Verifying the Co-Simulation Orchestration Engine for INTO-CPS*

Casper Thule

Aarhus University, Department of Engineering
Finlandsgade 22, 8200 Aarhus N, Denmark
casper.thule@eng.au.dk

Abstract. The development of Cyber-Physical Systems often involves cyber parts controlling physical artefacts and this interaction is challenging. Therefore, it can be useful to create models of the constituent components and simulate them in a co-simulation to help discover undesired behavior. It is crucial that the tool performing co-simulation does so correctly, and an approach to verifying this is to give semantics to the parts involved in a co-simulation and thereby enable formal verification.

Keywords: co-simulation, FMI, UTP, VDM-RT, Modelica, semantics, CyPhyCircus, Cyber-Physical Systems.

1 Introduction

Cyber-Physical Systems (CPSs) consist of cyber elements controlling physical entities in a dependable way. This interaction is challenging because of the inherent heterogeneity in such systems. As CPSs are becoming increasingly complex, it can be useful to model the constituents that are combined to form a given system. Different disciplines typically use different formalisms to represent constituent models, but here we will restrict ourselves to Discrete Event (DE) and Continuous Time (CT) models. Such constituent models can be simulated in a collaborative simulation (co-simulation), which is capable of coupling models creating with different formalisms. The co-simulation uses the components to simulate the entire system and exchanges data as the common simulated time progresses. It is important to note that co-simulation deals with approximations of real systems in order to achieve a resonable simulation speed. The result of a co-simulation is a trace, which is a collection of outputs from the models at different points in time. The correctness of a simulation trace depends on how accurately the simulation trace resembles the trace of the real system, and therefore depends on the difference between these. As co-simulations approximate a real system, it can be necessary to define a tolerance relative to the trace of the real system, which the simulation trace must be within to be considered correct. This is further elaborated in Sect. 3.

* INTO-CPS is an acronym for Integrated Tool Chain for Model-based Design of Cyber-Physical Systems

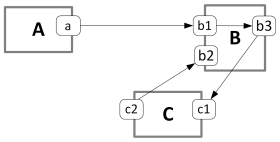

Fig. 1. An example of a simulated CPS with dependencies between SUs (the rectangles A, B and C) via their respective ports (the rounded squares a, b1, b3, b2, c1 and c2).

A co-simulation is typically organised using a master-slave architecture, where the master orchestrates the simulation. The master uses a Master Algorithm (MA) to allow for different kinds of simulations, which will be described below. Figure 1 shows an example of three slaves (simulation units), their input/output ports, and dependencies. A typical co-simulation consists of three phases: Initialisation, simulation and teardown. In initialisation the master retrieves the properties of the simulation units (SUs) and initialises them, chooses an MA and establishes the communication channels. The next phase is simulation, which is an iterative process that repeats until a predetermined end time is reached. An iteration is referred to as a step. In this phase the output values are retrieved from the SUs and the dependencies between SUs are resolved. Afterwards, the SUs are invoked to progress for a determined amount of time called the step size. Once the SUs have performed the step they must respond with a status of whether the step was accepted or rejected. It can be necessary to perform a rollback (if possible) if a step was rejected[1] and rerun the step with a different step size. In the final phase, tear down, the SUs are shut down, results are reported, memory is released and so forth. Different MAs can be used, e.g. a fixed-step-size MA where the step size is predefined and the same for every step. Another version is a variable-step-size MA where the SUs are asked to supply a step size. One can also imagine hierachical co-simulations where some SUs are tightly coupled and must run with a small step size can be combined with other SUs that run with a larger step size. This would consist of different MAs configured with different step sizes.

Because of the heterogeneity involved in co-simulation of CPSs it can be difficult to create a generalised solution, as complex multi-disciplinary systems cannot be modelled naturally in one simulation tool alone [1]. The Functional Mock-up Interface (FMI) standard[2] was created to solve the challenge of representing different systems. The standard describes an interface in the C language and an SU implementing this is called a Functional Mock-up Unit (FMU). Part of this PhD project is to develop an application called the Co-Simulation Orchestration Engine (COE) capable of executing co-simulations based on FMI.

A contribution of this PhD project is to validate the MA used in the COE. This will be carried out by giving semantics to the constructs involved in a co-simulation, which will be described in Sect. 2. The semantics of the constructs

[1] This can happen if the calculated value is outside the aforementioned tolerance.
[2] The standard is located at https://www.fmi-standard.org/

will be used to verify the simulation results of a given simulation and prove certain properties of the MA as described in Sect. 3. Thereby the semantics are used as the ultimate oracle in determining correct simulation results. Finally, Sect. 4 will present the current progress and highlight the contribution of this PhD project.

2 Semantics

To achieve the desired oracle functionality it is necessary to give semantics to the models involved in a given co-simulation and coupling mechanisms. To represent a heterogeneous system the co-simulation will consist of CT models, expressed in Modelica, and DE models, expressed in VDM-RT. An overview of the concepts necessary to achieve the oracle functionality is shown in Fig. 2 and described below.

