Radboud Repository

Radboud University Nijmegen

PDF hosted at the Radboud Repository of the Radboud University Nijmegen

The following full text is a publisher's version.

For additional information about this publication click this link.
http://hdl.handle.net/2066/195250

Please be advised that this information was generated on 2019-10-17 and may be subject to change.

CoHLA: Design Space Exploration and Co-simulation Made Easy

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

Tim Broenink
University of Twente
Enschede
The Netherlands
t.g.broenink@utwente.nl

Jan Broenink
University of Twente
Enschede
The Netherlands
j.f.broenink@utwente.nl

***Abstract*—The inherent multi-disciplinary nature of cyber-physical systems makes it difficult to get early insight in key system properties and trade-offs that have to be made. Our aim is to support system architects of such systems by facilitating the co-simulation of models from different disciplines and design space exploration. This has been achieved by defining a domain-specific language called CoHLA which allows a high-level description of a system architecture and simulation parameters to be specified. A generator has been implemented that generates a co-simulation of component models using an implementation of the HLA standard. Component models that adhere to the FMI standard can be incorporated easily. Moreover, CoHLA includes primitives to express design space parameters and metrics; this information is used to generate tooling for automated design space exploration.**

***Index Terms*—Domain Specific Language, Cyber-physical systems, Co-simulation, Design Space Exploration, HLA, FMI**

I. Introduction

The development of a cyber-physical system (CPS) is a complex multi-disciplinary process. It involves different disciplines to develop specific subsystems, all having their own methods and workflows. To combine all these approaches together into one development process can be very challenging. To overcome this problem, model simulation can be helpful. However, different disciplines may use different modelling techniques, which makes it hard to combine into one simulation model representing the system as a whole. The use of such co-simulations of models during the development of industrial CPSs could help to speed up the development process or to gain more insight in the working of the complete system. These co-simulations could also improve the testing process by supporting hardware-in-the-loop (HIL) or software-in-the-loop (SIL) simulations. Early-stage co-simulation could also provide additional insight in design decisions during Design Space Exploration (DSE) [10].

For the construction and execution of co-simulations, standards such as the High Level Architecture (HLA) [1] and the Functional Mock-up Interface (FMI) [4] were designed. FMI aims for model interconnectivity by specifying a standard interface that wraps a model in such a way that other simulators or modelling tools can read or execute them. The FMI standard is supported by a wide range of tools, which enables the connection of a wide variety of different types of models. HLA is an interface standard that specifies a set of rules for the definition and execution of co-simulations of models created in different tools. An HLA implementation consists of a Run-Time Infrastructure (RTI) together with a set of model simulations. Each of these simulations is called a *federate*, whilst the co-simulation is called a *federation*.

Although the definition of a co-simulation using HLA is very useful, it is rather complex and time consuming [8][6]. An HLA configuration is based on a configuration file, called the Federation Object Model (FOM), which stores all information about the connected federates and their attributes. Additionally, each simulation requires a federate implementation specific for the RTI implementation of HLA. This is particularly time-consuming when the co-simulation changes frequently. Consequently, constructing a co-simulation of a system using HLA might be very costly.

Therefore, we aim for a model-based approach that enables easy usage of HLA during the development of CPSs. Models that are currently only being used for the design of specific subsystems could also be used to get a better understanding of the system design as a whole.

Additionally, we also aim for a clear separation and co-simulation of models of hardware and models of software. Explicit modelling of the software architecture improves the possibilities for concurrent development of a CPS. For this, we will also provide support for co-simulating models created in the Parallel Object-Oriented Specification Language (POOSL) [15]. POOSL is particularly useful for rapid modelling of software architectures and POOSL models can be simulated using the Rotalumis simulator. Hence, the combination of these goals should greatly simplify the construction of powerful HLA co-simulations. By supporting both the FMI standard and POOSL as modelling tools, support for a wide variety of modelling techniques is embedded in our method.

Our second goal is to provide an automated method to perform DSE on a set of models to gain more insight in the system as a composition of subsystems. This requires a structured method to specify a design space together with a set of performance metrics to be able to compare the results of each of the executed design space configurations. The simulation results and performance metric results should be properly organised to support the system architect in making

the right design decisions. Automated execution of the system co-simulation over a design space could very well be done overnight so that the designers can proceed their work the next morning while having the most recent results.

