Towards a Metamodel for Modular Simulation Environments

Sandro Koch, Frederik Reiche, and Robert Heinrich

(sandro.koch,frederik.reiche,robert.heinrich)@kit.edu
Chair for Software Design and Quality
Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany

Abstract. Long-living systems often cope with fast-changing trends in the current market. Changes originating from new features or bug fixes can create great effort. Therefore, it is desirable to know the impact of the changes on quality aspects of a system beforehand. Simulating the system allows estimating the impact of a change without the time and cost investment of an actual implementation. Nevertheless, the code quality of simulations is often neglected. This results in increased development cycles for the simulation until the cycle time is no longer sustainable. Therefore, we propose using a metamodel to describe modular simulations. This will allow reusing simulations and because of the modularity, the simulations will be composable. Consequently, it will also reduce maintenance and development efforts because simulations can be smaller designed and reused. Smaller means that a simulation could serve only one purpose. In this paper, we propose an excerpt of our metamodel with the focus on coupling and interoperability of simulations.

1 Introduction

In software engineering designing and running long-living systems is a hot topic. Software is getting older. Distinct releases with long development cycles are disappearing. Instead, short development cycles and incremental releases are advancing. Companies like Microsoft (Office365) or Adobe (CreativeCloud) use subscription models for their software portfolio. Therefore, maintenance and implementing new features while not breaking the code are getting more important. To achieve short development cycles without breaking the software system, the impact of these changes should be known beforehand. A simulation allows analyzing impacts to a software system without actually implementing these changes [10]. As Tolk et al. [13] remark, simulations reduce development time and thus costs. Consequently, simulations are a key tool to develop and maintain software systems. However, developing simulations does not emphasize reusing or maintaining the simulation itself. Simulations are mainly developed ad-hoc. As a result, maintaining a simulation alongside the main system will increase the duration of development cycles and thus, the time to market of the software. We intend to improve the development of simulations by using a metamodel. Therefore, we have analyzed existing approaches in *simulation*

modeling, and *simulation coupling and interoperability*. Challenges to consider when modularizing simulations are described in our previous work [7]. In this paper, we want to introduce the current state of our metamodel while presenting excerpts of the metamodel. The paper is organized as follows. A motivating example is introduced in Sec. 2. State of the art is described in Sec. 3. Sec. 4 gives an overview of the developed metamodel. The paper concludes in Sec. 5.

2 Motivating Example

As a motivational example, we will introduce two independent simulations. One simulation analyzes a public transport system with bus stops and buses. Each bus has a fixed number of seats and is part of a bus line. A bus line consists of a number of bus stops. These bus stops must be visited in a given order. Furthermore, people waiting at a bus stop will enter a bus if not all seats are occupied. If all seats of a bus are occupied, the remaining people will wait for the next bus to arrive. The second simulation simulates a person's daily routine. However, the routine is simplified and does not represent a real person's life. A person has a fixed home and workplace. The way to work can be walked or driven by public transportation. Each person has two bus stops which have to be used on their way to work. Whether a person walks or drives can be random, fixed or also be influenced by the weather. The difference between random and weather is that the first is individual for each person and the second affects a batch of people. Nevertheless, bus stops are present in both simulations. The population of the bus stops of the bus simulation is normally generated within the bus simulation. However, now the bus stops will be populated by the people of the life simulation. Therefore, the two simulations will affect each other. We will introduce how these two simulations can be coupled. Hence, in this paper, we focus on the coordination of the data exchange of these two simulations.

3 State of the Art

Approaches to improve the evolution of software simulations are divided into two categories. The *composition* of simulations and the *interoperability* between simulations, and the *modeling* of simulations. Approaches to composition and interoperability are presented together.

Simulation Composition and Interoperability Reusing a simulation or parts of a simulation require composability and interoperability. The Distributed Interactive Simulation (DIS) [5] approach is decentralized. Data is specified at the protocol level. Information about the whole simulation has to be present for each participant. The successor of DIS is the High Level Architecture (HLA) [4]. Information is stored by a central manager, the Runtime Infrastructure (RTI). Composable Discrete-Event scalable Simulation (CoDES) by Teo et al. [12] is a component-based approach. It allows semantic composition of simulations if

implemented strictly according to the CoDES specification. Zeigler et al. [15] developed the Discrete Event System Specification (DEVS) approach. DEVS allows formal definitions for simulation states and transactions between simulations. However, these approaches are very restrictive and cannot easily be mixed with other approaches.

Simulation Modeling Data and behavior of an HLA simulation has been modeled by Topcu et al. [14]. Scerri et al. [11] introducing agent-based models and shared variables to extend the HLA approach. In the context of autonomous systems, the modeling language Systems Modeling Language (SysML [6]) was extended by Bocciarelli et al.[2]. The work of Benjamin et al. [1] asserts that an ontology facilitates the modeling of simulation. Model-driven development approaches are utilized by Cetinkaya et al. [3]. Despite not using an ontological or metamodeling approach, Law's reference book "Simulation Modeling and Analysis" [9] is giving an overview of general modeling approaches in the domain of simulations. All approaches have in common that the composition and interoperability aspect is not considered.