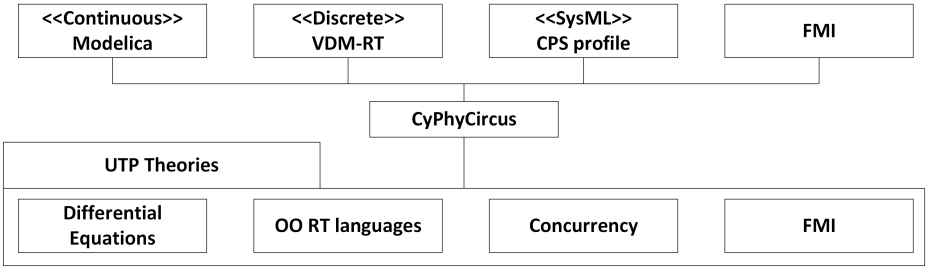

Fig. 2. An overview of the concepts necessary for the oracle functionality.

The Modelica language has been chosen to model CT parts of a system, as the language is non-proprietary and equation based. Models described in the Modelica language can be exported as FMUs using the open-source tool OpenModelica [3].

The discrete parts of a system are modeled in VDM-RT. VDM-RT extends VDM-SL [6] such that it can be used to model and analyse Real-Time embedded and distributed systems, object-orientation, and concurrency [7]. The open-source tool Overture can export VDM-RT models as FMUs.

The connections between models are modeled in the Systems Modeling Language (SysML). SysML has the concept of profiles, which is a way of providing extensions for particular domains. Therefore, to connect models in a correct manner, a SysML profile has been created for CPSs [7].

Furthermore, the FMI standard is written in natural language and thus not subject to formal verification. Therefore it is important to give semantics to FMI in pursuance of formally verifying an MA [2].

CyPhyCircus is a lingua franca for the languages mentioned above. CyPhyCircus is an extension of the Circus language [8], which supports concurrency

and communication in the style of CSP, object-orientation, real-time, and more. CyPhyCircus will contain additional constructs for differential equations, CT, and FMI. All of these concepts will be underpinned by Unifying Theories of Programming (UTP) [5] theories, which will be described below, and therefore CyPhyCircus acts as a front-end for giving semantics to the top elements in Fig. 2.

UTP offers a basis capable of providing a formal specification for a wide variety of programming paradigms. Furthermore, UTP has been mechanised in the Isabelle/HOL proof assistant to aid in proof checking and verification of models. To describe a domain useful for modeling particular problems, one can define UTP theories. UTP theories can be used to express language constructs and thereby act as building blocks for giving semantics to programming languages [4]. Therefore several UTP theories underpin the CyPhyCircus language.

3 Validation

As mentioned above, it is desirable to use the semantics as an oracle in determining correct simulation results. The approach to achieving this is to execute a simulation using the COE, retrieve the trace, model the co-simulation in CyPhyCircus as described in Sect. 2, and then utilize Isabelle to examine the trace. The following is required in order to utilize Isabelle to examine the simulation results: An Isabelle expression capable of describing COE traces and an Isabelle tactic capable of determining whether the trace is valid in the abstract model. As CT is involved it is necessary to perform numerical approximation as part of the checking. This process can be expressed in a commutating diagram as shown in Fig. 3.

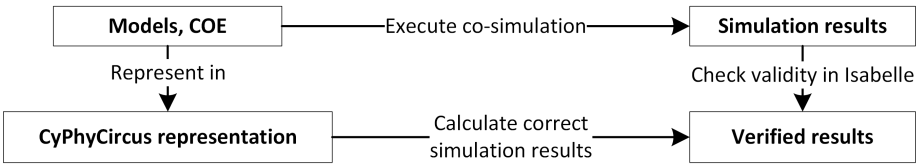

Fig. 3. An overview of the verification process.

The conceptual idea is depicted in Fig. 4 using traces. The green line represents a trace generated from the semantics (STrace) and therefore considered to be correct. The yellow line is the trace of a co-simulation with a step size of one (OneTrace) and therefore contains 15 steps. The blue line is the trace of a co-simulation with a step size of two (TwoTrace) and therefore contains 7 steps. The blue bar represents the tolerance, for which the co-simulation traces must be within. The tolerance is relative to the STrace and can be configured based on the given co-simulation, as different co-simulations have different requirements. One

of the challenges is therefore to define how such a tolerance can be implemented in a sound fashion and coupled with the semantics. In this example it can be seen that the TwoTrace contains three values outside the tolerance (red dots) and therefore the results from the given co-simulation cannot be used. As mentioned above, this can possibly be solved by using smaller step sizes to obtain more granularity as illustrated by the OneTrace.

