Co-simulation of Cyber-Physical Systems using HLA

Thomas Nägele
Radboud University
Nijmegen
The Netherlands
t.nagele@cs.ru.nl

Jozef Hooman
Radboud University & TNO-ESI
Nijmegen & Eindhoven
The Netherlands
hooman@cs.ru.nl

***Abstract*—The development of cyber-physical systems (CPSs) with mechanical, electrical and software components requires a multi-disciplinary approach. Moreover, the use of models is important to support trade-offs and design decisions early in the development process. Since the different engineering disciplines use different modelling languages and tools, this calls for a co-simulation framework for discrete and continuous models. The main challenge is the proper synchronisation of time and data between these models. Given the increasing importance of software in CPSs, our work concentrates on the incorporation of software models in such a framework. We have created a proof of concept of a co-simulation framework based on the High Level Architecture (HLA) and Functional Mock-up Interface (FMI) standards. We demonstrate the incorporation of software models expressed in the Parallel Object-Oriented Specification Language (POOSL) in this framework. This allows the use of virtual prototypes of CPSs early in the development process.**

***Index Terms*—Cyber-physical systems, Co-simulation, HLA**

I. Introduction

The development of cyber-physical systems involves a number of different domains and requires close collaboration of software engineers, control engineers, mechanical engineers and thermal engineers. Decisions in one engineering discipline might have a strong impact on other disciplines. In close collaboration with an industrial partner, we work on cases where complex high-tech systems have to be redesigned. One of these cases concerns a cost reduction by using lighter mechanical parts and an investigation whether the consequences of this can be compensated in software.

The general aim of our work is to support the industrial development of these multi-disciplinary systems by modelling and simulation. Ideally, early in the development process models of different disciplines can be simulated together to investigate the impact of design decisions. This would also allow concurrent engineering and, for instance, fault injection to study the robustness of the system. Moreover, models can also be used later in the development process for fast and realistic testing without the need of expensive and scarce lab facilities.

As already observed in [8], such a model-integrated development approach for cyber-physical systems requires new techniques. The main difficulty is that different engineering disciplines in general use different modelling languages and tools. The main challenge in co-simulating models from different domains is time synchronisation. Different domains often use different time scales. Models from the physical domain often describe processes in a continuous manner, whereas software engineers work in a discrete time domain.

In this work we study the possibilities for co-simulation with an emphasis on the incorporation of software models. This becomes increasingly important since a quickly increasing part of current cyber-physical systems is realised by software. After a discussion of related work in Section I-A we will introduce our approach in Section I-B.

A. Related work

An early attempt to combine tools of different disciplines can be found in [7] where a UML-based CASE tool (Rose RealTime) and Simulink[1] have been coupled to allow simultaneous simulation. Establishing a common notion of time appeared to be complicated due to the lack of a proper notion of simulation time in Rose RealTime. To start from a software modelling tool with a solid notion of simulation time, tool support for the Vienna Development Method (VDM) has been used in [5]. VDM models can be co-simulated with continuous-time Bond Graphs in the 20-sim tooling[2].

The research mentioned above is based on dedicated protocols to exchange data and to coordinate progress of time between the various tools. In this paper we investigate the use of standards, that is, the Functional Mockup Interface[3] (FMI) [2] and the High Level Architecture (HLA) [1].

FMI is a standard for model exchange and for co-simulation. The FMI standard specifies interfaces to support the exchange of models and to enable co-simulation of dynamic models as so-called Functional Mock-up Units (FMUs). FMI does not specify how a distributed co-simulation environment should be developed and integrated into a coherent distributed simulation environment.

A possible co-simulation framework is the HLA standard which is a general-purpose architecture for distributed computer simulation systems accompanied with a so-called Run-

[1] http://www.mathworks.com
[2] http://www.20sim.com/
[3] https://www.fmi-standard.org/

978-1-5090-4228-9/17/$31.00 ©2017 IEEE

Time Infrastructure to orchestrate co-simulation of Federates (i.e., simulation entities). The HLA standard consists of an interface specification, an object model template and rules that simulations must obey to be compliant with the interface. The interface specification covers, amongst others, data distribution management and time management, which supports various ways of time synchronization.

