Towards a Cloud-native Tool for Model-based Systems Engineering of Spacecraft

Dennis Eller[1 [0009−0009−9006−6460]], Philipp Chrszon[1 [0000−0002−8785−0272]], Philipp M. Fischer[1 [0000−0003−2918−5195]], and Andreas Gerndt[1,2 [0000−0002−0409−8573]]

[1] German Aerospace Center (DLR)
Institute for Software Technology, Brunswick, Germany
{dennis.eller, philipp.chrszon, philipp.fischer, andreas.gerndt}@dlr.de
[2] University of Bremen, Bremen, Germany

Abstract. The landscape of modeling tools for Model-based Systems Engineering is largely dominated by desktop applications. While these tools are highly advanced, their deployment is often challenging, especially if used by large teams. For each machine they are used on, installation and updating must be managed. Also, these applications often require considerable hardware resources. Transforming MBSE tools to use web and cloud technologies can alleviate these issues. We propose a cloud-native architecture as an example for transforming Eclipse-based tools. It is inspired by a set of requirements that have been identified by consulting several experienced domain experts. A prototype implementation shows that almost all of the must-be requirements can be fulfilled or partially fulfilled by utilizing existing cloud-native technologies. Finally, we conducted performance measurements, which showed that the overhead of cloud technologies decrease with increasingly complex computations.

Keywords: Model-based Systems Engineering · Modeling Tool · Cloud-native · Eclipse Rich Client Platform

Addressing Advancements, Limitations, and Customization of MBSE Approaches in Space

1 Introduction

Model-based Systems Engineering (MBSE) fundamentally requires tool support for creating, modifying, and sharing of models. Traditionally, MBSE tools have been implemented as classical desktop applications, e.g., on the basis of the Eclipse Rich Client Platform. One example for such a tool is Virtual Satellite [3], which is developed by the German Aerospace Center (DLR) and is mainly used for concurrent engineering studies in early lifecycle phases. In recent years, there has been a trend to transform applications to individually containerized microservices using cloud-native technologies, as this promises several advantages like high scalability and availability as well as preventing vendor lock-in [2]. Furthermore, they enable a web-based access using a browser on almost any device and thus simplify the deployment as well as the maintenance of the application.

In this paper, we propose and evaluate an architecture to transform an application like Virtual Satellite to cloud-native technologies. First, we collected requirements using surveys. Next, we evaluated which requirements can be fulfilled in a prototype implementation using mostly existing cloud-native technologies. Finally, the prototype has been utilized for conducting a performance evaluation to determine the overhead of the proposed architecture.

2 Related Work

Tools and Frameworks. Since our architecture mainly targets usage in research and requires high customizability, we focus on open-source tools for the implementation. MBSE tools, such as VSD, OCDT, COMET, Nanospace [4], Capella [1] or Virtual Satellite [3] provide a web API or web-based user interface, but we do not consider them as cloud-native. To create cloud-native IDEs, there exist tools and frameworks, like Eclipse Theia and Eclipse Che. However, they are primarily intended for software development and thus are not tailored for MBSE. Nevertheless, they may serve as a basis for implementing modeling tools, as exemplified by RIDE [5], an IDE for the Reflex language for modeling cyber-physical systems, or tCollab [9], which is used for defining requirements and generating diagrams. For creating web and cloud based modeling tools, EMF.Cloud can be used.

Requirements Engineering. For gathering requirements, user stories [6] are a popular choice, as they combine a formal specification with natural language descriptions. Requirements can be divided into functional and quality requirements by the ISO 25010 standard [7] and may be further classified into priority groups using the Kano model [8], for which the table from Pouliot [10] may be applied. According to the Kano model, requirements can be classified into the following categories: *Must-be* requirements are taken for granted and result in dissatisfaction when not fulfilled. *One-dimensional* requirements result in satisfaction if fulfilled and dissatisfaction when not fulfilled. *Attractive* requirements are not expected and result in satisfaction when fulfilled. *Indifferent* requirements do not have an impact on satisfaction. It is recommended to combine multiple sources for requirements engineering [6]. *Must-be* requirements can be derived from existing systems and their documentation, *one-dimensional* requirements may be sourced via surveys and *attractive* requirements can be derived via creative techniques.

