2018 21st Euromicro Conference on Digital System Design

Building Distributed Co-simulations using CoHLA

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

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

Jack Sleuters
ESI (TNO)
Eindhoven
The Netherlands
jack.sleuters@tno.nl

***Abstract*—The construction of a co-simulation for large cyber-physical systems can be very time consuming. We have defined a domain specific language called CoHLA that facilitates this construction based on the standards FMI and HLA. Scalability of this approach is investigated by the application to Internet of Things (IoT) systems. Because of the repetitive nature of these systems, we developed a separate domain specific language that allows the user to describe the system and easily generate a co-simulation definition for CoHLA. Additionally, we extended CoHLA to speed up the co-simulation execution by distributing the simulation across multiple nodes. This method also allows the co-simulation to be executed in the cloud easily.**

***Index Terms*—Domain Specific Language, Cyber-physical systems, Co-simulation, Distribution, Internet of Things, HLA, FMI**

I. Introduction

The design of cyber-physical systems (CPSs) is a complex process. It involves the collaboration of multiple different disciplines to design a single system. Each of these disciplines has its own development methods and modelling tools, which makes it hard to verify the behaviour of the entire system during the development. To verify this behaviour in an early stage of the development process, co-simulation standards such as the High-Level Architecture (HLA) [1] and the Functional Mock-up Interface (FMI) [4] were developed. HLA is an interface specification to construct a co-simulation of models. FMI provides an interface for the simulation of a model and allows for easy model exchange between tools.

Since it requires quite some effort to construct an HLA co-simulation from a set of simulation models, CoHLA [10] was developed. CoHLA is a Domain Specific Language (DSL) that allows the user to easily specify the simulation models that are part of the co-simulation and how those models are connected to each other. From this specification, configuration files and code are being generated to be executed with OpenRTI[1], which is an open source implementation of the HLA standard. CoHLA supports simulation models that are compliant to the FMI standard, thus supporting a wide variety of modelling tools. Additionally, POOSL [12] models are supported to allow a convenient definition of discrete-time models of software and use them in the co-simulation.

As systems become more and more complex, co-simulations of these systems also become more complex. Consequently, both the construction and execution of the co-simulation become more time consuming.

In particular, this is the case for Internet of Things (IoT) systems [13]. IoT systems generally consist of a large number of connected components, such as sensors and actuators. Examples are a self-regulating climate control system and a smart lighting system for a whole building. These systems consist of large number of interconnected sensors and actuators, which makes them heavy to simulate. The size also affects the effort required to construct the co-simulation.

A. Approach

To improve the scalability in terms of performance for large co-simulations, we extended CoHLA with a method to easily distribute federates across multiple computation nodes. Since OpenRTI already provides support for distributed co-simulations, our focus is on extending CoHLA to make this easy to use. This will also provide a method to easily distribute a co-simulation across systems in the cloud.

To construct a co-simulation in HLA, the user should define all simulation instances and how those are connected to each other. Even though CoHLA reduces this effort, it is still quite some repetitive labour for very large systems such as IoT systems. To reduce the construction effort required for such systems, we present a DSL-based approach to generate CoHLA specifications. Next, working co-simulations can be generated from these CoHLA specifications.

A case study of a smart lighting system is used to illustrate both the DSL-based approach and distributed co-simulation.

B. Related work

Based on a CoHLA specification, we generate a co-simulation where HLA's RTI is the master for a number of simulations that conform to the FMI interface. In addition, for discrete components we support POOSL simulations (see Section III-C). The combination of HLA and FMI was already proposed in [2]. A mechanism to develop an HLA-compliant federate using a wrapper around an FMI component is described in [14]. An approach which uses wrappers to integrate multiple FMI components into one HLA co-simulation has been demonstrated in [11]. Nevertheless, in these approaches the definition of a co-simulation still requires quite some effort and the aim of CoHLA is to make this much easier.

