fig.3c. In our example, the proposed ShipyardExecutionSuite comprises four scenarios with the required initial sequence (in parenthesis), namely: rollback (dev.delivery) which execution is shown in Fig.3b, hardening evaluation warning (dev.delivery) and remediation (production.remediation). The rollback scenario assumes the failure of the production.delivery sequence, while the hardening evaluation warning scenario includes a warning for the hardening.delivery.evaluation task. Finally, by default, we assign a passed event to any sequences and tasks not explicitly referred to in a ShipyardExecutionSuite model. For this reason, we assume that all sequences and tasks in the remediation scenario are passed.

Fig3b shows the outcome of a completed debugging session driven by the events defined in the rollback scenario. Accordingly, development and hardening stages are successfully executed. During the production stage, the delivery sequence failed to trigger a successful rollback sequence. The remediation sequence in the production stage is not executed since the execution started from the delivery sequence in development stage, and no triggers are generated during the execution of the Shipyard JSON document in Fig.3a.

The definition of simulation scenarios for Shipyard CD pipelines is still under development. For example, we are investigating using the IEEE XES standard [34] to model more complex event-based scenarios.

[7] https://keptn.sh/docs/0.13.x/reference/cli/commands/
[8] https://download.eclipse.org/gemoc/docs/releases/3.4.0/userguide-lw-make-language-executable.html

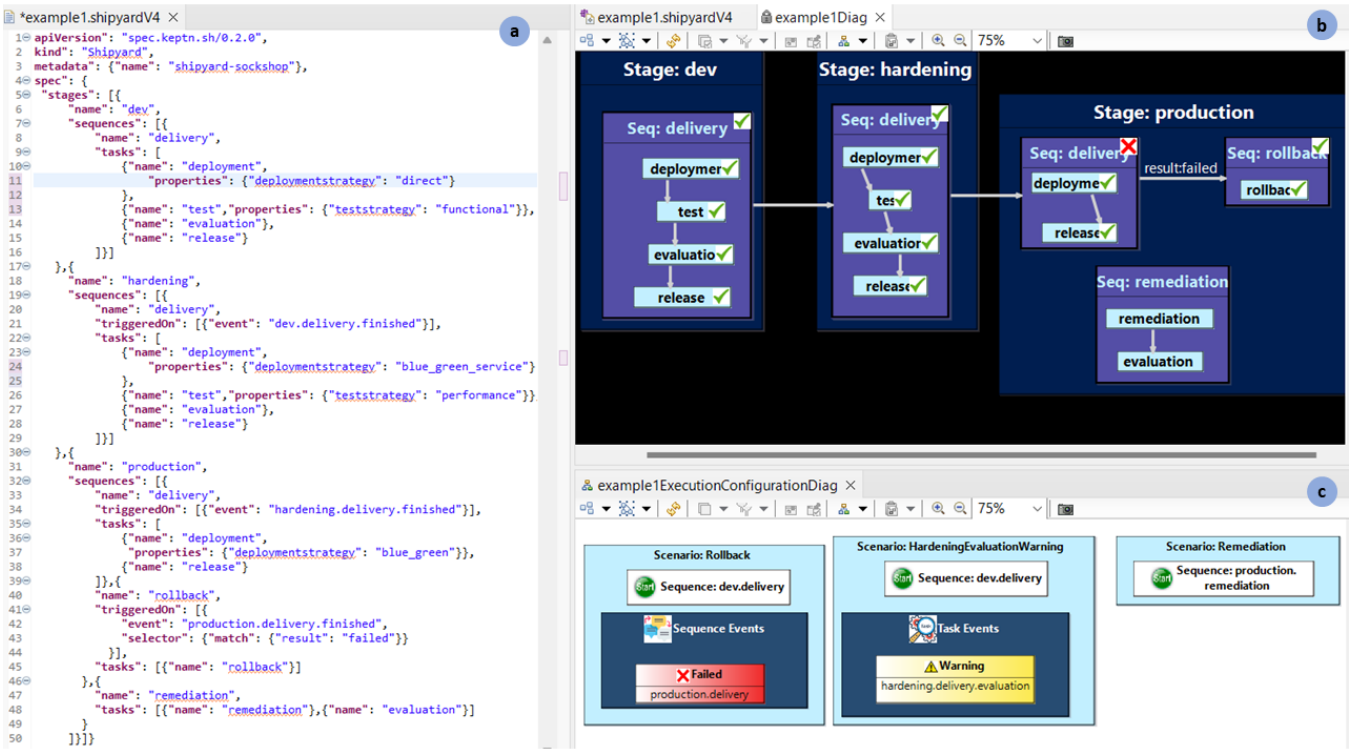

Figure 3: A Shipyard model can be edited as a JSON document (a), depicted on a workflow diagram (b), and simulated according to a given event-driven scenario (c).