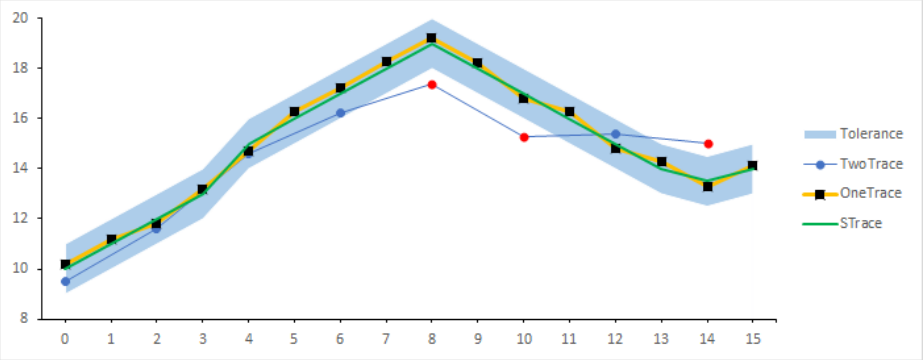

Fig. 4. The conceptual idea of using semantics and tolerance to verify traces.

The verification of the MA deals with complying with the FMI standard. The FMI standard describes a finite state automaton, which MAs must comply with. The approach to verifying the MA is to express it in CyPhyCircus and use the FDR model checker to prove certain properties such as termination, livelock-free (livelocks can appear because of rollbacks), and handling of all error cases [2].

4 Progress and Summary

Currently work is progressing on defining the semantics for VDM-RT, Modelica, FMI, and the SysML profile. This work includes defining the CyPhyCircus language and underpinning UTP theories. We are not directly involved in the definition of the semantics, but once the semantics has been defined it will be possible to begin the validation process described in Sect. 3, which will be the contribution of this PhD project. Furthermore, we take part in developing the COE and additional features are underway such as concurrent simulations [9], distributed simulations across operating systems and hardware, variable step size, and soft real-time simulation.

The research questions posed in this paper are the following:

- Create an Isabelle expression capable of validating COE traces.
- Create an Isabelle tactic capable of determining whether a given trace exists in the abstract model. This includes numerical approximations as CT models are used.

- Define a tolerance and realise it in Isabelle in a semantically sound fashion.
- Create a case study connecting the concepts in Fig. 2.
- Express the MA of the COE in the FDR model checker.

Once a framework containing the elements above is developed, then it can not only be used to verify a single COE, but also to compare different COEs. For example, if one COE performs co-simulations faster than another COE and the simulation results are within the tolerance, then this COE can be used instead. Furthermore, it can be used to verify hierachical compositions of co-simulations, where different MAs are used for different parts of a given co-simulation as mentioned in Sect. 1. The hope is that different COEs and MAs can be combined to produce optimal co-simulations, e.g. in terms of execution time, without risking incorrect simulation results.

Acknowledgements. The work presented here is partially supported by the INTO-CPS project funded by the European Commission's Horizon 2020 programme under grant agreement number 664047. Furthermore, we would like to thank Victor Bandur, Simon Foster, Stefan Hallerstede, Peter Gorm Larsen, Kenneth Guldbrandt Lausdahl and Jim Woodcock for providing valuable input to this paper.

References

1. Bastian, J., Clauss, C., Wolf, S., Schneider, P.: Master for Co-Simulation Using FMI. In: 8th International Modelica Conference (2011)
2. Cavalcanti, A., Woodcock, J., Amalio, N.: Behavioural Models for FMI Co-simulations. In: Accepted for publication at ICTAC 2016. Taipai, Taiwan (2016)
3. Foster, S., Thiele, B., Cavalcanti, A., Woodcock, J.: Towards a UTP semantics for Modelica. In: 6th International Symposium on Unifying Theories of Programming (2016)
4. Foster, S., Zeyda, F., Woodcock, J.: Unifying heterogeneous state-spaces with lenses. In: Accepted for publication at ICTAC 2016. Taipai, Taiwan (2016)
5. Hoare, T., Jifeng, H.: Unifying Theories of Programming. Prentice Hall (April 1998)
6. ISO: Information technology – Programming languages, their environments and system software interfaces – Vienna Development Method – Specification Language – Part 1: Base language (December 1996)
7. Larsen, P.G., Thule, C., Lausdahl, K., Bardur, V., Gamble, C., Brosse, E., Sadovykh, A., Bagnato, A., Couto, L.D.: Integrated Tool Chain for Model-based Design of Cyber-Physical Systems. In: The 14th Overture Workshop: Analytical Tool Chains. Cyprus (November 2016)
8. Oliveira, M., Cavalcanti, A., Woodcock, J.: A UTP semantics for Circus. Formal Aspects of Computing 21(1), 3–32 (2009)
9. Thule, C., Larsen, P.G.: Investigating Concurrency in the Co-Simulation Orchestration Engine for INTO-CPS. In: Preliminary Proceedings of the 10th Anniversary Spring/Summer Young Researchers' Colloquium on Software Engineering). pp. 223–228. ISP RAS, Krasnovidovo, Russia (May 2016)