Scalability analysis of cloud-based distributed simulations of IoT systems using HLA

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

***Abstract*—Gaining insight in the properties of an Internet of Things (IoT) system during the design phase is difficult. The co-simulation of such a system would be very useful, but creating it is usually time consuming. By means of domain specific languages (DSLs) we support the fast construction of large co-simulations of IoT systems. This approach includes the use of CoHLA, a DSL that generates co-simulation code based on the HLA and FMI standards. Due to the large number of connected sensors and actuators in an IoT system, the time needed for simulation can be a blocking factor. Hence we facilitate distributed co-simulation in the cloud. To do that efficiently, we have conducted a set of experiments to analyse scalability and the performance impact of distribution methods. From these experiments, lessons were learned on how to distribute the co-simulation of IoT systems.**

***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 multidisciplinary process. Every discipline has its own development tools and modelling standards. Consequently, it is hard to find small errors during the design of the whole system based on these individual models. To get early insight in the collaboration of all components during the design process, it is useful to simulate them together. Standards such as the High Level Architecture (HLA) [1] and the Functional Mockup Interface (FMI) [3] were developed to build co-simulations of different types of models. HLA is an interface standard that specifies a set of rules to construct a co-simulation, while the FMI standard specifies an interface for the simulation model.

Even though these standards exist to simplify the construction of a co-simulation, it still requires quite some development time to implement a working co-simulation. To reduce this effort, CoHLA[1] [12] was developed. CoHLA is an open source domain specific language (DSL) that allows the user to quickly describe a co-simulation by specifying the simulation models in terms of input and output attributes and by defining how these models are connected. From this description, co-simulation code for OpenRTI[2] is generated. OpenRTI is an open source C++ implementation of the HLA standard. CoHLA provides support for Functional Mockup Units (FMUs) – simulation models compliant to the FMI standard – and POOSL [16] models.

[1]`https://github.com/phpnerd/CoHLA`
[2]`https://sourceforge.net/projects/openrti/`

However, the co-simulation of large systems introduces scalability challenges. Internet of Things (IoT) systems are an example of such systems, as they usually consist of a large number of interconnected sensors and actuators. This introduces scalability challenges for both the specification of an IoT system as well as the simulation speed.

To improve to simulation speed of large co-simulations, the HLA standard allows distributed simulation execution. The HLA implementation that is used determines whether and how distribution is possible. Our goal is to find out how to effectively distribute co-simulations with HLA and to ease the use of such distribution methods using CoHLA.

A. Approach

To measure the performance impact of running a distributed co-simulation, we will run simulations of a large IoT system. To speed up the creation of such a co-simulation, the co-simulation will be specified using CoHLA. CoHLA will then generate the code for the co-simulation. As CoHLA generates code for use with OpenRTI, this implementation will be used.

Different sizes of co-simulations will be used to measure the performance impact of running the simulation in a distributed manner. The number of computation nodes is changed to determine the scalability of the distribution. These measurements should provide insight in some of the performance characteristics of distributing a co-simulation using HLA.

Another goal is to find out how the HLA standard exactly helps in running a distributed co-simulation. Some experiments are conducted to improve the performance of the co-simulation by optimising the HLA configuration.

Finally, we will discuss how CoHLA can be used to support the use of distributed co-simulations.

B. Related work

The INTO-CPS tool chain [10] supports the co-simulation of FMUs, where VDM [9] is used for discrete event models. Instead of exploiting HLA, they developed a Co-simulation Orchestration Engine (COE) to manage the co-simulation of multiple simulation models. The combination of HLA and FMUs has already been proposed in [2] and an approach to

978-1-5386-7308-9/18/$31.00 ©2018 IEEE
DOI 10.1109/ICPADS.2018.00146

integrate multiple FMUs into one HLA co-simulation using wrappers can be found in [14]. Our work on CoHLA [12] is intended to make the combination of these standards easy to use and to enable a fast construction of such co-simulations. This allows, for instance, convenient design space exploration.

A combination of Ptolemy [15] and HLA is described in [5]. They observed that for a wireless sensor network application is was not possible to simulate a desired scenario with 4000 actors on a single processor. With 8 machines a speedup factor 4 has been achieved. Additional experimental results about the use of HLA can be found in [4] which describes five different scenarios. PERFOSIM [6] aims for performance predictions by simulating an HLA distributed simulation. Experiments with a relatively small number of federates (up to 30) are reported.