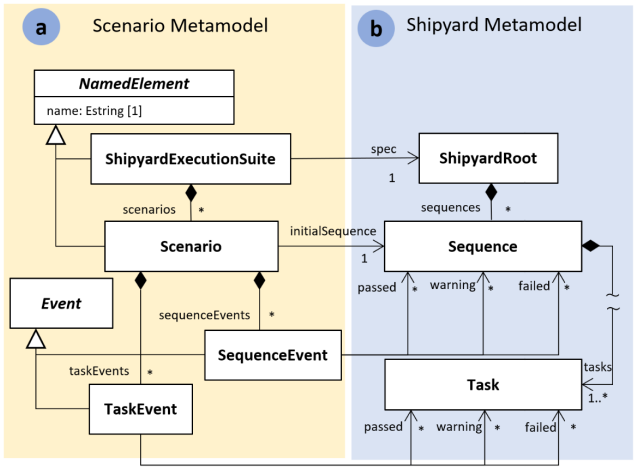

Figure 4: Scenario (a) and excerpt of the generated Shipyard (b) metamodels.

5 RELATED WORK

5.1 On Blended Modeling of JSON Documents

For related work concerning JSONware and MDE of this subject, the reader should refer to [16].

Blended modeling is a recent research topic. The most recent work on blended modeling support in commercial and open source tools is provided by David et al. in [18]. In their work, 5000 academic papers and nearly 1500 entries of gray literature have been reviewed. They identified 26 tools representing the current spectrum of modeling tools. To the best of the authors' knowledge, none of the selected tools address model-driven approaches and blended modeling concerns related to JSON documents.

5.2 On Simulation of DevOps Processes

In [41], Medvedev et al. propose the performance analysis of a multi-agent CI/CD pipeline, which is mapped to a queuing multi-channel system. The simulation is focused on improving the schedule of CI/CD tasks among server agents to improve the overall performance of the CI/CD pipeline.

In [2], Mesmia et al. model a DevOps workflow using the Business Process Modeling Notation (BPMN) and manually translate it into a non-Markovian Stochastic Petri Net (SPN) executable model for the sake of verification and duration prediction.

In [1], Alonso et al. present the overall architecture and capabilities of the DECIDE DevOps framework for multi-cloud applications. DECIDE includes a set of modules supporting different CI/CD activities. One such module, OPTIMUS [19], runs pre-deployment simulations to determine the best deployment scenarios, based on matching the application's non-functional requirements and profiles of available cloud services.

Unlike [1, 2, 41], our solution follows a systematic model-driven approach, implementing the operational semantics of an existing non-executable JSON-based DSL, i.e., Shipyard. Moreover, our approach offers automation capabilities thanks to MDE techniques and practices [9] by generating supporting tools like editors and interpreters. This aspect is neglected in [1, 2, 41]. For example, [2] proposes a translational semantics to stochastic Petri Nets (SPN), but the transformation to SPN is performed manually.

In [1], the authors mentioned MELODIC [33, 42] as the only related work to support deployment simulation and optimization of big data applications adopting a model-driven engineering approach. MELODIC is a comprehensive approach for modeling, deploying, and optimizing of multi-cloud applications. In MELODIC, the multi-cloud native application is modeled in the Cloud Application Modelling Execution Language (CAMEL) and Business Process Models (BPMs). The CAMEL application model is transformed into a Constraint Programming Model for mathematical optimization of the deployment. Our approach builds upon the same MDE principles and practices. Our goal is nevertheless different and focused on JSON-based artifacts already used by existing MDE-agnostic DevOps tools, like Keptn.

Concerning simulation, the purposes of collected related work are formal verification [2], performance [41], and overall optimization [1, 33] of the deployment process. The Shipyard operational semantics presented in this work is realized for the functional tests of CD simulation scenarios. Nevertheless, quality aspects of simulation scenarios can be considered in Keptn by combining SLI/SLO and Shipyard models as shown in [17].