A. Approach

To achieve the goals described above, we will continue our work on building a Domain Specific Language (DSL) to quickly construct HLA co-simulations [11]. This DSL is called CoHLA (Configuring HLA) and can be used to specify simulation models as federate objects, each of them having their own attributes, timing behaviour and simulator type. By means of the DSL these objects can easily be reused and connected to each other to form a co-simulation of models. CoHLA also provides functionality to enable logging and to change model initialisation parameters. From a co-simulation instance defined in CoHLA, code is generated that can be integrated with an implementation of HLA. This concerns, for instance, wrapper code to allow the communication between an FMI-based simulation and RTI implementation. Currently, the generator of the DSL leads to code for use with OpenRTI[1]. OpenRTI is an open-source implementation of the HLA standard written in C++. The DSL relies on a set of libraries written for OpenRTI to connect different types of models to the RTI. These libraries have been introduced in [11].

The approach proposed here is tested on a number of case studies. In this paper, we describe an academic system which was constructed such that it shares key characteristics with a confidential industrial case that we are working on in parallel.

B. Related work

There already is a number of tools and methods to explore the design space during the design of CPSs. The INTO-CPS [9] project is working on a tool chain that also focuses on co-simulating heterogeneous models of systems. INTO-CPS offers an analysis tool called Automated Co-model Analysis to perform design space exploration [7]. The analysis reports a number of objectives for each of the configurations to support comparison and design decisions. Since the INTO-CPS project greatly relies on a specified set of tools, it is difficult to change to another tool chain once using INTO-CPS. We intend to use existing standards to allow interchangeability between tool chains.

The OpenMETA [14] tool chain contains a number of tools that support quite extensive DSE [13]. The construction of a co-simulation of a bunch of models, however, is rather time consuming as it requires a lot of meta modelling. The tool chain also does not work with an existing co-simulation standard for heterogeneous sets of models.

Our research is also related to work that intends to facilitate the use of HLA and exploit the combination with the FMI standard. The SEE HLA Starter Kit [6] has been developed to make the construction of an HLA-based simulation less time consuming and error-prone. It provides a Java-based software framework to support the implementation of space simulators.

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

The use of HLA's RTI as a master for FMI compatible simulation components has already been proposed in [2]. [17] describes a mechanism to develop an HLA-compliant federate using a wrapper that connects a FMU to HLA. Similarly, Neema et. al. [12] have demonstrated an approach to integrate multiple FMUs [4] into one HLA co-simulation in which FMU containers are automatically wrapped as federates. The definition of a co-simulation still requires quite some effort because a number of meta-models have to be specified.

A few possibilities to combine HLA and FMI are discussed in [8]. Our CoHLA approach matches the adapter-based approach, using the FMI for co-simulation. Our main achievement is the automatic generation of wrappers and XML descriptions based on a concise instance of a DSL. A related tool that is capable of creating model-based co-simulations of systems based on generated federate code is SimGE [16]. The generated code, however, is limited to skeleton code and hence still requires some manual implementation steps.

II. Sample case: SliderSetup

To develop, evaluate and illustrate the proposed approach, a SliderSetup example was designed to reflect a number of problems that were encountered in an industrial case we are working on. The SliderSetup consists of two independent rails, each having a movable pin mounted, which is displayed in Figure 1. A motor for each pin allows it to move over its rail, which is 30 cm long. One rail is mounted at the bottom of a cage while the other is mounted at the top. Both pins are directed to the centre of the cage. Consequently, the pins can collide with each other in the middle.

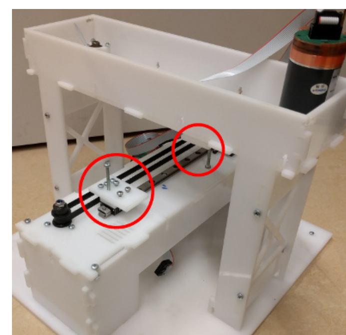

Figure 1. The assembled SliderSetup. Marked in red are the movable slider components.

