Executable Behavioral Models in Model-based Systems Engineering using Capella

Philipp Chrszon[1], Anton Donner[2], Moritz Edelhäuser[2], Lucia Andrea Jara Garcia[2], Robert Uebelacker[2], Philipp M. Fischer[1], and Andreas Gerndt[1,3]

[1] Institute for Software Technology, German Aerospace Center (DLR), Braunschweig, Germany,
`philipp.chrszon@dlr.de`, `philipp.fischer@dlr.de`, `andreas.gerndt@dlr.de`
[2] Galileo Competence Center, German Aerospace Center (DLR), Weßling/Oberpfaffenhofen, Germany,
`anton.donner@dlr.de`, `moritz.edelhaeuser@dlr.de`, `lucia.jaragarcia@dlr.de`, `robert.uebelacker@dlr.de`
[3] University of Bremen, Bremen, Germany

Abstract. Methodologies and tools for model-based systems engineering often not only support the specification of a system's structure, but also its behavior. Behavioral diagrams such as state machines and sequence diagrams allow the description of mode-dependent behavior and interaction scenarios, respectively. The key characteristic of these behavioral diagrams is their inherent dynamic nature which allows them to be executed. Having tool support for execution can promote a better understanding of the system, allows consistency checking, and even enables the application of verification techniques.
In this paper, we present an add-on for the open-source MBSE tool Capella supporting the execution of state machines, either manually or driven by scenarios. The add-on is targeted towards facilitating the integration of approaches for checking the consistency and correctness of the modeled system.

Keywords: Model-based systems engineering · Behavioral modeling · Simulation

1 Introduction

The validation of the correct and expected behavior of a system before its deployment is of utmost importance, especially when developing a system for space applications. An unexpected and unintended behavior of the system cannot be corrected by on-site maintenance, which in turn may lead to partial or complete mission failure.

In this paper, we present an approach that allows incorporating the validation of the system's behavior in the model-based systems engineering (MBSE) process during the design phase. To this end, we have implemented a simulation add-on for the MBSE tool Capella.

Integrating the validation of behavior into the design process has two main advantages. First, problems in the spacecraft architecture can be identified very early in the design process. Then, corrections are possible without massive cost and schedule impact, as would be the case if the problems were identified later during testing of real hardware and software. Second, it allows for a more holistic analysis of the system. Model-checking approaches can be applied to cover more than just specific use cases like in traditional test campaigns.

Among the many advantages obtained by modeling state machines as part of the MBSE approach, the following stand out for space applications [4].

- *Identification and recovery of hazardous states.* Executable state machines help identify undesirable states or deadlocks/livelocks during system design. The occurrence of these deadlocks can be reproduced and understood by simulating the state machines.
- *Validation of requirements and operational procedures.* Requirements and operational procedures can be modeled in most MBSE tools. Both can be traced to a certain state-transition sequence that can be validated within the MBSE environment.
- *Fault detection and isolation mechanisms.* Recovery measures can be validated directly in the model when traced to an executable state machine.

Checking the behavior model of a spacecraft not only allows unhealthy or unexpected state combinations to be identified. It also enables the validation of specific command sequences required to successfully operate the spacecraft by issuing the correct command sequence to activate the desired function.

In contrast to (semi-)static mass budgets, both data rates (on-board and space-to-ground) and power consumption result from dynamic processes. Generally, commands and external events trigger different payload operational states. Especially in early design phases, it is of key importance to dimension data links and power systems properly—or to identify command sequences leading to overload situations.

Although there are existing approaches integrated into MBSE tools that combine modeling with simulation of system behavior, none of them satisfy our requirements completely. Specifically, the tool itself must be open to unrestricted modification, since a primary concern is its usage in applied research. Additionally, it should be able to simulate operational procedures, allow the identification of deadlocks/livelocks, and provide the possibility to interface with external verification tools such as model checkers.