3 Methodology and Gathered Requirements

Methodology. To collect the requirements for the architecture, user stories are used. Specifically, the following steps were taken:

1. Gather user stories via a survey with domain experts and own research.
2. Categorize user stories according to ISO 25010.
3. Conduct Kano questionnaire for priority assessment according to Pouliot.
4. Derive architecture based on the user stories.
5. Evaluate the proposed architecture if it is fulfilling the user stories.
6. Evaluate the performance of the architecture in comparison to local execution.

Gathered Requirements. In total, 7 experienced domain experts participated in the survey and 52 requirements were identified. Functional requirements were split into the groups *Modeling*, *Concurrent Engineering*, *Interfaces & Export of Data*, *Product Lines, Versioning & Synchronization*, and *Authentication & Authorization*. For non-functional requirements, ISO 25010 is referred. 19 *Attractive*, 20 *Must-be*, 3 *One-dimensional*, and 10 *Indifferent* requirements have been identified. Table 1 shows a selection of requirements which are relevant for the architecture. The assessment if requirements could be fulfilled were done by the authors in regards to the acceptance criteria of the user story.

Table 1: Selection of requirements which are relevant for the architecture.
M: Must-be, A: Attractive, I: Indifferent; ✓: Yes, (✓): Partial, ✕: No

No.	Requirement	Kano	Fulfilled
	Functional Requirements		
	Concurrent Engineering		
R1	Multiple engineers can simultaneously access the system model.	M	✓
R2	A workspace can be shared with other engineers.	A	(✓)
	Interfaces & Export of Data		
R3	The system model can be accessed via a web API.	A	✓
R4	The software can trigger web APIs of external software.	I	✓
	Non-functional Requirements		
	Performance Efficiency		
R5	The software scales with the load.	I	✓
	Compatibility		
R6	The software can exchange data with other software.	A	✓
R7	The software can be used with existing hardware.	M	✓
R8	The software can be used from different devices.	M	✓
	Reliability		
R9	The software remains operational on high load.	M	✓
R10	The software remains operational in the event of errors.	M	✓
	Security		
R11	The software uses an encrypted communication channel.	I	✓
R12	The software can be deployed on own hardware.	I	✓
R13	The software can be used without an outside internet connection.	M	✓
R14	The software can only be used by authorized engineers.	M	✓
	Portability		
R15	The software provides easy access to the system model.	M	✓
R16	The software runs on standard hard- and software.	M	✓

4 Derived Architecture

Fig. 1 shows the architecture which is derived from the requirements. We use Kubernetes as an orchestration software to manage the containers in the proposed service-based architecture. This ensures that the software runs on standard hard- and software (R7, R12, R16) and it enables that the software remains operational in the event of high loads or errors (R5, R9, R10). On top of Kubernetes, we deployed Eclipse Che as a workspace server, because it fulfills many standard requirements (R13) like user (R14) or workspace (R1, R2) management as well as exposing individual services (R3, R4, R6, R11). A basic MBSE tool was developed on the basis of Eclipse Theia and EMF.Cloud. It can be used by any device with a web browser (R8, R15). It is packaged as an Eclipse Che workspace with the help of a Devfile, that includes dependencies and software that the tool needs to run (R1, R2, R8, R15). Also, we developed a mass calculation and a model checking service as internal services which can be used by every workspace. An internal service is a containerized application which provides functionality via a web API. Providing an application as an internal service may be necessary in case an application exceeds the hardware limits of a workspace. A command-line interface (CLI) tool was developed to demonstrate the integration of external tools as an additional way to access the system model (R6, R15). External services like a version-control system (DLR Gitlab) for accessing project files and collaboration (R2) or a public container image registry for deployment of the cluster are needed. With the use of a version-control system, the system model can also be modified outside the cluster without an internet connection (R13). The private image registry contains images of self developed tools which shall not be publicly available and is also needed in an offline scenario (R13).

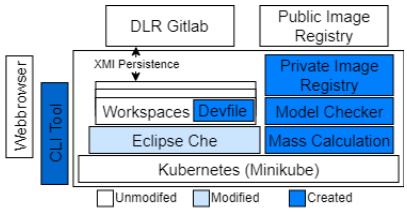

Fig. 1: Proposed architecture of the MBSE tool

5 Evaluation

Within the following evaluation, two research questions (RQs) are answered:

(RQ1) To what extent can the requirements for a space MBSE tool be satisfied using available cloud technologies? In particular, are *versioning*, *product-line management*, *synchronization*, *confidentiality*, and *different views* as well as *abstraction levels* covered?

(RQ2) How does the utilization of cloud technologies influence the performance of calculations and verifications?