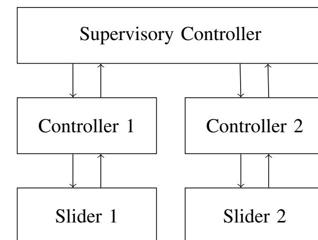

Figure 2. Architectural overview of the SliderSetup.

The SliderSetup consists of five subsystems, which are displayed in Figure 2. Since both sliders and their controllers are exactly equal, the system consists of three different subsystems that should be developed. The components are all connected by wires.

- **Supervisory controller**: Controller running on an embedded board that provides setpoints and running modes to both motor controllers. The supervisory controller coordinates movements of both sliders of the SliderSetup.
- **Slider controller**: Controls a single slider motor by providing the electrical input for the motor. Receives

setpoint information as input from the supervisory controller.
- **Slider**: The slider component contains a motor which receives electrical input from the controller and produces some kinetic action. It also contains a limit switch which sends a signal when the slider is close to the end of the rail.

For each of these subsystems a model is created. The supervisory controller was modelled using POOSL and is a discrete-time model. Both the slider controller and the slider were created in 20-sim[2] and are discrete-time and continuous-time models respectively. The 20-sim models were exported to Functional Mock-up Units (FMUs), which are simulation containers using the FMI standard. An earlier case study, however, shows that the models could also be created in OpenModelica[3] or any other modelling tool that supports exporting an FMU.

III. CoHLA

This section starts with a brief description of the structure of our DSL approach as introduced in [11] and our steps towards the extension for automated DSE. CoHLA is developed using Xtext [5] together with a code generator developed using Xtend [3]. These tools are based on the Eclipse IDE[4] and are suitable to create a custom language with a code generator, possibly for multiple targets.

A. Co-simulation definition in CoHLA

CoHLA can be used to define a co-simulation by specifying a number of simulation models and their connections. The main language constructs that are available to define a co-simulation are described below.

- **FederateClass**: Each simulation model should be specified in a FederateClass. An example for the Slider in the SliderSetup is displayed in Listing 1.

```
1  FederateClass Slider {
2   TimePolicy RegulatedAndConstrained
3   Attributes {
4    Input Boolean enable
5    Input Real encoder
6    Output Boolean limit_switch
7    Input Real motor
8    Output Real position
9   }
10  Initialisables {
11   Initialisable MaxmiumDistance "Maxmium.
       distance" as Real
12   Initialisable Position_realInitial "
       position_real.initial" as Real
13  }
14  SimulatorType FMU
15  DefaultModel "SliderAxis.fmu"
16  DefaultStep Time
17  DefaultStepSize 0.00005
18  DefaultLookahead 0.00001
19 }
```

Listing 1. FederateClass definition for the Slider in the SliderSetup.

[2] http://www.20sim.com/
[3] https://openmodelica.org/
[4] https://www.eclipse.org/

It contains the type of simulator to use (line 14), the default model to use (line 15), some HLA-specific information (lines 2, 17 and 18) and the externally visible attributes in the model (lines 3 to 9). An attribute consists of an HLA sharing type, a type, and a name. Additionally, initialisation attributes can be defined for a FederateClass to specify some internal model attributes (lines 10 to 13) that can used to assign initial values.

- **Confederation**: A confederation is a co-simulation definition, consisting of a set of federate instances and their connections. A part of the SliderSetup Confederation is shown in Listing 2.

```
1  Confederation SliderSetup {
2   Instances {
3    Instance bottomSlider as Slider
4    Instance topSlider as Slider
5    Instance bottomController as Controller
6    Instance topController as Controller
7    Instance supervisoryController as
         HighLevelSliderController
8   }
9   Connections {
10   Connection { bottomController.limit_switch
         <- bottomSlider.limit_switch }
11   Connection { bottomController.encoder <-
         bottomSlider.encoder }
12  }
13 }
```

Listing 2. Confederation definition of the SliderSetup.

Each federate instance should be given a unique name together with a FederateClass to specify its simulation model (lines 2 to 8). A set of connections can be created to specify how the attributes from different federate instances are connected (lines 9 to 12).