6 CONCLUSION AND FUTURE WORK

This paper presents a work in progress to enable the blended modeling and simulation of DevOps processes specified using a JSON-based DSL. For this purpose, we leveraged and combined in a pipeline the model-driven capabilities of our JSONSchemaDSL approach and the GEMOC Studio, enabling the semi-automated generation of fully fledged executable DSLs and tool support from JSON schema documents. At the time of writing, the tool support comprises an Xtext-based textual editor, a Sirius-based graphical viewer, and a GEMOC-based interpreter. We presented an extended case study based on Keptn, an open-source tool for DevOps automation of cloud-native applications. Keptn provides several JSON schemas or *specifications* used by Keptn experts to validate DevOps artifacts to suitably configure the continuous delivery of cloud-native applications.

In this paper, we are paving the way towards the blended modeling and event-based simulation of continuous delivery pipelines edited in Shipyard, a JSON schema provided by Keptn. Nevertheless, to accomplish such a goal, additional implementation effort is required. First, a fully operational blended modeling capability needs the implementation of editing actions into the Sirius-based viewer, enabling bidirectional propagation of model changes via the underlying EMF-based APIs. Second, to cope with temporary inconsistencies, we aim to investigate adopting the GitOps framework [51] as already used in Keptn and discussed in [17], leveraging the textual JSON model as source of truth.

Finally, as future work, we also aim to provide our JSONSchema-DSL approach by challenging its capabilities and potential integration with existing model-driven technologies with extensions of the Keptn case study, e.g., by developing an ecosystem of (x)DSLs and linked models based on Keptn specifications.

ACKNOWLEDGMENT

This work was partially funded by the following projects: The EU Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 813884 (www.lowcomote.eu); the AIDOaRt project funded by the ECSEL Joint Undertaking (JU) under grant agreement No 101007350 (www.aidoart.eu). The Austrian Science Fund (P 30525-N31) (https://se.jku.at/lea-language-engineering-for-analyzable-executable-dsmls/). The Austrian Research Promotion Agency (FFG), program ICT of the Future, project number 867535 (https://hybridlux.wu.ac.at/). The research also contributed to the ITEA3 BUMBLE project (18006) (https://itea4.org/project/bumble.html) .

REFERENCES

[1] Juncal Alonso, Kyriakos Stefanidis, Leire Orue-Echevarria, Lorenzo Blasi, Michael Walker, Marisa Escalante, María José López, and Simon Dutkowski. 2019. DECIDE: An Extended DevOps Framework for Multi-cloud Applications. In *Proceedings of the 2019 3rd International Conference on Cloud and Big Data Computing, ICCBDC 2019, Oxford, UK, August 28-30, 2019*. ACM, 43–48. https://doi.org/10.1145/3358505.3358522

[2] Walid Ben Mesmia, Mohamed Escheikh, and Kamel Barkaoui. 2021. DevOps Workflow Verification and Duration Prediction Using Non-Markovian Stochastic Petri Nets. *J. Softw. Evol. Process* 33, 3 (mar 2021), 25 pages. https://doi.org/10.1002/smr.2329

[3] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. 2016. *Site reliability engineering: How Google runs production systems.* " O'Reilly Media, Inc.".

[4] Jean Bézivin. 2005. On the unification power of models. *Software & Systems Modeling* 4, 2 (2005), 171–188.

[5] BISE Institute, JKU, Linz. 2022. JsonSchemaDSL. https://zenodo.org/record/5149206#.YuTp-XZBxD9, last accessed on 31/07/22.

[6] BISE Institute, JKU, Linz. 2022. Shipyard Editors. https://github.com/lowcomote/shipyard-operational-semantics, last accessed on 31/07/22.

[7] Modeling Languages Blog. 2019. DevOps for models and modeling DevOps. https://modeling-languages.com/devops-modeling-workshop/, last accessed on 31/07/22.

[8] Francis Bordeleau, Jordi Cabot, Juergen Dingel, Bassem S. Rabil, and Patrick Renaud. 2020. Towards Modeling Framework for DevOps: Requirements Derived from Industry Use Case. In *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, Jean-Michel Bruel, Manuel Mazzara, and Bertrand Meyer (Eds.). Springer International Publishing, Cham, 139–151.