HLA-based simulations are typically used to train humans to perform tasks and to experiment with different scenarios in a simulated world. Although HLA was developed for defence applications there is a growing number of applications in other domains. Related to our aim of including software components in simulations is research on the use of HLA for cyber-physical energy system with a combined simulation of both power and ICT systems [6]. Lasnier et al. [9] used the Ptolemy II framework [3] to model and to simulate different federates and exploited HLA to co-simulate them in a distributed environment. Instead of developing a distributed environment, we will focus on how HLA can be used to connect models from different tools.

B. Approach

As mentioned above, our approach is based on FMI and HLA. Given our industrial applications, the use of such standards is highly preferable. Moreover, an advantage of HLA is the availability of open source and commercial tools. This allows both academic experiments and industrial usage.

Both FMI and HLA, however, are biased towards the simulation of continuous-time systems. There is hardly any literature on the incorporation of software models to analyse the complex software architectures of modern high-tech systems. To investigate the incorporation of software models in HLA we use the Parallel Object-Oriented Specification Language (POOSL) [10]. POOSL is a formal language which allows modelling of complex software architectures, including timing aspects. The POOSL tooling[4] allows simulation and debugging of models. Since this approach has already been used to model complex industrial software architectures, co-simulation with models of the continuous environment would be very beneficial.

To investigate the main concepts of HLA, we first have modelled HLA in POOSL, as described in Section II. Next, Section III addresses the incorporation of POOSL models into HLA. Concluding remarks can be found in Section IV.

II. Modelling HLA in POOSL

We describe how HLA has been modelled in POOSL. In HLA, a Run-Time Infrastructure (RTI) handles time management. A set of simulations is called a *federation* and one single simulation is called a *federate*. Federates communicate with the RTI by means of so-called *ambassadors*.

A system-level view of a POOSL model with two federates is depicted in Figure 1. The main components of the system are modelled as instances of *clusters* (indicated by a C). The clusters are connected by channels between the ports of the clusters. For instance, a channel (coloured green) connects port "rtiamb" of the RTI with the ports "rti" of two RTIAmbassadors. Federates call an RTIAmbassador to perform certain actions, such as the update of attribute values and requests to advance their local time. The RTI will inform federates via FederateAmbassadors to reflect changed attributes and