[1] https://sourceforge.net/projects/openrti/

978-1-5386-7377-5/18/$31.00 ©2018 IEEE
DOI 10.1109/DSD.2018.00066

A recent paper of Falcone et al. [8] confirms our motivation to facilitate the definition of HLA-based simulations: "the development of distributed simulations based on the HLA standards remains a challenging task that requires a considerable effort in terms of time, cost and expertise". To ease the development of such simulations, they developed the HLA Development Kit framework (DKF), a general-purpose software framework on top of available HLA implementations.

Interesting experimental results about the use of HLA can be found in [5], which describes five different scenarios. Related to our smart lighting case is a wireless sensor network application with many distributed sensors and moving people. Here, distributed simulation was essential to obtain results.

To support HLA-based simulations on a cloud architecture, [9] presents an approach for the automatic management of the underlying resources. This saves time of the modellers and avoids that they need detailed knowledge of distributed execution. In our work we focus on an easy-to-use DSL and have used a rather straightforward approach to distribute simulation on the nodes in the cloud. More advanced strategies can be found in the literature. In, for instance, [15] a cloud simulation platform has been proposed for the efficient execution of an HLA simulation on virtual machines in the cloud.

This paper is structured as follows. The lighting case study is described in Section II. Section III provides a brief overview of the technologies used, including a few basic examples of CoHLA. The new CoHLA extensions are presented in Section IV. The DSL for the lighting system is described in Section V. Results are discussed in Section VI and concluding remarks can be found in Section VII.

II. Case study: Smart lighting system

To illustrate the issues encountered when creating a co-simulation of large CPSs, a smart indoor lighting system will be used as case study. A similar lighting system was used for perform robustness analysis by Doornbos et al. [6]. The lighting system includes occupancy sensors, lights and controllers within a building. Based on the sensor information, specific lights should be switched on or off automatically or they should be dimmed to a specific brightness level.

Using co-simulation to simulate such a lighting system gives insight in the structure and working of the system as a whole. It also provides a method to analyse performance characteristics such as energy usage. Analysing the behaviour of a lighting system for different scenarios, sensors or configurations is also important during system design because changes are difficult when the system has been installed in a building.

A. Lighting schema

Light controllers of offices receive input from one or more occupancy sensors and turn on connected lights when activity is detected. A building may consist of different types of offices, each of them having their own lighting behaviour. Offices are connected by corridors, which also contain a number of sensors and lights and may behave differently than offices. For example, lights in a corridor may not be turned off when there is still activity being detected in one of the bordering areas.

B. Models

To be able to co-simulate a lighting system, occupancy sensor models, light models and controller models were created. Occupancy sensors receive the coordinates of persons walking inside the building as input attributes and should therefore also know their own location. To prevent the sensors from detecting activity through walls, the bounds of the area (office or corridor) in which they are located are also provided. Light and sensor models are created in 20-sim and exported to FMU.

A separate controller model is created in POOSL for every type of area. These models implement the timers and control logic for the lights, based on input from the connected occupancy sensors.

C. Scalability

Assuming that an average office contains three sensors, three lights and one controller, the average office consists of seven model simulations. The co-simulation of a building therefore quickly becomes rather large. Since all components in all areas should be instantiated and properly connected to each other, the construction of such a co-simulation is very time-consuming and error prone. Additionally, having a large system increases computation time as well.

III. Background

A. Functional Mock-up Interface (FMI)

The Functional Mock-up Interface (FMI) [4] is a standard that aims for model exchange and co-simulation. Models exported to so-called Functional Mock-up Units (FMUs) comply to the FMI interface that allows them to be imported or simulated in other modelling or simulation tools. Binaries of the models may also be included to support simulation of the model while retaining the confidentiality of the sources. The FMI standard is widely supported by modelling tools.