A cloud simulation platform has been proposed in [17] for the execution of an HLA simulation on virtual machines in the cloud. It uses a heuristic search algorithm for the allocation of simulations to execution nodes. Also the decomposition of simulations into subtasks is considered, while we consider simulations as a black box and only address their external interface. Another scheme to execute HLA-based simulations in the cloud has been proposed in [8]. It includes automatic management of the underlying resources, saving time of the modellers. Experiments with synthetic load address the comparison with grid-based computing, showing that cloud computing achieves similar performance but has advantages in terms of energy consumption and security. Our work concentrates more on HLA-specific aspects and the distribution of IoT applications.

This paper is structured as follows. Section II describes the type of smart lighting system that will be used as a case study. Background information is provided by Section III, after which Section IV outlines the implementation details of the experiments. Section V describes the experiments that were conducted together with the results, after which Section VI concludes the paper.

II. Case study: Smart lighting system

The smart lighting system is a sample case that is used to illustrate how distributed co-simulation can improve the simulation speed of the co-simulation of an IoT system. The smart lighting system is based on a lighting system that was introduced in [7]. The system consists of a number of occupancy sensors, lights and controllers. Controllers receive input from the sensors and control the state of connected lights.

Being able to easily co-simulate such a lighting system enables the designer to check whether the designed system behaves correctly, meaning that lights turn on and off at the expected moments. By using a co-simulation, different scenarios and settings can be explored. Once the lighting system is installed on location, it is difficult to make changes to the system or to find an error. It is therefore good practice to simulate the system beforehand. Running a co-simulation of a lighting system also provides means for measuring and optimising energy efficiency of the system.

A. *Lighting schema*

Every room in the building has a controller that is connected to all sensors and lights in the room and turns the lights on or off based on the status of the sensors. Because buildings can have different types of rooms, there are different types of controllers with different behaviour. Rooms are connected by corridors, which also have a controller that is connected to all lights and sensors located in the corridor. Additionally, corridor controllers are connected to the sensors in bordering rooms to ensure that corridor lights are not turned off when there is still some activity detected in one of those rooms.

B. *Models*

For every component in the lighting system a model was created. Continuous-time models of a light and an occupancy sensor are created with the modelling tool 20-sim[3] and exported to FMUs. For each type of area – room, office or corridor – a discrete-time controller model is created in POOSL.

III. Background

This section briefly outlines the used technologies.

A. *Functional Mock-up Interface (FMI)*

The Functional Mock-up Interface (FMI) [3] is a standard for model exchange and co-simulation. The standard describes an interface that allows tools to interact with the model by reading and writing attribute values and controlling simulation time. Modelling tools adhering the standard allow for model exportation to a Functional Mock-up Unit (FMU). An FMU contains a standardised XML description of the model, together with the model itself. FMI is widely supported by modelling tools. Although an FMU can be used for running a co-simulation of models, it still requires some external master algorithm to synchronise and execute all separate model simulations.

B. *High-Level Architecture (HLA)*

The High-Level Architecture (HLA) [1] is a standard interface specification for co-simulations. HLA aims for co-simulation of models from different modelling tools by specifying an interface description and a set of rules to ensure proper co-simulation. In HLA, the Run-Time Infrastructure (RTI) is the central component that handles time and attribute synchronisation between all individual simulations. Each of these simulations joins the co-simulation as a federate, while the co-simulation itself is called a federation. Both commercial and public implementations of the HLA standard are available.

C. *Parallel Object-Oriented Specification Language (POOSL)*

The Parallel Object-Oriented Specification Language (POOSL) [16] is a modelling language for creating discrete-time models of systems. Because the language is similar to a regular programming language, POOSL is very suitable to create models of software designs and software architectures. POOSL models can be simulated by the Rotalumis simulator[4].

[3]http://www.20sim.com/
[4]http://www.es.ele.tue.nl/poosl/Tools/rotalumis/

D. *Configuring HLA (CoHLA)*

Configuring HLA (CoHLA[1]) [12] is a domain specific language that was developed to simplify the implementation of a co-simulation in HLA. To manually create a co-simulation in HLA, every simulation model should comply to the HLA interface, which usually requires some wrapper to be developed. Additionally, every type of simulation model should be described as a federate class, which specifies its attributes, sharing type and timing behaviour. The step from having a set of models that should be co-simulated together to a running co-simulation is therefore quite time consuming.