- **ClassInitialisation**: A ClassInitialisation consists of a number of values for initialisation attributes for a specific FederateClass. Such a ClassInitialisation can be applied to federate instances, see the Situation below. To allow reusability, each ClassInitialisation is specified by a unique name. Listing 3 shows an example with name "onLimit".

```
1  ClassInitialisation onLimit for Slider {
2   Position_realInitial = "-0.15"
3  }
```

Listing 3. ClasInitialisation for the Slider model in the SliderSetup.

- **Situation**: A situation can be used to describe the initial states of federate instances in the federation. It applies given ClassInitialisations – containing initialisation values for attributes – on federate instances.

```
1  Situation customMPs {
2   Apply customMP to bottomController
3   Apply customMP to topController
4   Apply onLimit to topSlider
5  }
```

Listing 4. Situation for the SliderSetup.

Each situation consists of a set of ClassInitialisations that are applied on specific federate instances within the Confederation. The situation in Listing 4, for example, applies the Configuration "onLimit" that was given in Listing 3

on the federate instance with the name *topSlider*. Multiple ClassInitialisations can be applied to a single federate instance, they will be applied in the order of appearance.

CoHLA supports code generation for OpenRTI. It is, however, relatively simple to add code generators for other HLA implementations. Simulation models following the FMI standard and POOSL models are supported. The OpenRTI libraries that were created also allow easy addition of a logger to the co-simulation.

From a co-simulation definition in CoHLA, a co-simulation project is generated. Such a project consists of two parts. The first part includes the HLA configuration file (FOM XML) together with a CMake[5] project. The sources of the project are all C++ source and header files that depend on the OpenRTI libraries described earlier. For each of the FederateObjects defined in the co-simulation definition, code is generated that can be compiled to an executable. The code includes all handlers for sharing attributes, simulating the model and connecting to an RTI.

The second part consists of a Python[6] script and a set of configuration files. The Python script allows the system architects to easily build and run the co-simulation using rather simple commands. The configuration files contain information about the attribute connections between federates and Situations that can be applied. The Python script ensures that all federates are being started according to the configuration specified by the system architect. This modular approach allows us to generate federate code only once, while being able to configure different co-simulations by only specifying another configuration file.

B. Design Space Exploration

To define a design space to explore, initialisation attributes in CoHLA are very useful. As Situations and ClassInitialisations affect the initialisation values for federate instances, these can also be used to specify a design space. CoHLA has been extended by allowing the user to specify lists of different initialisation values for federate instances or attributes. As there are three methods to assign initialisation values, we also consider three different types of lists that can be used to define a design space.

- *Lists of initialisation values* are lists that specify specific values for initialisation attributes of a specific federate instance. For every initialisation attribute for every federation instance, such list may be given in a DSE definition.

```
1 Set bottomSlider.startPosition : "0.15", "
      0.05", "-0.05", "-0.14"
```

Listing 5. Example of a list of initialisation values.

- *Lists of ClassInitialisations* are lists that specify ClassInitialisations that should be applied to a specific federate instance. A DSE definition may contain one list for every federate instance in the federation.

[5]https://cmake.org/
[6]https://www.python.org/

```
1 ClassInitialisations for bottomSlider :
      onLimit1, onLimit2
```

Listing 6. Example of a list of ClassInitialisations.

- *Lists of Situations* may contain situations that should be applied on the federation. Since a situation may apply ClassInitialisations to multiple federates, only one list per DSE definition is allowed.

```
1 Situations : situation1, situation2
```

Listing 7. Example of a list of Situations.

The exploration behaviour of the design space is described as either *independent* or *linked*. Linked DSE requires all design space lists to have equal length and executes the simulations in order. Independent DSE combines every element in the list with all other design space elements defined. Table I shows a design space for a hypothetical system. The system consists of two federates, each having two attributes. All but one attributes have three possible design parameters. When using *independent* exploration behaviour, there are 27 possible system configurations: (x, a, q, u), (x, a, q, v) .... *Independent* behaviour can only be used when leaving out Attribute C from the design space, which results in three possible system configurations: (x, a, u), (y, b, v) and (z, c, w).