[9] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2017. *Model-Driven Software Engineering in Practice, Second Edition.* Morgan & Claypool Publishers. https://doi.org/10.2200/S00751ED2V01Y201701SWE004

[10] Loli Burgueño, Federico Ciccozzi, Michalis Famelis, Gerti Kappel, Leen Lambers, Sébastien Mosser, Richard F. Paige, Alfonso Pierantonio, Arend Rensink, Rick Salay, Gabriele Taentzer, Antonio Vallecillo, and Manuel Wimmer. 2019. Contents for a Model-Based Software Engineering Body of Knowledge. *Software and Systems Modeling* 18, 6 (2019), 3193–3205. https://doi.org/10.1007/s10270-019-00746-9

[11] Necco Ceresani. 2016. The Periodic Table of DevOps Tools v.2 is Here. https://blog.xebialabs.com/2016/06/14/periodic-table-devops-tools-v-2/, last accessed on 31/07/22.

[12] Federico Ciccozzi, Matthias Tichy, Hans Vangheluwe, and Danny Weyns. 2019. Blended Modelling - What, Why and How. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. 425–430. https://doi.org/10.1109/MODELS-C.2019.00068

[13] Federico Ciccozzi, Matthias Tichy, Hans Vangheluwe, and Danny Weyns. 2019. Blended Modelling - What, Why and How. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. ACM/IEEE, 425–430. https://doi.org/10.1109/MODELS-C.2019.00068

[14] Cloud Native Computing Foundation. 2022. Keptn Specifications. https://github.com/keptn/spec, last accessed on 31/07/22.

[15] Alessandro Colantoni, Luca Berardinelli, and Manuel Wimmer. 2020. DevOpsML: towards modeling DevOps processes and platforms. In *MODELS '20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems, Virtual Event, Canada, 18-23 October, 2020, Companion Proceedings*, Esther Guerra and Ludovico Iovino (Eds.). ACM, 69:1–69:10. https://doi.org/10.1145/3417990.3420203

[16] Alessandro Colantoni, Antonio Garmendia, Luca Berardinelli, Manuel Wimmer, and Johannes Bräuer. 2021. Leveraging Model-Driven Technologies for JSON Artefacts: The Shipyard Case Study. In *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 250–260. https://doi.org/10.1109/MODELS50736.2021.00033

[17] Alessandro Colantoni, Benedek Horváth, Ákos Horváth, Luca Berardinelli, and Manuel Wimmer. 2021. Towards Continuous Consistency Checking of DevOps Artefacts. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. 449–453. https://doi.org/10.1109/MODELS-C53483.2021.00069

[18] Istvan David, Malvina Latifaj, Jakob Pietron, Weixing Zhang, Federico Ciccozzi, Ivano Malavolta, Alexander Raschke, Jan-Philipp Steghöfer, and Regina Hebig. 2022. Blended modeling in commercial and open-source model-driven software engineering tools: A systematic study. *Software and Systems Modeling* (June 2022). https://doi.org/10.1007/s10270-022-01010-3

[19] DECIDE Project. 2020. Optimus Tool. https://www.decide-h2020.eu/content/optimus, last accessed on 2022-07-31.

[20] Digital.ai. 2022. Periodic Table of DevOps Tools. https://digital.ai/devops-tools-periodic-table, last accessed on 31/07/22.

[21] Diverse Project. 2022. Kermeta 3. http://diverse-project.github.io/k3/, last accessed on 31/07/22.

[22] Dynatrace Gmbh. 2021. Modern continuous delivery with Keptn - Talk at Lowcomote Heterogeneous Low-Code Engineering in Industry. https://youtu.be/Zlt0HolMK08?t=7182, last accessed on 31/07/22.

[23] Eclipse Foundation. 2022. Eclipse Modeling Framework. www.eclipse.org/modeling/emf/, last accessed on 31/07/22.

[24] Eclipse Foundation. 2022. Eclipse Theia. https://theia-ide.org/ last accessed on 31/07/22.

[25] Eclipse Foundation. 2022. EMF Cloud. https://www.eclipse.org/emfcloud/ last accessed on 31/07/22.

[26] Eclipse Foundation. 2022. Sirius. https://www.eclipse.org/sirius/, last accessed on 31/07/22.

[27] Eclipse Foundation. 2022. Xtend. https://www.eclipse.org/xtend/, last accessed on 31/07/22.