CoHLA allows the user to specify all simulation models as federate classes and generates wrapper code for the classes. Listing 1 shows a sample federate class description in CoHLA. A co-simulation can be specified by defining instances of these classes and connecting their attributes to each other. Listing 2 shows how a co-simulation can be constructed in CoHLA. From this, co-simulation sources, some configuration files and a run script are generated. The run script can be used to build all executables and start the co-simulation.

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

Listing 1. CoHLA definition for a light controller federate class.

```
Import "RoomController.hla"

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

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

CoHLA can generate wrapper code for models exported to an FMU and POOSL models. Additionally, CoHLA provides some extra features such as logging, measuring performance metrics and Design Space Exploration (DSE).

E. *Lighting DSL (LDSL)*

Due to the high number of federates in a large IoT system, the construction of a co-simulation of such a system is – even in CoHLA – very time consuming and error prone. The Lighting DSL (LDSL[5]) [13] is a domain specific language developed to simplify the creation of CoHLA specification of lighting systems as described in Section II. The LDSL shows how a DSL can be used to decrease the effort required to create a co-simulation specification in CoHLA for large systems. Users can design a building by specifying rooms and corridors using coordinates and add lights and sensors to these areas in an intuitive manner. From this building description, a CoHLA definition of the co-simulation of the building is generated, including all federate connections, configurations and predefined scenarios. Listing 3 shows a very simple building in the LDSL, consisting of only one room with one sensor and one light. In addition to the CoHLA definition of the co-simulation of the building, also an image of the building and an interactive viewer for log files are generated.