Federate	Attribute	Design Space
Federate 1	Attribute A	x, y, z
	Attribute B	a, b, c
Federate 2	Attribute C	q
	Attribute D	u, v, w

Table I
HYPOTHETICAL DESIGN SPACE.

```
1 DSE strokeStartPositions {
2   SweepMode Independent
3   Set bottomSlider.Position_realInitial :  "0.15",
        "0.05", "-0.05", "-0.14"
4   Set supervisoryController.initSpeed : "1.0", "
      1.5", "2.0", "2.5", "3.0"
5 }
```

Listing 8. Design Space definition for the SliderSetup.

Listing 8 displays a design space definition for the SliderSetup. The design space specifies two lists, each containing initialisation attribute values. The initialisation attributes are used to specify a starting position for the bottom slider and an initialisation speed for the supervisory controller. As the SweepMode is set to *Independent*, the design space has a size of 20 $(4 \cdot 5)$ possible configurations.

From this design space definition, a DSE configuration file is generated by the code generator. These configuration files can be used to easily perform a different DSE as described in Section III-A. The simulation results will be written to log files.

C. Metrics

CoHLA DSE allows system architects to automatically execute different system configurations. Each of these system configurations result in some logging information that should be processed. Especially when the design space is rather large, this may be very time consuming. To overcome this

problem, we extended CoHLA with a method to specify performance metrics. At the end of each system configuration, each performance metric result is calculated and logged. After exploring the design space, all metric results are collected and logged to a separate file. This approach simplifies the comparison of different configurations in the design space by only comparing values of interest instead of processing all simulation data.

CoHLA currently supports four types of performance metrics that can be specified. These four types are briefly described below.

- **Error**: An error metric can be used to calculate the mean error for an attribute relative to another attribute. This metric can be useful to compare a measured value with a target value. Calculation of the mean squared error is also supported.
- **EndValue**: Returns the end value for a specific attribute. Optionally, the end value is returned as an absolute value, which is particularly useful to calculate the distance to the end value of a optionally referenced attribute.
- **Min/Max**: Returns the minimum or maximum value an attribute has reached during the simulation.
- **Timer**: A timer metric can be used to return the time when a condition is first met during the simulation. A condition consists of a comparison between an attribute value and a specified constant value.

Performance metrics must always be defined in a *MetricSet*, which is a set of metrics. A MetricSet requires a unique name and includes a measure time, which specifies the duration of a simulation that should be used to calculate the metric results. It should also contain one or more metrics. In addition to the configurable measure time, timer metrics can also be used as an end condition. These optional end conditions override the provided measure time and cause the simulation to stop once all timer metric conditions – that are flagged as end condition – are met. These flags are particularly useful to avoid unnecessary simulation after the behaviour of interest has finished.

```
1 MetricSet Initialisation {
2  MeasureTime: 300.0
3  Metric InitialisationTime as Timer for
       supervisoryController.initialised == true (
       EndCondition)
4  Metric MinBottomPosition as Minimum of
       bottomSlider.position
5 }
```

Listing 9. MetricSet to measure the initialisation speed of the SliderSetup.

Listing 9 shows the MetricSet "Initialisation" for the SliderSetup. A time to measure and two metrics are defined. The "InitialisationTime" timer metric returns the time when the supervisory controller first sets the SliderSetup as being initialised. As this metric is flagged to be an end condition, the simulation will be stopped when this condition is met. The minimum position of the bottom slider during this initialisation will be returned by the second metric.

For each MetricSet in a Confederation a configuration file is generated. This configuration can be passed to the run script as described in Section III-A to measure the metrics provided in the MetricSet. When performing a DSE, the run script will collect all results and combine them in a single output file.

IV. Results

This section describes a design space experiment that was conducted on the SliderSetup that was introduced in Section II. Section IV-A explains the design aspects that were to be investigated, after which Section IV-B describes how this has been done using CoHLA. Finally, Section IV-C provides the simulation results for our DSE and explains how the approach supported the design process.

A. Goals

Our goal is to design an initialisation procedure for the SliderSetup. When the power is turned on, the current location of both sliders is unknown. The goal of the initialisation is to move the sliders as fast as possible to a known position. This is done using a limit switch, which is located at one end of the rail. Figure 3 illustrates the positions of the sliders and limit switches of the SliderSetup.