For answering **RQ1**, we used the prototype implementation to check which of the requirements can be fulfilled. To answer **RQ2**, we deployed the prototype on Kubernetes as it would be deployed on a cluster. Two kinds of experiments have been conducted. In the *mass calculation* experiment, the masses of the system elements are aggregated, and in the *model checking* experiment, a state machine is searched for deadlocks. To determine the overhead of the architecture, we compare the performance of the experiments to a local deployment. In that case, all functionality is invoked locally, without involving any cloud technologies, web server, or network communication. The experiments have been executed on a computer with an Intel Core i9 11950H processor, 64 GB of RAM, running Windows 10 Enterprise and a virtual machine with Ubuntu 20.04 LTS. Each performance measurement was carried out 1,000 times.

5.1 Results

We first consider the requirements listed in **RQ1**. In general, it is possible to model a structure of a system and attach masses to the elements. The behavior can be modeled with a flow chart and checked for deadlocks. Constraints on the flow chart can be defined by a DSL and checked in a model analysis. *Synchronization* and *versioning* of the model is covered by the utilized version-control system. Using branching, simple *product-line management* can be realized. The *confidentiality* of the data during communication is protected by encryption. But, the data storage is unprotected in the current implementation. Providing different *views on the data model* and *different abstraction levels* is possible by using EMF.Cloud and EMF. They allow the definition of extensible data models as well as the creation of different editors, e.g., tree editors and flow diagrams. Fig. 2 shows the fulfillment of the requirements regarding their Kano categorization and requirement group.

Checking the requirements for **RQ1** revealed that 28 requirements are fulfilled, 11 are partially fulfilled, and 13 are not satisfied.

Fig. 3 and Fig. 4 show the results of the performance measurements for a sequential and a concurrent execution of the experiments, respectively. In the diagrams, the median time and 99% percentile of the measurements are shown. For the mass calculation, the cluster is significantly slower, while the model checking shows similar performance, especially for concurrent execution.

Answering **RQ2**, we conclude that cloud technologies introduce a measurable overhead in case the performed tasks are simple and fast. For longer lasting computational tasks, the overhead diminishes.

5.2 Discussion

The evaluation of satisfied requirements showed that most of the non-functional requirements are satisfied, but the satisfaction of several functional requirements

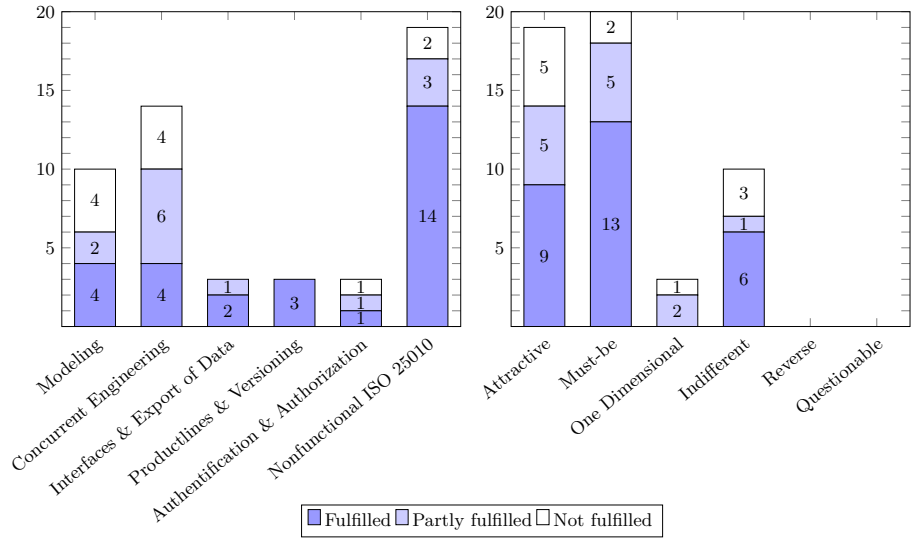

Fig. 2: Requirement fulfillment w.r.t. to categories (left) and Kano (right)

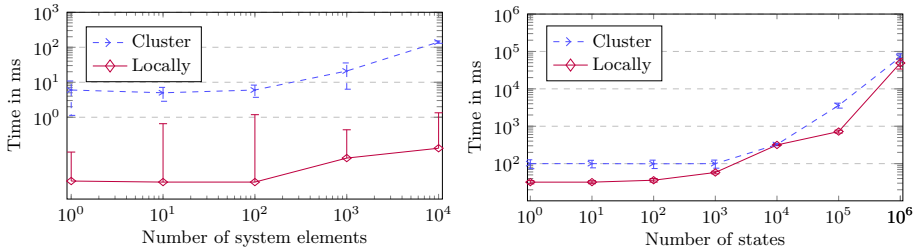

Fig. 3: Sequential execution of mass calculation (left) and model checker (right)