[4] http://poosl.esi.nl/

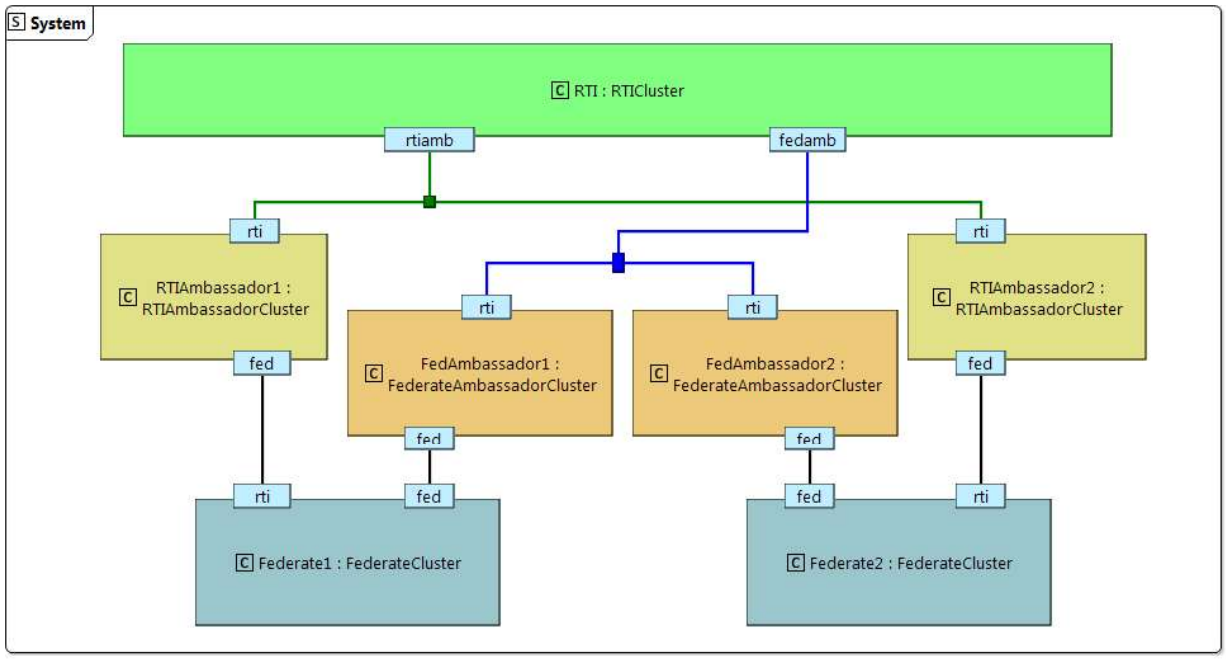

Figure 1. HLA architecture in POOSL

grant time advances. Clusters are further decomposed. As an example, Figure 2 shows the Federate cluster with two processes (indicated by a P). As before, channels connect ports of the processes and the cluster.

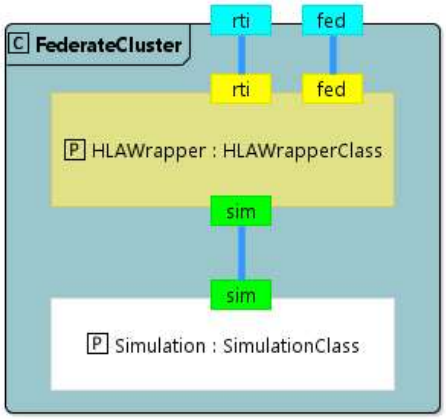

Figure 2. Federate cluster in POOSL.

A POOSL process is specified in an object-oriented language, as illustrated by the abstract model of a simulation in Figure 3. It consists of a set of methods which may specify the sending of messages along ports (indicated by the exclamation mark "!") and the receipt of messages (indicated by the exclamation mark "?"), similar to notations such as CSP and CCS. Communication is synchronous; asynchronous communication can be modelled easily using queues. POOSL also contains constructs to specify data types. Progress of time can be modelled by means of `delay` statements; all other statements are executed without consuming simulation time. POOSL models may contain the built-in socket objects to connect the simulation with other components.

```
process class SimulationClass(i : Integer)
ports sim
messages
    sim?Start
variables msg : String
init Initialise()()
methods
    Initialise()()
        sim?Start; Compute()()
    Compute()()
        sim!Output("msg " + (i printString));
        sim!next; Receive()()
    Receive()()
        sel sim?Input(msg); Receive()()
        or sim?Compute; Compute()() les
```

Figure 3. Simulation process in POOSL.

The main focus of the HLA model was an understanding of the time synchronisation mechanisms and the exchange of time-stamped messages. In the model, all federates are time-regulating (i.e., they can send time stamp ordered messages) and time constrained (i.e., they can receive time stamp ordered messages). Note that there is no global notion of simulation time, each federate may have a different local time.

Federates should not send messages with a time stamp smaller than their local time plus some lookahead value. They explicitly request the RTI to allow an update of their local time. The RTI will grant such a request if it can guarantee that the federate will never receive messages with a time stamp less than its own local time. The main challenge of the POOSL model was the modelling of the RTI to ensure the required properties. For instance, in a first version a TimeAdvanceRequest (TAR) for time T was granted by a TimeAdvanceGrant (TAG) message if T is less or equal than the local time plus the lookahead of all other federates. This worked well for two federates, but leads to a problem with three federates. The model has been adapted such that for federates that did a TAR, the requested time is used instead of the current local time. This is correct because after a TAR a federate should not send messages with a time stamp less than the requested time plus the lookahead value.

POOSL models can be edited and simulated using an Eclipse-based IDE. During model simulation, the exchanged messages can be inspected by means of running sequence diagram, as depicted in Figure 4. Models can also be simulated without the Eclipse IDE, using the underlying Rotalumis simulator which can be controlled by means of a socket.