2 Related Work

This section gives an overview of the related work on behavioral models and simulation as well as their use and support in MBSE tools, particularly in Capella.

2.1 State Machines

There are several standards for graphical modeling notations which encompass behavioral models, including state-machine diagrams and sequence diagrams, also called message sequence charts. The Specification and Description Language (SDL) is designed for distributed systems in general and telecommunications in particular. While the Unified Modeling Language (UML) is mainly applied in software design, the Systems Modeling Language (SysML), originally derived from UML, is targeted towards systems engineering activities.

Basic state machines consist of the core elements state and transition. A *state* represents a situation in which the system is, that could be stable or linked to some quasi-stationary behavior or action. A *transition* represents the change of the state from one to another, like from the off-state of a system to the on-state. Transitions can be triggered by conditions or events.

2.2 Behavior Modeling in MBSE Tools

Simulation, a feature provided by several MBSE tools, may be employed for an early verification of the system behavior during modeling. In general, the actual simulation approach depends on the chosen tool. In some tools, simulation is fully integrated, while others offer import and export functionalities to use the model data within specialized simulation tools. Without claiming to be exhaustive, the following is an overview of common tools:

- For Cameo Systems Modeler, the Cameo simulation toolkit is available. Besides this integrated solution, Cameo also provides export capabilities targeting simulation tools like Matlab, Simulink, and Modelica.
- Enterprise Architect offers the connection to different simulation toolkits including Matlab, Simulink, and OpenModelica.
- OpenModelica includes the modeling of clocked state machines, immediate and delayed transitions, conditional data flows, and hierarchical state machines, which can be simulated by using OMSimulator.
- For Virtual Satellite, an approach that integrates simulation for verifying the satisfaction of mission requirements has been presented [7]. Furthermore, state machines modeled in Virtual Satellite can be analyzed using model checking to detect, e.g., deadlocks and livelocks. For this, a prototype integration of multiple model-checking tools has been implemented [3].
- Capella does not implement an out-of-the-box integrated simulation environment, but there are 3$^{\text{rd}}$-party add-ons available. The add-on Dynamic Execution and System Simulation (DESS) by PGM is a tool-integrated solution, whereas other add-ons allow the export of model data into simulation software like Ansys ModelCenter. Siemens also offers a Capella integration within the Siemens product lifecycle management (PLM) solution Teamcenter, connecting the model to several other Siemens design and simulation tools.

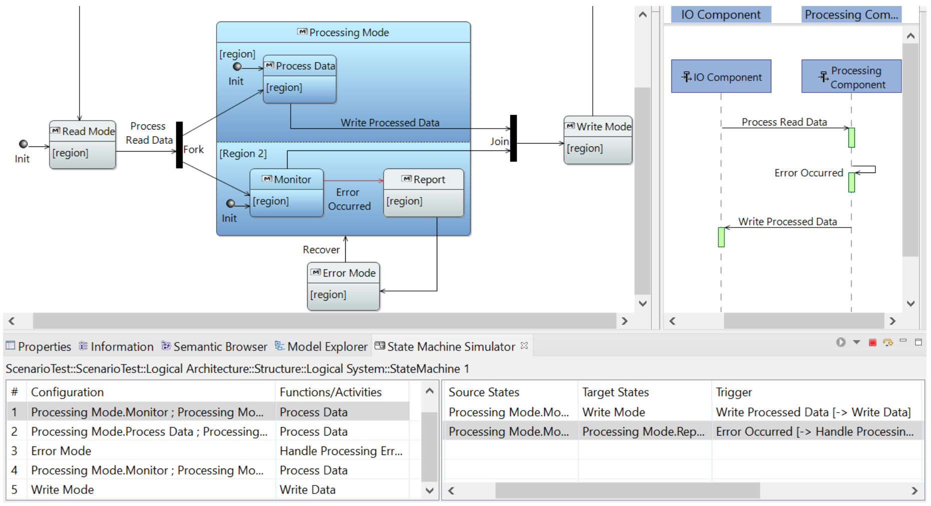
Capella was chosen as the basis for this work because it is an open-source tool and provides all the necessary interfaces for data exchange as well as custom extensions.