B. High-Level Architecture (HLA)

The High-Level Architecture (HLA) [1] is an interface specification for co-simulation of separate simulations. In HLA, a co-simulation is called a *federation*, which consists of a number of simulations called *federates*. HLA provides a set of rules that all federates in a federation must comply to. A Run-Time Infrastructure (RTI) is responsible for the orchestration and compliance – including synchronisation of attributes and time – of the co-simulation. Both commercial and open source implementations of HLA are available.

C. Parallel Object-Oriented Specification Language (POOSL)

The Parallel Object-Oriented Specification Language (POOSL) [12] is a modelling language that is used to create discrete-time models of systems. The language is very similar to a regular programming language which makes it useful for modelling software architectures and comprehensible by software architects. POOSL models can be simulated using the Rotalumis simulator[2].

[2] http://www.es.ele.tue.nl/poosl/Tools/rotalumis/

D. Configuring HLA (CoHLA)

Configuring HLA (CoHLA[3]) [10] is a Domain Specific Language (DSL) to construct an HLA co-simulation. Since this requires every simulation model to comply to the HLA interface, the construction of a co-simulation can be very time-consuming. This is in particular the case when the simulation models or the co-simulation itself change frequently or when the co-simulation is very large. CoHLA was developed to enable rapid construction and modification of co-simulations. From a co-simulation specification in CoHLA, code is generated to be simulated using OpenRTI as RTI.

A CoHLA specification consists of a federate class description for each of the simulation models in the co-simulation. This description specifies all input and output attributes of the model as well as some HLA specific configuration. Listing 1 shows a CoHLA definition of a federate class for an FMU model of a basic light controller. The controller receives an *occupied* flag as an input and outputs a value *setpoint*.

```
FederateClass BasicController {
  Attributes {
    Input Boolean occupied
    Output Real setpoint
  }
  SimulatorType FMU
}
```

Listing 1. CoHLA definition for a light controller model.

To construct the HLA federation, each federate is defined as an instance of one of the federate classes. From this specification, wrapper code for each of the federate classes and a set of configuration files is generated. Listing 2 shows a basic smart lighting co-simulation defined in CoHLA. The co-simulation consists of one controller, sensor and light.

```
Confederation SampleFederation {
 Instances {
  Instance sense as Sensor
  Instance light as DimmableLight
  Instance ctrl as Controller
 }
 Connections {
  Connection { light.setpoint <- ctrl.setpoint }
  Connection { ctrl.occupied <- sense.activity }
 }
}
```

Listing 2. CoHLA definition of a federation for a lighting system.

CoHLA provides support for generating wrapper code for models exported as FMU and POOSL models. It also provides support for extensions that allow logging, collection of basic performance metrics and Design Space Exploration (DSE).

IV. Extending CoHLA

To tackle the scalability issue in terms of simulation time needed for large systems, CoHLA was extended to ease the specification of distributed co-simulation as supported by OpenRTI. This section briefly outlines the changes made to CoHLA and the implementation decisions taken.

A. Distribution architecture

OpenRTI supports two methods of creating a distributed co-simulation. The first method is to start a single RTI on one node and connect all federates to this RTI. The second method supported by OpenRTI is to start multiple RTIs and connect them to each other. One of the RTIs acts as parent RTI, while the other RTIs are connected as child RTIs. An advantage of this method is that the communication between the two RTIs is bundled and compressed, resulting in less communication over the network interface. Several test runs using both methods confirm that the parent-child architecture is a little faster.

B. Distribution implementation

CoHLA generates a run-script for each co-simulation that allows the user to easily start and configure the co-simulation. To provide an easy method to start the co-simulation in a distributed manner, the generated run-script is extended to support this. To distribute all simulations across a number of nodes, two distribution methods are implemented.

The first method allows the user to assign a weight to every federate class to represent the simulation complexity of the model. The run-script determines which federates to start based on their weights. No further configuration is required.