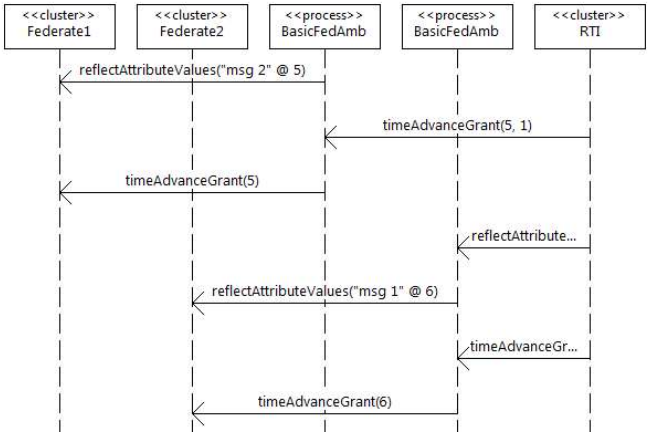

Figure 4. Sequence diagram during POOSL simulation.

III. Incorporating POOSL into HLA

As a next step, experiments with the incorporation of models from different tools in HLA have been conducted. For these experiments we created a model of a simple home heating system, because it is rather straight-forward to validate the results. The home heating system consists of a room with one large window, one heater and a thermostat. These components are described in two separate models: a continuous model and a discrete model.

The continuous model describes the room and the heater it contains. It is created in OpenModelica[5]. The model describes a room with a volume of 98 m^3, a window with an area of 12.25 m^2 and one heater with a maximum power of 550 W.

[5]https://www.openmodelica.org/

We assume that the outside temperature is 5 °C at all times and that there is no energy loss through the walls.

The discrete model is expressed in POOSL and describes the control software running on the thermostat to control the heater state. It switches the heater state when the measured temperature drops below or rises above a given threshold from the target temperature. The control software periodically checks the temperature. The cycle period is stored as a parameter of the model.

Shared attributes between the models are the *temperature*, which is output of the room model and input for the control model, and the *heaterState*, which is set by the control model and used by the room model.

A. Architecture

Since our goal is to co-simulate the models through the HLA, the models should be run in a simulator that supports interactions with an HLA environment. We use the open source poRTIco[6] framework, which provides an RTI implementation for HLA and includes both Java and C++ interfaces for federates. We decided to use the Java version. Figure 5 shows how each of the federates should be connected to the HLA RTI. Note that both the RTI and the HLA interface implementation are shared among the connected federates. We shall briefly explain all components.

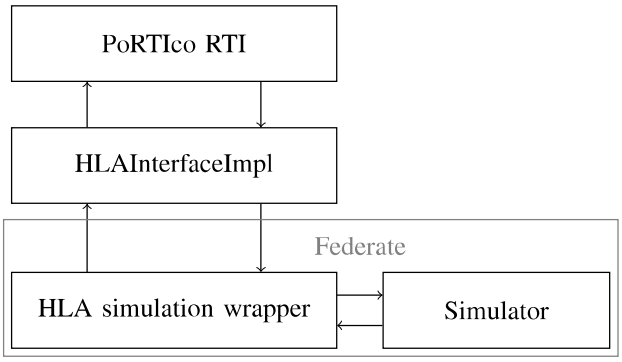

Figure 5. Connection model for HLA federates.

- The *PoRTIco RTI* is provided by the poRTIco project. The RTI routes all communication and controls time.
- *HLAInterfaceImpl* is an implementation of the HLA interface that is provided by poRTIco. The implementation is simulation specific and specifies attributes, events, federates, and connections between them.
- The *HLA simulation wrapper* is the component that facilitates proper communication between the simulator and the HLA implementation. Since most simulators do not support direct connection to an HLA RTI, the wrapper sets up a connection to the simulator and translates calls to and from the HLA.
- The *Simulator* is the simulator that executes the model.

[6]http://www.porticoproject.org/

The implementation of the HLA simulation wrapper requires communication with the simulator to control it. We describe the main ideas for each of the models.