[28] Eclipse Foundation. 2022. Xtext. https://www.eclipse.org/Xtext/, last accessed on 31/07/22.

[29] Romina Eramo, Vittoriano Muttillo, Luca Berardinelli, Hugo Bruneliere, Abel Gomez, Alessandra Bagnato, Andrey Sadovykh, and Antonio Cicchetti. 2021. AIDOaRt: AI-augmented Automation for DevOps, a Model-based Framework for Continuous Development in Cyber-Physical Systems. In *2021 24th Euromicro Conference on Digital System Design (DSD)*. 303–310. https://doi.org/10.1109/DSD53832.2021.00053

[30] Jokin Garcia and Jordi Cabot. 2019. Stepwise Adoption of Continuous Delivery in Model-Driven Engineering. In *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, Jean-Michel Bruel, Manuel Mazzara, and Bertrand Meyer (Eds.). Springer International Publishing, Cham, 19–32.

[31] GEMOC. 2022. The GEMOC Initiative On the Globalization of Modeling Languages. http://gemoc.org/, last accessed on 31/07/22.

[32] Dynatrace Gmbh. 2022. Keptn Repository. https://keptn.sh/, last accessed on 31/07/22.

[33] Geir Horn and Pawel Skrzypek. 2018. MELODIC: Utility Based Cross Cloud Deployment Optimisation. In *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. 360–367. https://doi.org/10.1109/WAINA.2018.00112

[34] IEEE. 2016. IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams. *IEEE Std 1849-2016* (2016), 1–50. https://doi.org/10.1109/IEEESTD.2016.7740858

[35] JSON. 2021. JSON Web Page. http://json.org/, last accessed on 31/07/22.

[36] JSON Schema . 2022. Schema Store. https://www.schemastore.org/json/, last accessed on last accessed on 2022-07-31.

[37] JSON Schema. 2021. JSON Schema Web Page. http://json-schema.org/, last accessed on 31/07/22.

[38] Ivan Kurtev, Jean Bézivin, and Mehmet Aksit. 2002. Technological spaces: An Initial Appraisal. *CoopIS, DOA* (2002).

[39] Ralf Lämmel. 2018. *Software languages: Syntax, semantics, and metaprogramming*. Springer.

[40] Benjamin Maiwald, Benjamin Riedle, and Stefanie Scherzinger. 2019. What Are Real JSON Schemas Like?. In *Advances in Conceptual Modeling*, Giancarlo Guizzardi, Frederik Gailly, and Rita Suzana Pitangueira Maciel (Eds.). Springer International Publishing, Cham, 95–105.

[41] Denis Medvedev and Konstantin Aksyonov. 2021. The Development of a Simulation Model for Assessing the CI/CD Pipeline Quality in the Development of Information Systems Based on a Multi-Agent Approach. *MATEC Web of Conferences* 346 (01 2021), 03095. https://doi.org/10.1051/matecconf/202134603095

[42] MELODIC. 2022. Melodic Homepage. http://www.melodic.cloud/, last accessed on last accessed on 2022-07-31.

[43] JKU Institute of Business Informatics. 2020. DevOpsML. https://github.com/lowcomote/devopsml/tree/1.2.2, last accessed on 31/07/22.

[44] OMG. 2008. Software & Systems Process Engineering Metamodel. https://www.omg.org/spec/SPEM/About-SPEM/, last accessed on 31/07/22.

[45] OMG. 2011. Business Process Model And Notation. http://www.bpmn.org/, last accessed on 31/07/22.

[46] OMG. 2017. Unified Modeling Language. https://www.omg.org/spec/UML, last accessed on 31/07/22.

[47] Felipe Pezoa, Juan L. Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. 2016. Foundations of JSON Schema. In *WWW '16*. 263–273.

[48] JSON Schema. 2022. JSON Schema Implementations. https://json-schema.org/implementations.html, last accessed on 31/07/22.

[49] JSON Schema. 2022. JSON Schema Test Suite. https://github.com/json-schema-org/JSON-Schema-Test-Suite, last accessed on 31/07/22.

[50] Dániel Varró, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, István Ráth, and Zoltán Ujhelyi. 2016. Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework. *Softw. Syst. Model.* 15, 3 (2016), 609–629. https://doi.org/10.1007/s10270-016-0530-4

[51] Weaveworks. 2022. GitOps. https://www.weave.works/technologies/gitops/, last accessed on 31/07/22.