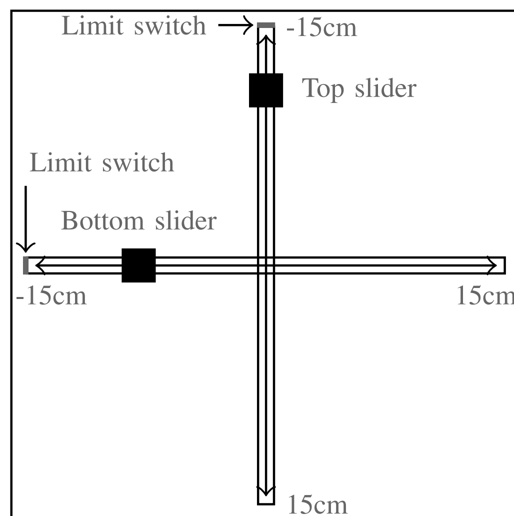

Figure 3. SliderSetup geometry as seen from above.

When the slider passes the limit switch, a signal is triggered. Since the position of the limit switch is known to be at position -15 cm on the rail, the position of the slider is known from the moment the limit switch's signal is triggered. However, after passing the limit switch, there is only 2 mm of rail left before hitting the hard limit, which should be avoided as it may introduce errors in the calibration of the system. We will use DSE for finding a suitable initialisation procedure that quickly initialises the system and does not collide with the hard limit of the rail.

Three types of control modes are supported by our controllers. The first mode (StrokeMode) requires a setpoint and a stroke time to be set. The controller then moves to the setpoint position in exactly the duration of the stroke time. Secondly, there is a mode that moves to a given setpoint as fast as possible (FastMode). The last mode that is supported requires

only a speed setpoint (FixedMode). The controller will then move at the specified speed until the speed is changed.

As for the best initialisation method we seek, we can use either of the three control modes. Therefore, we use DSE to compare these three initialisation procedures with each other. For all modes, we iterated over a small set of start positions of the slider. For the StrokeMode, we will also iterate over a set of stroke times to modify the movement speed. This will also be done for the FixedMode, in which we can iterate over different movement speeds.

For all three modes, the initialisation procedure consists of the slider moving towards the limit switch and moving back to position -10 cm when the limit switch is triggered. Once this position is reached, the initialisation is finished. Expected is that methods that initialise with a higher movement speed are more likely to hit the hard limit. Therefore, DSE may help us in finding a proper trade-off between speed and accuracy.

B. Implementation

Since not all modes require the same parameters, we will split up the design space into three smaller design spaces in CoHLA: one design space for each initialisation mode. For each of the modes, four starting positions of the slider will be simulated, being 0.15, 0.05, -0.05 and -0.14 m. As both sliders and their controllers are exactly the same, the top slider will already be located on the limit switch, so that we only measure the initialisation time for one slider. Measuring the initialisation time for only one slider decreases our design space by a factor of 2.

```
1 DSE FixedMode {
2   SweepMode Independent
3   Set bottomSlider.Position_realInitial : "0.15",
      "0.05", "-0.05", "-0.14"
4   Set hlController.initSpeed : "0.02", "0.04", "
      0.06", "0.08", "0.1", "0.12", "0.14", "0.16"
      , "0.18", "0.2"
5 }
```

Listing 10. DSE configuration for comparing initialisation procedures.

Listing 10 displays the design space definition for the FixedMode. With four different starting positions and ten different initialisation speeds (m/s) the design space dimension is 40 configurations. The configuration for the StrokeMode is the same as Listing 8. The design space for the FastMode is the smallest, with only four configurations, as the only parameter to sweep over is the starting position of the slider. The design spaces that will be explored are displayed in Table II.

DS Name	Attribute	Values
StrokeMode	Starting position (m)	0.15, 0.05, -0.05, -0.14
	Stroke time (s)	1.0, 1.5, 2.0, 2.5, 3.0
FastMode	Starting position (m)	0.15, 0.05, -0.05, -0.14
FixedMode	Starting position (m)	0.15, 0.05, -0.05, -0.14
	Movement speed (m/s)	0.02, 0.04, 0.06, 0.08, 0.10, 0.12, 0.14, 0.16, 0.18, 0.20

Table II
DESIGN SPACES FOR THE INITIALISATION PROCEDURE TO EXPLORE.