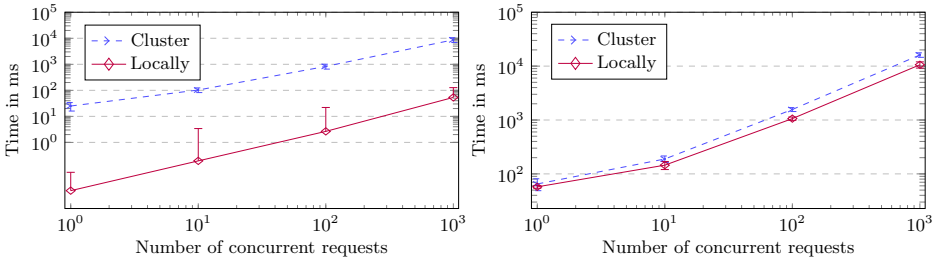

Fig. 4: Concurrent execution of mass calculation (left) and model checker (right)

is lacking. This means that there is no fundamental issue in utilizing cloud technologies for MBSE tooling and that for reaching full functionality, more implementation work is required. The performance measurements indicate that cloud technologies indeed introduce an overhead. For practical sized models, the overhead is still well below one second. With more compute-intensive operations, the overhead is negligible.

6 Summary and Outlook

In this paper, it was investigated what engineers expect from an MBSE cloud tool. It was shown that with current cloud-native technologies, an MBSE tool can be created. In the proposed architecture, compute-intensive operations scale better than simple operations. In future work, the architecture could be deployed in a real cloud environment as well as tested in a concurrent engineering study.

References

1. Calio, E., Giorgio, F.D., Pasquinelli, M.: Deploying Model-Based Systems Engineering in Thales Alenia Space Italia. In: INCOSE Italia Conf. on Systems Engineering (2016)
2. Carrión, C.: Kubernetes Scheduling: Taxonomy, Ongoing Issues and Challenges. ACM Computing Surveys **55**(7), 1–37 (Dec 2022). https://doi.org/10.1145/3539606
3. Fischer, P.M., Lüdtke, D., Lange, C., Roshani, F.C., Dannemann, F., Gerndt, A.: Implementing model-based system engineering for the whole lifecycle of a spacecraft. CEAS Space Journal **9**(3), 351–365 (Jul 2017). https://doi.org/10.1007/s12567-017-0166-4
4. Gateau, T., Senaneuch, L., Cordero, S.S., Vingerhoeds, R.: Open-source Framework for the Concurrent Design of Cubesats. In: 2021 IEEE Intl. Symp. on Systems Engineering (ISSE). pp. 1–8 (2021). https://doi.org/10.1109/ISSE51541.2021.9582549
5. Gornev, I., Liakh, T.: RIDE: Theia-Based Web IDE for the Reflex Language. In: 2021 IEEE 22nd Intl. Conf. of Young Professionals in Electron Devices and Materials (EDM). IEEE (Jun 2021). https://doi.org/10.1109/edm52169.2021.9507678
6. Herrmann, A.: Grundlagen der Anforderungsanalyse: Standardkonformes Requirements Engineering. Springer Fachmedien Wiesbaden (2022). https://doi.org/10.1007/978-3-658-35460-2
7. ISO/IEC 25010: ISO/IEC 25010:2011, systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models (2011)
8. Kano, N., Seraku, N., Takahashi, F., Tsuji, S.: Attractive quality and must-be quality. The Journal of the Japanese Society for Quality Control **14**, 39–44 (01 1984)
9. Saini, R., Bali, S., Mussbacher, G.: Towards Web Collaborative Modelling for the User Requirements Notation Using Eclipse Che and Theia IDE. In: 2019 IEEE/ACM 11th Intl. Workshop on Modelling in Software Engineering (MiSE). IEEE (May 2019). https://doi.org/10.1109/mise.2019.00010
10. Shahin, A., Pourhamidi, M., Antony, J., Park, S.: Typology of Kano models: A critical review of literature and proposition of a revised model. Intl. Journal of Quality & Reliability Management **30**, 341–358 (03 2013). https://doi.org/10.1108/02656711311299863