The second method allows the user to specify the distribution of federates over a predefined number of computation nodes. Listing 3 displays a sample distribution configuration. From this specification, a configuration file is generated that can be provided to the run-script.

```
Distribution dist3 over 3 systems {
  System 0: fed1 fed3 fed4
  System 1: fed2 fed5 fed6
  System 2: fed7 fed8
}
```

Listing 3. Manual co-simulation distribution specification.

Both methods for distributing the simulations of a co-simulation require the user to specify a node identifier for each of the nodes when starting the co-simulation to enable the run-script to start the correct federates.

V. Lighting DSL

With distributed simulation supported by CoHLA, the scalability issue regarding simulation performance for the co-simulation of large systems has been addressed. The construction of a CoHLA system specification for an IoT application, however, still requires quite some labour. As the lighting system consists of a large number of sensors and lights, a CoHLA specification for the system consists of many federate instances and many more attribute connections to connect them all together. Manual specification of such a system is very error prone and difficult to debug. To simplify the definition of such a CoHLA specification, a DSL-based approach was used.

A separate DSL[4] was developed using the Xtext and Xtend frameworks [7][3] in Eclipse[5]. The DSL allows the user to model a building by specifying its rooms, corridors, sensors and lights. A CoHLA co-simulation definition is generated from such a building specification, which can be used to generate code for the co-simulation.

[3] https://github.com/phpnerd/CoHLA

[4] https://github.com/phpnerd/Lighting-DSL

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

A. Buildings

Because it is assumed that all simulation models have already been specified in CoHLA, the lighting DSL only generates a federation specification. The DSL supports a number of types of building components that can be used, each of them corresponding to a simulation model specified as federate class in CoHLA. File import statements for the correct federate classes together with a federation definition consisting of federate instance definitions and their connections are generated. This includes both connections from every sensor and light to the controller of the area in which they are located as well as connections to their bordering corridor controller.

Every sensor has a vision radius, which is bounded by the walls of the room. Therefore, corner coordinates for each of the areas in a building as well as the location of every sensor must be specified. From this information, a model parameter configuration file is generated that bounds the sensors ranges upon initialisation. Additionally, the information is also used to generate an image of the building.

B. Scenarios

The co-simulation of a smart lighting system is capable of providing some insights in the behaviour of the lights. However, the co-simulation itself is not that interesting when no (virtual) person interacts with the system simulation. For this purpose, CoHLA supports the definition of scenarios that apply predefined changes of attribute values during the simulation. The lighting DSL provides an easy method to add an actor that walks a specified path through the building. From this scenario definition, a CoHLA scenario file is generated.

C. Building simulation verification

Although the image of the building as generated by the lighting DSL enables the user to verify static properties such as sensor location and area size, it does not provide a way to verify the behaviour of the building given a scenario. CoHLA provides support for adding a logging federate to verify this behaviour after running a co-simulation. A log contains a large table with the states of all sensors and lights during the simulation. Such a log is difficult to read; plotting a graph of the data increases readability, but still is hard to read when the number of lights and sensors is high. For this, the lighting DSL generates a web page that allows the user to import a log file and replay the simulation while showing the state of all building components as an image. This provides more intuitive insight in the behaviour of the lighting system.

D. Distributions

As described in Section IV-B, CoHLA supports two methods for distributing federates across a number of nodes. The first method does not require any configuration beforehand and therefore does not require support from the lighting DSL. The second method, however, requires a manually defined distribution of the federates. For the smart lighting system there is a rather intuitive distribution configuration: all federates forming a single area – office or corridor – will run on the same node. In this way, components that communicate with each other frequently are simulated on the same node, allowing optimisation of network traffic. The lighting DSL automatically generates CoHLA distribution configurations following this distribution method.

VI. Results

A. The system

To verify our approach for the co-simulation of IoT systems using CoHLA, a sample building that will function as a test case was created using the lighting DSL. The building consists of eleven offices and three corridors connecting them. Figure 1 shows the rooms, corridors, sensors and lights in the building. The building has 36 lights, 38 occupancy sensors and 14 light controllers. The resulting co-simulation also contains a logger federate and an actor that walks through the building according to a provided scenario. A total of 90 federates is connected to the RTI.