2.3 Behavior Modeling in Capella

In Capella [6] and the respective Arcadia method [8], behavior of a system can be modeled and described using "mode and state machine" diagrams. These diagrams are inspired by UML/SysML diagrams, to which they are largely equivalent. Capella distinguishes between a mode of a system, which is an expected behavior of a system (e.g., triggered by an operator), and a state of a system, which describes the behavior of the system imposed through external influences such as environmental conditions. States and modes cannot be mixed within the same machine. A further difference in Capella is the direct coupling with the functional system model. Functions, which may be active in a mode of a mode machine, are part of the functional model and are selected from there. It is also possible to define which functions are available in a mode but not necessarily active when the mode is entered. Similar to modes, also transitions can be linked to the functional model. In particular, the trigger that initiates the firing of a transition can be a functional exchange, i.e., a connection or data flow between functions, or an exchange item. Additionally, transitions may be triggered by timers or change events like in SysML.

Scenario diagrams describe the sequence of functions for a set of components carrying out a specific capability of the system. Here, modes of mode machines can be referenced to specify conditions under which certain functionalities are available.

3 State Machine Simulation Add-on



ScenarioTest::ScenarioTest::Logical Architecture::Structure::Logical System::StateMachine 1

#	Configuration	Functions/Activities
1	Processing Mode.Monitor ; Processing Mo...	Process Data
2	Processing Mode.Process Data ; Processing...	Process Data
3	Error Mode	Handle Processing Err...
4	Processing Mode.Monitor ; Processing Mo...	Process Data
5	Write Mode	Write Data

Source States	Target States	Trigger
Processing Mode.Mo...	Write Mode	Write Processed Data [-> Write Data]
Processing Mode.Mo...	Processing Mode.Rep...	Error Occurred [-> Handle Processin...

Fig. 1. A state machine during execution of a scenario. The selected configuration and transition are highlighted.

This section gives an overview of our state machine simulation add-on and its capabilities. The add-on provides the State Machine Simulator view, as shown in the bottom half of Fig. 1. During an execution, the left table shows the execution trace, i.e., the sequence of configurations, where each configuration comprises one or more states. The right table lists the transitions that are enabled in the currently selected configuration. The view is synchronized with the diagram editors in which the selected configuration and transition are highlighted in blue and red, respectively.

The underlying execution engine of the simulator is based on the formal semantics for UML state machines presented in [5]. Table 1 provides an overview of the currently implemented syntactic constructs. Both orthogonal regions within states as well as fork and join pseudostates are supported, allowing for modeling and executing concurrent behavior. Furthermore, support for sub-state-machines, i.e., nested states, is fully implemented, enabling a hierarchical modeling of complex systems.

The simulator supports both a manual, user-guided exploration and an automatic exploration of the configuration space. In the manual mode, the next configuration is chosen by selecting and executing one of the enabled transitions using the right table in the simulator view. This mode is especially useful during the initial creation of the state machine, as it allows a quick identification of modeling issues as well as missing transitions or states. Moreover, a manual exploration may be useful for explaining and illustrating the system's behavior in discussions with engineers or stakeholders.