Room model

As mentioned before, the model of the room is a continuous model created in OpenModelica. Since OpenModelica does not support external connections to control the simulation, we cannot develop a wrapper that directly connects to the tool. As discussed in [4], the Modelica model can be exported to a Functional Mock-up Unit (FMU). We exported the model of the room to an FMU using JModelica[7]. To control the simulation from our Java environment, we embedded the Java FMI (JFMI[8]) library in our simulation wrapper. JFMI provides an API to read/write attributes from/to the simulation and to control the simulation time. Our wrapper uses this API to connect the FMU to the HLA implementation.

Control model

For the control software, the HLA simulation wrapper should communicate with the POOSL control model. This can be done by either connecting to the debugging socket of the Rotalumis simulator or to a socket in the POOSL model. Because this connection should communicate time control messages and attribute updates, we have experimented with four connection possibilities.

1) Send all communication through the Rotalumis debugging socket. The major problem of this configuration is that it is very slow.
2) Communicate time control messages through the Rotalumis debugging socket and send attribute updates through a socket in the model. This method, however, causes difficulties in keeping both sockets synchronised.
3) A configuration that uses the embedded socket for time control messages and the Rotalumis debugging socket for attribute updates has the same timing difficulties as the previous point.
4) Use an embedded socket in the POOSL model for both the time control messages and attribute updates. This method is fast, but it requires some modifications to the POOSL model.

Because the last configuration provides the best performance, we use a POOSL process, called HLAproc, to connect a POOSL control model to our HLA simulation wrapper. It is described in the next section.

B. POOSL process HLAproc

The POOSL process HLAproc provides a small set of methods to connect a given POOSL model to the HLA simulation wrapper. HLAproc communicates over a socket connection with the HLA simulation wrapper and communicates with the given POOSL model through a POOSL port. HLAproc can be added to an existing POOSL model rather easily and provides methods for time, attribute and event synchronisation with the HLA RTI. Each of these aspects will be explained briefly.

[7]http://www.jmodelica.org/

[8]http://ptolemy.eecs.berkeley.edu/java/jfmi/

Time control

In a POOSL model, `delay` e statements are used to add a notion of time. To keep time synchronised with the RTI, the model should send a TAR and wait for a TAG to advance time. A TAR can be sent by `hla!timeAdvanceRequest(`e`)`, after which a response `hla?timeAdvanceGrant(`t`)` is awaited. These statements should replace existing `delay` statements.

Attribute control

To let the HLA process read and write arbitrary attributes from and to the POOSL model, these shared attributes should be stored in a publicly accessible object. For this, we use a data structure in POOSL that is very similar to a map structure. This structure provides a `read` and a `write` statement to the POOSL model.

Event control

As HLA also provides methods to broadcast and receive events, our HLA process is also capable of sending and receiving events. For example, an event may be sent using the `hla!interact(`e`)` statement. The HLA process will send this event to the HLA simulation wrapper, which will pass on the event to the HLA RTI.

Even though the model requires some changes to work with our HLA process, it is feasible to build a small application or script that applies these modifications automatically.

C. Experiments

We conducted a number of experiments with our home heating system co-simulation. This includes the impact of the step size and lookahead for the room model on both temperature control and simulation speed. We also discuss the results of different cycle periods for the thermostat to poll the temperature of the room. All experiments were conducted by running the same scenario, in which four hours of simulation time are executed. Initially, the target temperature of the thermostat is set at 18°C. After 45 minutes, the target temperature is increased to 20°C, after which it is lowered again to 18.5°C after two hours. After an hour, the target temperature is lowered again to 18°C.

For the first experiment, we set the cycle period of the control model to 30 seconds. Then the step size of the room is altered: 1, 5, 10, 30 and 60 seconds were simulated. The lookahead is equal to the step size. Figure 6 displays the results of the simulation runs, only displaying the plots of 10 and 60 seconds for readability. From this figure, it is clear that a smaller step size for the room leads to a more accurate functioning of the thermostat, even though the cycle period of the thermostat simulation in untouched. We can explain this with a small example. Assume the thermostat reads the temperature of the room at time $T_{\text{thermostat}}$. HLA federates cannot send messages earlier then their local time plus the lookahead. The thermostat has a lookahead of 1, which means that an update of the heater state can be received by the room simulation at time $T_{\text{update}} = T_{\text{thermostat}} + 1$. Let S be the step size of the room. The RTI delivers the update to the room only if the requested time of the room is equal or higher than the time