```
Building VerySmallBuilding {
  Room room1 {
    Draw: (0,0) (300,0) (300,300) (0,300)
    Devices {
      Light l1 on (150,150)
      Sensor s1 on (150,150)
    }
  }
}
```

Listing 3. A sample building as specified in the lighting DSL.

IV. Implementation

This section briefly describes how the experiments from Section V are created and executed.

A. *The system*

To measure the scalability and optimise the configuration for distributed co-simulation in HLA, a number of experiments is conducted. The co-simulations that function as experiments are all co-simulations of lighting systems as described in Section II. They are all defined using CoHLA. This CoHLA definition of the lighting system is generated using the LDSL. From the CoHLA definition of the co-simulation, code is generated that is compiled and executed using OpenRTI.

Although all experiments are conducted on a lighting system co-simulation, each of the experiments may use a different configuration of the system. For each experiment, the system configuration that is co-simulated is described.

B. *Actors*

A smart lighting system only shows interesting behaviour if the occupancy sensors are triggered. Hence, an actor federate is added to each of the co-simulations to trigger occupancy events at predefined moments during the simulation. These events are defined using the LDSL, from which a CoHLA scenario is generated.

C. *Experiment execution*

All experiments are conducted on a set of cloud nodes. These nodes are virtual systems (Droplets) rented at Digital Ocean[6]. Every node has 8 vCPUs and 16 GB of memory and runs on Ubuntu 18.04. Due to the fact that we cannot control the physical hardware the nodes are running on, there may be small variations in the nodes. The nodes are optimised for compute-intensive tasks and connected with each other via a 40 GbE internal network. Experiments are executed in an automated manner. For this, a Python script was developed to automatically connect to all nodes using SSH. The script automatically installs all dependencies and executes the specified set of benchmarks.

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

[6]https://www.digitalocean.com/

V. Experiments

This section introduces the experiments that have been conducted and discusses the results.

A. Distribution methods

CoHLA provides two methods for distributing a co-simulation. The first method is an automated one that is included in the generated run-script, which is used to start the co-simulation. Every federate class is assigned a weight, which represents the required computational power for running the simulation. The run script then distributes all federate instances over the available nodes, based on the the weight of the instances. The resulting distribution is a rather straight-forward *weight-based* distribution of the co-simulation.

The second method allows the user to specify the distribution of the co-simulation for a given number of nodes. By doing so, federate instances that interact with each other can be simulated on the same node to reduce the amount of network traffic. The LDSL is capable of generating these *grouped* distributions for a given number of computation nodes. These distributions group the simulations of the light controller, sensors and lights for one area all on the same node. Consequently, there may be large differences between the load of each of the nodes. Running a co-simulation consisting of 8 rooms on 7 nodes results in one node having to simulate two rooms while the others only have to simulate one room.

A third method is added to compare the impact of the allocation. This method also generates a predefined distribution over a given number of nodes. In contrast to the second method, this method does not group all federates of a single area, but it separates them by distributing them over all nodes. This represents a worst-case distribution in the sense that it maximises the communication between the nodes. We shall refer to this method as a *separated* distribution.

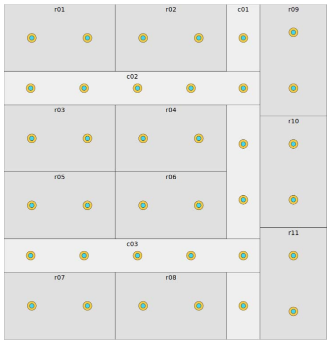

Figure 1. Visual representation of the lighting system for the first experiment.

To measure the performance difference between these three methods, a sample lighting system was designed using the LDSL. This system is displayed in Figure 1. In this figure, corridors are blocks with a light grey background, while rooms are filled a little darker. Every light is represented by a yellow circle. Occupancy sensors are displayed as smaller blue circles. The system consists of 14 light controllers, 36 lights and 38 occupancy sensors. After adding an actor and a logger, the co-simulation consists of 90 federates.

Both the weight-based and grouped distributions were executed on 2 up to 14 nodes, while the separated distribution was only executed on 7 and 14 nodes. As the exact set of nodes that executed a distributed co-simulation could change from distribution to distribution – these are selected based on availability – the speedup of a distribution is calculated by comparing its execution time to the average execution time of all collaborating nodes for this distribution. Every distribution is executed three times.

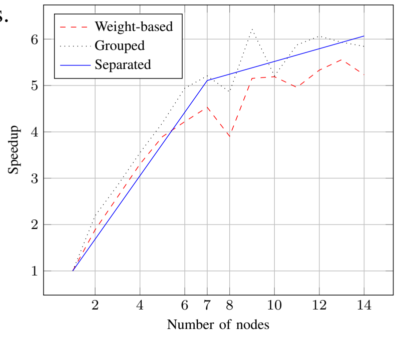

Figure 2. Average speedup achieved per distribution method.

Figure 2 shows the speedup results for the system of Figure 1. From these results, we can conclude that there is not much difference between the three distribution methods in terms of performance scaling. Even though it was expected that the separated distribution would scale worse than the others, it is in fact the one that scales best, although the differences are very small.

The results can be explained by the fact that in HLA, the publish-subscribe mechanism works on class level instead of instance level. A light controller subscribes on the *occupied* attribute of the class OccupancySensor, which means that it receives all updates from all occupancy sensors. Filtering is done in the federate implementation. Consequently, the amount of network traffic is not reduced.