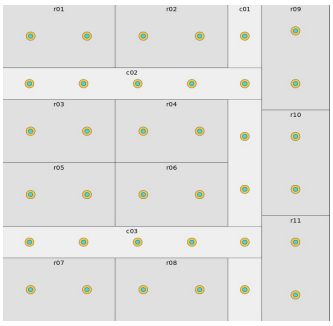

Figure 1. Our test case building.

B. Definition of the system

The use of the lighting DSL for the definition of the building is rather straightforward. Constructing a co-simulation of a building by specifying its components is more convenient than building the co-simulation using only CoHLA. In particular, it is convenient that the user does not have to worry about connections and initialisation configurations, which saves time. The required amount of code confirms this: our building specification is 239 lines of code long in the lighting DSL, which generates a CoHLA specification of 1659 lines of code. The lighting DSL is particularly useful when changes are applied to the building frequently.

C. Distribution

For the sample building, a scenario was created that mimics the behaviour of an employee in the office in the morning. The employee enters the building, walks to other offices, fetches coffee a couple of times and leaves after three hours. When the employee has left the building the simulation stops.

To measure the effects of distributing the simulation, the scenario described above was simulated on a number of virtual systems (nodes) in the cloud. Every node has 8 dedicated virtual CPUs and 16 GB of memory and an internal network connection to the other nodes. The simulation speed is measured on a single node, after which the co-simulation is distributed over two to seven nodes. Both the automatic weight-based distribution by CoHLA and the grouped distribution generated by the lighting DSL were measured and compared to each other. The speed is compared to the average single-node simulation speed of the nodes participating in

the co-simulation, which ranged from 1666 to 2253 seconds, depending on the node. For every co-simulation execution, the log file was checked to verify that the simulation was correct.

Figure 2 shows the speed-up compared to the average simulation time of a single node for both distribution methods. From this figure it can be concluded that grouping federates manually based on their connections slightly improves the performance of the distributed simulation execution.

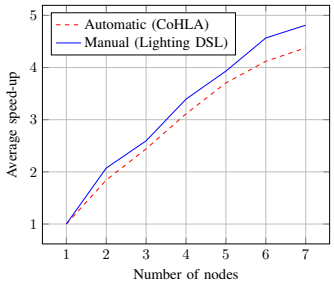

Figure 2. Speed-up per distribution method.

VII. Conclusion

We presented two approaches to address some of the issues that arise when co-simulating IoT systems using CoHLA. A DSL-based approach was used to simplify the construction of a co-simulation of large systems. Using a DSL that is specifically designed for a particular class of systems simplifies the construction of a co-simulation of such a system by generating a CoHLA definition. Although the development of such a DSL requires one or two days of development time using Eclipse with Xtext and Xtend, it reduces the time required to specify a CoHLA co-simulation from a couple of hours to about 15 minutes for our case study. Depending on the number of co-simulations to construct and their sizes, the development of a separate DSL can be worth the development time. This method is also more intuitive to use and is less error prone. For systems that contain many repetitive sets of simulation models, development of a DSL for generating CoHLA specification could improve the development speed. Since this requirement is often met by IoT systems, this approach can be very beneficial for the design such systems.

The second approach aims for increasing of the simulation speed of a CoHLA co-simulation. Two methods for distributing the simulations across a number of connected nodes were presented. The first distribution method supports weight-based distribution by CoHLA where the user may provide weights but does not need to specify the distribution of federates over nodes. The second method requires a manually provided distribution specification in CoHLA. A separate DSL for generating the CoHLA configuration can also generate such a predefined distribution. After testing both methods on a sample system co-simulation, both distribution methods were found to scale well. Distribution of the simulations in a large co-simulation across multiple nodes using CoHLA therefore is a good method to improve the simulation speed.