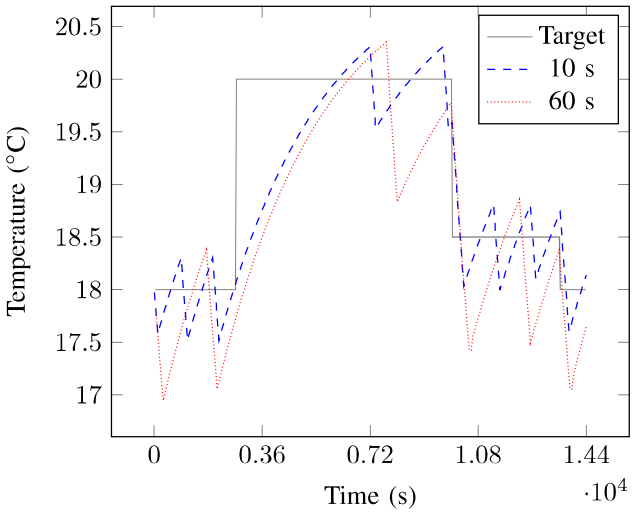

Figure 6. Room temperature with different HLA step size settings for the room.

stamp of the update ($T_{\text{room}} \geq T_{\text{update}}$). When $T_{\text{room}} = T_{\text{thermostat}}$ holds, the room simulation will only receive the update of the heater state on $T_{\text{room}} + S$, assuming $S > 1$. A bigger step size (S) therefore results in a slower response of the room regarding heater state updates, which leads to a less accurate simulation.

To investigate the impact of the step size of the room model on the accuracy and the duration of the simulation, we run a number of simulations with different step sizes. We then calculate the deviation from the target temperature to measure the accuracy. Moreover, we measure the time it takes to complete our simulation and calculate speed-up compared to the slowest simulation with step size 1. The results are shown in Figure 7. The figure shows that the simulation speed

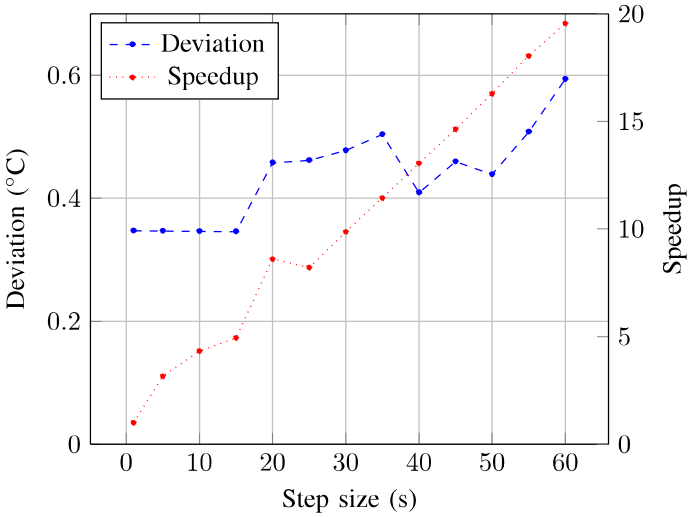

Figure 7. Step size impact on accuracy and simulation speed.

increases linearly with the step size. The results also show that up to a step size of 15 seconds, there is no loss in accuracy. All step sizes larger than 15 seconds result in a less accurate

simulation. Therefore, 15 seconds appears to be a the optimal step size for maintaining simulation accuracy, but improving speed for our simulation.

To check whether our implementation for requesting time grants by the POOSL model is functioning as expected, we ran our simulation with cycle periods of 30, 60 and 120 seconds for the control model and recorded the temperature of the room. The results are very similar to the results displayed in Figure 6, which is as expected.

D. User interaction

Many systems that are controlled by software involve human interaction. E.g., for our home heating system, a user should be able to control the target temperature of the room. To simulate scenarios in which a user may interrupt the system by providing input, we could either add predefined action sequences or let a user provide input through an interface. We applied the first method in our experiments as described in section III-C. This method is typically very useful for performing automatic testing. It may, however, also be useful to have a user interface to provide user input and to observe data of a model during the simulation through a user interface. Hence, for each federate in our system we developed a graphical user interface (GUI).