B. Scalability

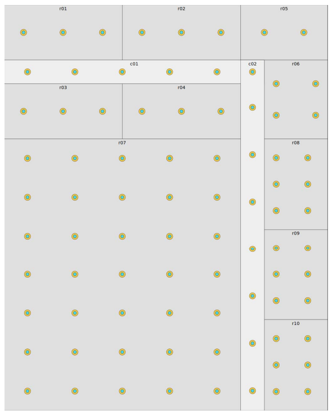

Figure 3. Visual representation of the lighting system for the second experiment.

Although Figure 2 already gives an indication of the scaling, the speedup starts flattening from a distribution over 7 nodes and up. For this, there are two possible causes: either the network overhead becomes too large or the co-simulation is simply not large enough to make efficient use of more than 7 nodes. A larger co-simulation is created to measure the scaling capacities of HLA.

The larger lighting system is roughly twice as big as the one from the first experiment. Figure 3 displays all corridors, rooms, lights (yellow) and occupancy sensors (blue). The system consists of 12 light controllers, 85 lights and 85 occupancy sensors. Including the actor and logger, the co-simulation contains 184 federates.

The co-simulation is distributed over up to 16 nodes using the weight-based distribution. Figure 4 displays the speedups that were achieved in both experiments. The speedup is calculated by comparing the simulation time for a distribution with the fastest, slowest and average simulation time of the participating nodes in this distribution.

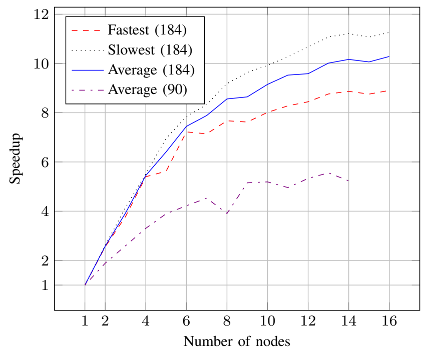

Figure 4. Speedups achieved compared to the fastest, slowest and average single node speed on the large building (184) and average speedup of the smaller building (90).

From Figure 4 it can be concluded that HLA scales well considering that distributing a large co-simulation over 10 nodes yields a speedup of 8 up to 10. We can also conclude that the scaling in the first experiment flattens due to the fact that the co-simulation was not large enough to be distributed over more than 7 nodes efficiently. This proves that large co-simulations of IoT systems can be distributed efficiently over a set of nodes.

C. Optimising HLA distribution

Even though the second experiment shows that a distributed HLA simulation scales well, the first experiment showed a possible optimisation. Due to the publish-subscribe mechanism in HLA, grouping the federates does not improve the distributed performance. To make this attribute sharing more efficient, the HLA standard includes the interface specification of the HLA Data Distribution Management (HLA-DDM). HLA-DDM allows the creation of regions to limit the scope in which attributes are shared by grouping the federates in regions. Although this seems to solve the problem, using HLA-DDM is rather difficult according to [11]. It is also mentioned that all major RTIs provide support for HLA-DDM, but we have found that this support is missing in some of the open source RTI implementations, such as OpenRTI.

Area	Instance	Class	
		SingleClass	**MultiClass**
Room 1	Controller	LightController	LightControllerRoom1
	Light 1	Light	LightRoom1
	Light 2		
	Sensor 1	Sensor	SensorRoom1
	Sensor 2		
Room 2	Controller	LightController	LightControllerRoom2
	Light 1	Light	LightRoom2
	Light 2		
	Sensor 1	Sensor	SensorRoom2
	Sensor 2		
Corridor	Controller	CorridorController	CorridorControllerCorridor
	Light 1	Light	LightCorridor
	Sensor 1	Sensor	SensorCorridor

Table I
INSTANCE AND CLASS CONFIGURATION FOR THE OLD AND NEW CO-SIMULATION SETUP.

To enable similar behaviour for federations running on RTIs not supporting HLA-DDM, each federate class may be split up for the number of groups in the co-simulation. For the lighting system, every room has its own federate class for a light instead of all rooms having instances of the same class. To illustrate this approach, Table I displays an example of the old instance and class configuration (*SingleClass*) versus the new one (*MultiClass*) for a small lighting system.

For the SingleClass setup of the small lighting system, there are 4 federate classes in total for 13 federate instances. Using the new MultiClass approach, the same system still has 13 federate instances, but now consists of 9 different federate classes. When a LightController using the SingleClass setup subscribes to an attribute from the Sensor class, it receives all updates from all instances of the Sensor class. These updates must be filtered by the instance itself, so all updates are processed by the RTI and the network. The MultiClass approach allows the LightControllerRoom1 to subscribe on an attribute from the SensorRoom1 class, which means that only updates from instances of that class will be sent to this controller instance. This should reduce the load for the RTI as well as for the network by a significant amount. This is particularly the case when the RTI is also capable of keeping those updates locally, so these do not have to travel between RTIs of the nodes in the distribution. The latter is supported by OpenRTI. The LDSL is extended to automatically generate all these separate classes for each room.

To measure the performance difference in terms of speedup between these approaches, a new lighting system is designed. Because we aim to measure only the difference between these approaches, the system consists of 16 separate rooms without corridors and interactions between them. Each of the rooms is identical and contains four lights, four sensors and one light controller. Three different approaches are tested, each of them resulting in a different number of federate classes and/or federates in the co-simulation. The first is our reference approach, in which no separate classes are generated for each room (*SingleClass*). The total amount of federates for this approach is 145: 64 lights, 64 sensors, 16 light controllers and one actor. The second is an approach for which separate classes for each area are generated (*MultiClass*). This approach results in the same amount of federates as the first method, but consists of 49 federate classes while the *SingleClass* approach consists of only 4. The third approach also generates separate actor classes for the actors, which results in a total of 160 federates. All actors play the same scenario. The approaches are distributed using the grouped distribution.