4 Modular Simulation Environment Metamodel

If a simulation has to be coupled with other simulations the coupling and communication have to be managed. In [8] the coupling on an abstract level is described. However, the management of the connected simulations is not considered. Management of simulations is to ensure data that must be exchanged is sent to the right simulation at the right time. Additionally, the sent data must be in the right format for the receiving simulation. Accordingly, we introduce a management element for coupling simulations. The Modular Simulation Environment (MSE) fulfills a similar role as the HLA RTI. In contrast to the RTI, the MSE describes the coupling on an architectural level. We will introduce the MSE and its advantages in contrast to the aforementioned approaches in Sec. 3. Fig. 1 shows an excerpt of our metamodel with a focus on the MSE. The metamodel and model of the bus and human simulation coupling can be found on GitHub[1].

Coordinator The *Coordinator* serves as the root element of the MSE. All elements regarding connection, data specification, and management are part of the *Coordinator*. A *Coordinator* stores all *Connectors* necessary for connecting simulations to the *Coordinator*. Multiple simulations can be connected to the *Coordinator*. In contrast to the HLA, several simulations coordinated by different *Coordinators* can be combined to create a more extensive simulation. Consequently, specific parts of a simulation can be encapsulated with a dedicated Coordinator and if necessary combined to form a substantial simulation. All data types available to the *Coordinator* are defined in the *DataSpecificationContainer*. Therefore, no simulation has to know which other data types are used by other simulations connected via the *Coordinator*. The *Annotation Interface* allows data

[1] https://github.com/MoSimEngine

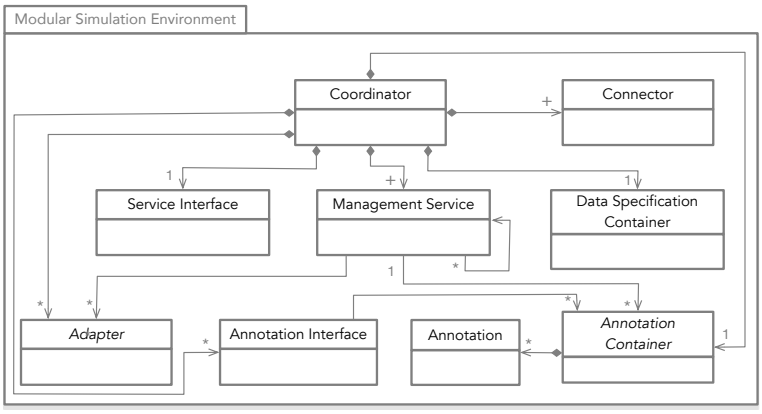

Fig. 1. Modular Simulation Environment (MSE)

exchange between simulations. Even if these simulations are developed independently. Conversion information necessary for the *Coordinator* is defined in the *Service Interface*. The *Service Interface* provides a set of coordinator functions callable by the connected simulations.

Connector The *Connector* enables communication between a simulation and the MSE. Each simulation and the *Coordinator* have to implement a *Connector*. The *Connector* of a simulation invokes functions of the *Coordinator*. These functions are provided by the *Connector*. However, an existing simulation will not have a *Connector* automatically built in. Therefore, a wrapper has to provide the *Connectors* functionality. The Wrapper will encapsulate the simulation. If changes will occur either for the simulation or the MSE, it will only affect the wrapper but not the whole system. A concrete realization of a *Connector* is not part of our metamodel, because the actual implementation is depending on the domain. Each simulation has to implement a *Connector* in order to access the *Connector*. Regarding the motivating example, a *Coordinator* has to be implemented. Otherwise, no data could be received or sent by the simulations.

Adapter In order to combine different simulations, the required and provided data of these simulations need to be aligned. The problem is that each simulation can require and provide different data types. As a result, a simulation has to know what data types are provided by the other simulations to enable a data exchange. However, reusing simulations may require a change when it will be coupled with new simulations. Therefore, we propose the *Adapter*. An *Adapter* can be developed independently of the used simulations and the MSE. For instance, two simulations using time information but one calculates in hours and the other in seconds. Thus, if these two simulations need to exchange time information the transformation can be defined in the *Adapter*. The motivating example needs one *Adapter*. The adapter transforms the walking distance of the human simulation from a floating point value to an integer value.