Each GUI is implemented as a separate federate that receives updates from its simulation federate and provides information to the simulation federate. The GUI federates are both time constrained and regulating, since they both should be capable of receiving and sending time stamped ordered (TSO) messages. Receiving TSO messages is required in order to correctly display all federate information and sending TSO messages ensures that the GUI can send updates at the correct moment in time.

The GUI for the room displays a chart with the temperature and is capable of either starting or stopping the simulation by controlling time. The thermostat GUI displays the measured temperature and target temperature of the room and the state of the heater. Moreover, it can be used to change the target temperature. Figure 8 shows the thermostat GUI. These two GUIs are added to the simulation as federates, leading to an HLA simulation with four federates. The addition of the GUIs does slow down the simulation speed significantly.

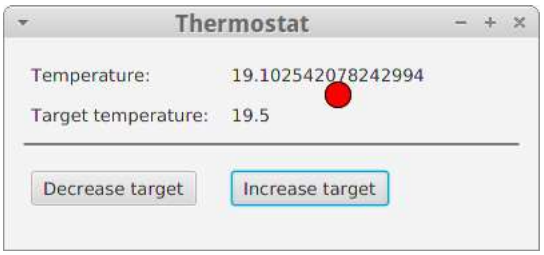

Figure 8. Screenshot of the thermostat GUI.

IV. Conclusion

We have used POOSL to create a model of the HLA framework, which enabled us to quickly understand the mechanisms in the HLA. We also developed a method to co-simulate a discrete-time model of control software with a continuous-time model of the physical system. Here, we found that time control for these models should be approached differently. The software controller initiates time advance requests by itself, whereas the continuous model does not. Therefore, actual time control for the continuous model was moved to the simulation wrapper.

In the near future, we will apply this method to co-simulate POOSL models of the software architecture of an industrial application with the models of its hardware. To investigate scalability of the approach for industrial applications, we will also experiment with distributed simulations.

References

[1] IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Framework and Rules. *IEEE Std 1516-2010*, pages 1–38, 2010.

[2] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, et al. The functional mockup interface for tool independent exchange of simulation models. In *8th Modelica Conference*, pages 105–114, 2011.

[3] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity - the Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003.

[4] A. Elsheikh, M. U. Awais, E. Widl, and P. Palensky. Modelica-enabled rapid prototyping of cyber-physical energy systems via the functional mockup interface. In *Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES 2013)*, pages 1–6. IEEE, 2013.

[5] J. Fitzgerald, P. Larsen, K. Pierce, and M. Verhoef. A formal approach to collaborative modelling and co-simulation for embedded systems. *Mathematical Structures in Computer Science*, 23:726–750, 8 2013.

[6] H. Georg, S. C. Müller, C. Rehtanz, and C. Wietfeld. Analyzing cyber-physical energy systems:the INSPIRE cosimulation of power and ICT systems using HLA. *IEEE Transactions on Industrial Informatics*, 10(4):2364–2373, 2014.

[7] J. Hooman, N. Mulyar, and L. Posta. Coupling simulink and uml models. In *Proceedings of Symposium FORMS/FORMATS*, pages 304–311, 2004.

[8] G. Karsai and J. Sztipanovits. *Model-Integrated Development of Cyber-Physical Systems*, pages 46–54. Springer Berlin Heidelberg, 2008.

[9] G. Lasnier, J. Cardoso, P. Siron, C. Pagetti, and P. Derler. Distributed simulation of heterogeneous and real-time systems. In *17th Symposium on Distributed Simulation and Real Time Applications*, pages 55–62. IEEE Computer Society, 2013.

[10] B. D. Theelen, O. Florescu, M. C. W. Geilen, J. Huang, P. H. A. van der Putten, and J. P. M. Voeten. Software/Hardware Engineering with the Parallel Object-Oriented Specification Language. In *5th Conference on Formal Methods and Models for Codesign*, MEMOCODE '07, pages 139–148. IEEE Computer Society, 2007.