Table 1. Syntactic constructs supported by the simulator compared to the underlying semantics [5] (classification adapted from [1], RTC: run-to-completion).

	states					transitions					
	ortho	sub	fork/join	choice	history	internal	inter-level	completion	RTC	variables	time
semantics [5]	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✖
simulator	✔	✔	✔	✖	✖	✔	✔	✖	✔	✖	✖

In the automatic mode, a selected scenario, e.g., a functional exchange scenario, is used to drive the execution by replaying the scenario's messages as triggers on the state machine. Consider again the screenshot in Fig. 1. Here, the exchange scenario on the right has been executed on the displayed state machine, creating the trace on the bottom left. The execution trace not only contains the final state reached in the scenario (Write Mode), but also all intermediate states that have been visited. Within a scenario execution, intermediate steps are allowed and are automatically taken as long as the execution remains deterministic. For instance, after the Error Occurred trigger has been received, the Report mode and Error Mode are visited before returning to the Processing Mode, where the next trigger (Write Processed Data) is executed. In case a scenario cannot be executed until the end, either because of a missing fitting transition or a nondeterministic choice, the execution stops and the simulator switches to the manual mode. Then, the partial execution trace and current configuration may help to either amend the state machine or the scenario. Since scenarios can be generated automatically from functional chains, scenario execution allows checking whether the state machine actually implements a functional chain from start to end. Additionally, if scenarios or state machines are linked with requirements, the add-on also enables checking the satisfaction of these behavior-related requirements.

The add-on, including the drop-in for installation in Capella, and its source code are provided in the accompanying artifact [2].

4 Conclusion and Outlook

We have presented an add-on for the MBSE tool Capella that allows for the step-wise execution of state machines, both manually and automatically. It supports the execution of exchange scenarios and functional chains, enabling a straightforward consistency checking.

The presented work can be extended in several directions. First, the simulator itself might be extended to support also history and choice pseudostates. Additionally, support for more complex scenarios involving, e.g., loop and alternative fragments, might be integrated. Second, the simulation of state machines may be complemented by formal verification, in particular model checking, since simulation alone cannot cover all possible scenarios and behaviors. For this, we plan to integrate state-of-the-art model-checking tools, similar to the approach presented in [3]. In case of violated requirements, the counterexample returned by the model checker could then be automatically imported into the state-machine simulator as an execution trace, aiding the comprehension of the faulty behavior.

References

1. André, E., Liu, S., Liu, Y., Choppy, C., Sun, J., Dong, J.S.: Formalizing UML state machines for automated verification – a survey. ACM Comput. Surv. **55**(13s) (jul 2023). `https://doi.org/10.1145/3579821`
2. Chrszon, P.: State machine simulator add-on (Feb 2024). `https://doi.org/10.5281/zenodo.10657885`
3. Chrszon, P., Maurer, P., Saleip, G., Müller, S., Fischer, P.M., Gerndt, A., Felderer, M.: Applicability of model checking for verifying spacecraft operational designs. In: 26th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2023, Västerås, Sweden, October 1-6, 2023. pp. 206–216. IEEE (2023). `https://doi.org/10.1109/MODELS58315.2023.00011`
4. Harris, J.A., Patterson-Hine, A.: State machine modeling of the space launch system solid rocket boosters. NASA Aeronautics Scholarship Program — Internship Final Report (2013), `https://ntrs.nasa.gov/citations/20160000328`, NASA Ames Research Center
5. Liu, S., Liu, Y., André, É., Choppy, C., Sun, J., Wadhwa, B., Dong, J.S.: A formal semantics for complete UML state machines with communications. In: Johnsen, E.B., Petre, L. (eds.) Integrated Formal Methods, 10th International Conference,

IFM 2013, Turku, Finland, June 10-14, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7940, pp. 331–346. Springer (2013). `https://doi.org/10.1007/978-3-642-38613-8_23`

6. Roques, P.: Systems Architecture Modeling with the Arcadia Method - A Practical Guide to Capella. iSTE (2018), ISBN: 978-1-78548-168-0
7. Schaus, V., Fischer, P.M., Lüdtke, D., Tiede, M., Gerndt, A.: A Continuous Verification Process in Concurrent Engineering (2013). `https://doi.org/10.2514/6.2013-5429`
8. Voirin, J.L.: Model-based System and Architecture Engineering with the Arcadia Method. Elsevier (2018), ISBN: 978-1-78548-169-7