Annotation Interface Data provided by a simulation must have context information, so that the data can be processed by the *Coordinator*. Therefore, we introduce the *Annotation Interface* to allow annotating context information to a datum. *Annotations* are a part of the *Annotation Interface*. These *Annotations* can be divided into two categories. The *Scheduling Annotation* defines in which order data has to be sent by the *Coordinator* to the receiving simulation. Further, the *Update Annotation* defines under which circumstances a datum has to be sent by the simulation. Three types of *Update Annotation* can be specified. Either on a fixed time interval, if a predefined condition occurs, or if a simulation requests a datum. Regarding the motivating example, we defined one *Annotation* for the incoming data order, which is ordered by timestamp.

Management Service The *Management Service* is responsible for the internal processes of the coordinator. For instance, a dedicated time management service is responsible for the advancing time of the simulation. Another *Management Service* may handle data in the *Coordinator*, it creates and destroys data if necessary. In order to handle data, the defined annotations which are stored in the *Annotation Container*, are accessible by the management services. The *Annotation Container* is stored in the *Coordinator*, thus an *Annotation* can be defined at a central point and reused if necessary. Also, the ownership of the data can be supervised by a *Management Service*. Each *Management Service* fulfills one purpose and can be reused by other coordinators. For the motivating example, three *Management Services* have to be implemented. A publishing service sends events to the right receiver (e.g., number of waiting persons at a bus station when a new person arrives). A subscribing service allows the subscription for events and a time advancing service manages the triggering of events regarding the time advancement.

Data Specification Container All data types present in the *Coordinator* are defined in the *Data Specification Container*. A central specification allows reusing the defined data types for different simulations. Furthermore, if a change requires to modify a data type then the modification can be made at the *Data Specification Container*. The *Data Specification Container* of our motivating example contains an `Integer` (number of seats, occupied seats, waiting persons and distances), a `String` (bus line name, person name, etc.), and a `Bool` (is raining) data type.

5 Conclusion and Future Work

In this paper, we proposed the metamodel for a modular simulation environment. We introduced a motivating example with a bus and a human simulation. These two simulations had to be coupled to create a more detailed simulation. Based on these two simulations, we described the necessity of the *Modular Simulation Environment Metamodel*. The purpose and functionality of each metamodel element were explained. We have shown that our metamodel allows describing the coupling and interoperability on an architectural level. However, the metamodel

has not been used for an actual coupling scenario. Therefore, we have to evaluate our metamodel as the next step. We have to ensure that the behavior of the two simulations is not negatively affected by the coupling. Thus, the behaviour of a simulation must be integrated in the metamodel. Currently we are working on a Domain Specific Language which uses the metamodel as base. Our goal is to allow a automated generation of the code which is necessary for the coupling.

Acknowledgement

This work was partially supported by the MWK (Ministry of Science, Research and the Arts Baden-Württemberg) in the funding line Research Seed Capital (RiSC). We would like to thank Prof. Shmuel Tyszberowicz for his valuable input.

References

1. Benjamin, P., et al.: Using ontologies for simulation modeling. In: Winter Simulation Conference. pp. 1151–1159. IEEE (2006)
2. Bocciarelli, P., et al.: A model-driven framework for distributed simulation of autonomous systems. In: Proceedings of the Symposium on Theory of Modeling; Simulation: DEVS Integrative. pp. 213–220. DEVS '15 (2015)
3. Cetinkaya, D., et al.: MDD4MS: A model driven development framework for modeling and simulation. In: Proceedings of the Summer Computer Simulation Conference. pp. 113–121. Society for Modeling & Simulation International (2011)
4. Dahmann, J.S., et al.: The department of defense high level architecture. In: 29th conference on Winter simulation. pp. 142–149. IEEE (1997)
5. DIS Steering Committee, et al.: The DIS vision: A map to the future of distributed simulation (1994)
6. Friedenthal, S., et al.: A practical guide to SysML: The systems modeling language (2012)
7. Koch, S.: Challenges in modularization of discrete event simulations. In: Collaborative Workshop on Evolution and Maintenance of Long-Living Software Systems. CEUR-WS (2018)
8. Koch, S.: Towards semantic composition of event-based simulation. In: 5th Design For Future Workshop. Softwaretechnik-Trends (2018)
9. Law, A.M., Kelton, D.W.: Simulation modeling and analysis. McGraw-Hill (1991)
10. Reussner, R.H., et al.: Modeling and simulating software architectures: The Palladio approach. MIT Press (2016)
11. Scerri, D., et al.: An architecture for modular distributed simulation with agent-based models. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1. pp. 541–548 (2010)
12. Teo, Y.M., Szabo, C.: CoDES: An integrated approach to composable modeling and simulation. In: Simulation Symposium, 2008. ANSS 2008. 41st Annual. pp. 103–110. IEEE (2008)
13. Tolk, A.: Metamodels and mappings - ending the interoperability war. In: Fall Simulation Interoperability Workshop (2004)
14. Topçu, O., et al.: Distributed Simulation - A Model Driven Engineering Approach. Springer (2016)
15. Zeigler, B.P.: Multifacetted modelling and discrete event simulation. Academic Press (1984)