The approach also allows the use of nodes in the cloud. The advantage of cloud-based nodes is that they are relatively cheap and flexible to use. Using the cloud for co-simulation also greatly increases the scalability of the processing power available for co-simulation and does not require the user to buy systems for running large co-simulations.

In future work, we will improve CoHLA for robustness testing by adding fault-injection. This enables robustness testing by co-simulation of a system. Additionally, functionality for event-based communication will be added to CoHLA. Intuitively, the federates in the smart lighting system use message-based communication. However, CoHLA is currently focused on attribute synchronisation instead of event-triggered communication, even though HLA does support this. Extending CoHLA with event-based communication will further improve the usability of CoHLA for IoT applications.

References

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

[2] M. U. Awais, P. Palensky, A. Elsheikh, E. Widl, and M. Stifter. The high level architecture RTI as a master to the Functional Mock-up Interface components. In *2013 International Conference on Computing, Networking and Communications (ICNC)*, pages 315–320. IEEE, 2013.

[3] L. Bettini. *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing Ltd, 2016.

[4] T. Blochwitz, M. Otter, et al. The Functional Mockup Interface for tool independent exchange of simulation models. In *8th Modelica Conference*, pages 105–114, 2011.

[5] A. V. Brito, H. Bucher, H. Oliveira, L. F. S. Costa, O. Sander, E. U. Melcher, and J. Becker. A distributed simulation platform using HLA for complex embedded systems design. In *Distributed Simulation and Real Time Applications (DS-RT), 2015 IEEE/ACM 19th International Symposium on*, pages 195–202. IEEE, 2015.

[6] R. Doornbos, J. Verriet, and M. Verberkt. Robustness analysis for indoor lighting systems. In *10th International Conference on Systems, Barcelona, Spain*, 2015.

[7] M. Eysholdt and H. Behrens. Xtext: implement your language faster than the quick and dirty way. In *Proceedings of the ACM international conference companion on object oriented programming systems languages and applications companion*, pages 307–309. ACM, 2010.

[8] A. Falcone, A. Garro, S. J. Taylor, and A. Anagnostou. Simplifying the development of HLA-based distributed simulations with the HLA Development Kit software framework (DKF). In *Proceedings of the 21st International Symposium on Distributed Simulation and Real Time Applications*, pages 216–217. IEEE Press, 2017.

[9] S. Guan, R. E. De Grande, and A. Boukerche. Enabling HLA-based Simulations on the Cloud. In *Proceedings of the 19th International Symposium on Distributed Simulation and Real Time Applications*, pages 112–119. IEEE Press, 2015.

[10] T. Nägele, J. Hooman, T. Broenink, and J. Broenink. CoHLA: Design Space Exploration and Co-simulation Made Easy. In *IEEE 1st Industrial Cyber-Physical Systems (ICPS 2018)*, pages 225–331, May 2018.

[11] H. Neema, J. Gohl, et al. Model-Based Integration Platform for FMI Co-Simulation and Heterogeneous Simulations of Cyber-Physical Systems. In *Proceedings of the 10th International Modelica Conference*, number 96, pages 235–245. Linkping University Electronic Press; Linkpings universitet, 2014.

[12] B. D. Theelen, O. Florescu, et al. 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.

[13] A. Whitmore, A. Agarwal, and L. Da Xu. The Internet of Things–A survey of topics and trends. *Information Systems Frontiers*, 17(2):261–274, 2015.

[14] F. Yilmaz, U. Durak, K. Taylan, and H. Oğuztüzün. Adapting Functional Mockup Units for HLA-compliant distributed simulation. In *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*, number 096, pages 247–257. Linköping University Electronic Press, 2014.

[15] S. Zhang, Z. Tang, X. Song, Z. Ren, and H. Meng. Virtual Machine Task Allocation for HLA Simulation System on Cloud Simulation Platform. In *AsiaSim 2012*, pages 395–403. Springer, 2012.