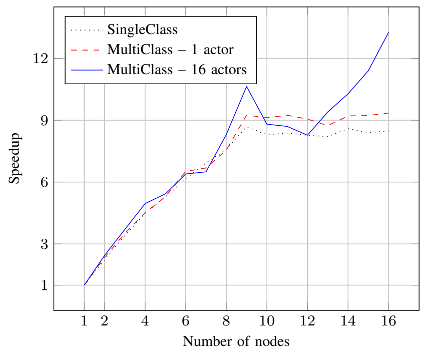

Figure 5. Average speedup achieved for the federate class configurations.

Figure 5 shows the average speedup results for the three federate class setups. With a low number of nodes, these approaches scale rather equally, but from 8 nodes up they start to differ. The difference between the SingleClass and MultiClass approach with only one single actor is relatively small. The speedup scales rather constant, which suggests this increase is achieved by requiring less network traffic. When also having separate actor classes, the speedup boosts to 13 when running on 16 nodes. The reason for this is that every node is running one single room without having interaction with federates on other nodes. Consequently, every node is basically running its own separate simulation of a room, while synchronising only time with each other. The results show that this method can be used to increase the speed of large distributed co-simulations when running on an RTI that does not support HLA-DDM.

However, not all systems are suitable for using this MultiClass approach. Systems that can be separated into rather similar and isolated subsystems are very suitable, while systems where basically all instances interact with each other are less suitable. The performance benefit is relatively small when there is at least one federate in the co-simulation that synchronises data to federates running on other nodes.

Additionally, using multiple classes also increases the absolute execution time for the co-simulation. The MultiClass approach using just one actor increased the average execution time on a single node by 17% compared to using the SingleClass approach. When also using multiple actors, the average execution time increased even further by 48%. Consequently, the absolute execution time of the MultiClass approach with 16 actors was very similar to the SingleClass approach when distributed over 16 nodes, even though the speedup for the MultiClass approach with multiple actors was better. This may be caused by the larger number of different executables using the MultiClass approach as well as the addition of 15 actors.

To increase the performance of distributed HLA co-simulation, HLA regions can be used to conditionally send attribute updates based on some requirement, such as being within a vision radius. This is not possible using the MultiClass approach. This approach, however, is very suitable for co-simulating rather statically connected IoT systems.

VI. Conclusion

By conducting a number of experiments on distributed HLA co-simulations, lessons were learned on how to deal with large co-simulations. Firstly, distributing a co-simulation over a number of computation nodes can significantly shorten the execution time of the simulation. We did not observe a limit on the number of nodes; distributed co-simulation reduced the load per node, which improved the simulation speed as long as the load is high enough.

Furthermore, in many co-simulations the exact distribution regarding the assignment of federates to nodes does not impact the speedup that can be achieved by distributing the co-simulation. Consequently, the automatic assignment of federates to nodes is a sufficiently good distribution method. Only when HLA-DDM is supported by the RTI or when the federation consists of many (almost) isolated groups of federates, creating a manual distribution over the nodes may be beneficial. The use of DSLs to generate the distributions and co-simulation code makes this approach convenient to use.

The experiments have shown that distribution of a co-simulation in HLA is a relatively easy method to speed up the simulation execution in a scalable way. In particular, the use of our automated execution script for conducting these experiments proved that it is possible to abstract from whether the co-simulation is executed in a distributed environment or not. Such a script may be generated by CoHLA, which greatly simplifies the use of cloud-based co-simulation.

Acknowledgements

We would like to thank Mathias Fröhlich for his efforts to develop OpenRTI and for the useful suggestions for performance improvements.

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] 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.

[4] 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.

[5] A. V. Brito, A. V. Negreiros, C. Roth, O. Sander, and J. Becker. Development and evaluation of distributed simulation of embedded systems using Ptolemy and HLA. In *17th Int. Symposium on Distributed Simulation and Real Time Applications*, pages 189–196, 2013.

[6] B. d'Ausbourg, J. L. Bussenot, and P. Siron. PERFOSIM: a performance evaluation tool for HLA distributed simulations. In *Proc. 6th IEEE International Workshop on Distributed Simulation and Real-Time Applications*, pages 23–30, 2002.

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

[8] 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.

[9] C. B. Jones. *Systematic Software Development Using VDM*. Prentice-Hall, 1990.

[10] P. G. Larsen, J. Fitzgerald, J. Woodcock, C. Gamble, R. Payne, and K. Pierce. Features of Integrated Model-Based Co-modelling and Co-simulation Technology. In *Software Engineering and Formal Methods*, pages 377–390. Springer, 2018.

[11] B. Möller, F. Antelius, M. Johansson, and M. Karlsson. Building Scalable Distributed Simulations: Design Patterns for HLA DDM. In *Proc. of Fall Simulation Interoperability Workshop, 2016-SIW-003*. Simulation Interoperability Standards Organization, 2016.

[12] 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.

[13] T. Nägele, J. Hooman, and J. Sleuters. Building Distributed Co-simulations using CoHLA. In *2018 21st Euromicro Conference on Digital System Design (DSD)*, pages 342–346, September 2018.

[14] 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.

[15] C. Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014.

[16] 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.

[17] 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.