The metrics used to evaluate and compare the configurations were already shown in Listing 9. The first metric checks when the initialisation is finished and causes the simulation to stop when this has been achieved, while the second metric reports the minimum position that was reached. This minimum position can be used to calculate the overshoot by subtracting -15, which is the first position where the limit switch triggers.

C. DSE results

For all three design space definitions, the simulations were automatically executed using the generated run script. Figure 4 shows the DSE results. System configurations that exceeded

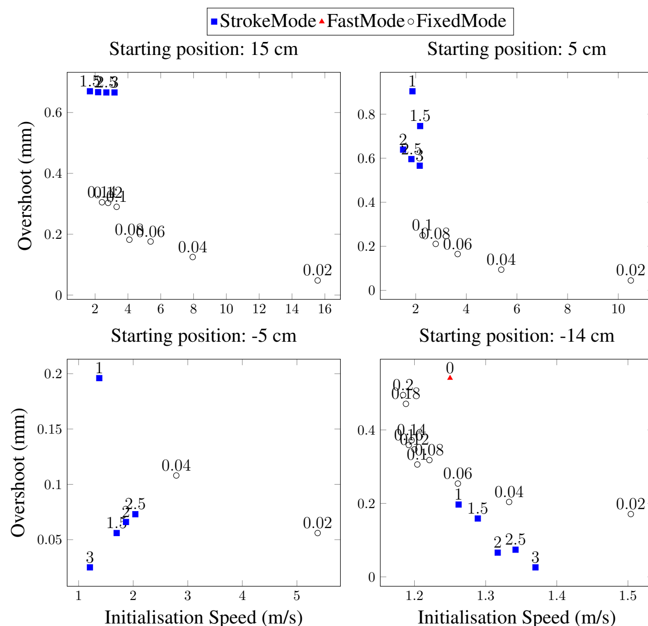

Figure 4. DSE results for the different initialisation procedures of the Slider-Setup. Every mark represents a single simulation configuration. Different marks and colours are used to represent the three possible initialisation modes. The coordinate is determined by its overshoot and speed of initialisation. Movement speed is displayed above each data point. When no movement speed parameter was used, '0' is displayed.

the hard limit were ignored in the figure, because they did not meet one of the requirements. Note that only one starting position for the FastMode initialisation is displayed in the results, because all other starting positions resulted in a large overshoot. Based on the results displayed in Figure 4 the following can be concluded.

- StrokeMode is typically faster in finishing the initialisation, except for when the starting position is very close to the limit switch, i.e. -14 cm.
- Strokemode results in a bigger overshoot than FixedMode when the starting position of the slider is far away from the limit switch.
- Movements speeds higher than 0.04 m/s are unsuitable in FixedMode, as they cause an unacceptable overshoot for starting position -5 cm and are therefore not displayed in Figure 4.

To select an initialisation procedure, a trade-off between initialisation speed and overshoot should be made. Since the initialisation procedure should never have an overshoot of

more than 2 mm, configurations exceeding this overshoot are not displayed. From these DSE results, it appears that either FixedMode using a movement speed of 0.04 m/s or StrokeMode using a stroke time of 1.5 s are most suitable to select.

Using CoHLA to co-simulate the SliderSetup has proven to provide useful insights in the system's internals during the design process. More than once, errors in the models of the controllers were found by running a co-simulation with a specific scenario that was not tested before. Additionally, the inclusion of DSE was only a small step once the system architecture was created in CoHLA and provided a tool to explore different implementation possibilities.

V. Conclusion

We presented CoHLA: a DSL that enables system architects to quickly define an HLA co-simulation of a system. It allows easy connection of models created in different modelling tools by supporting the FMI standard. CoHLA requires only minimal knowledge of HLA and FMI to construct a co-simulation and generates all ingredients required for running a co-simulation. A class definition of each simulation model and a specification of their connections are sufficient for CoHLA to generate configuration files, federate implementations and a script to easily run the simulation. Once the models of the SliderSetup were ready, defining the co-simulation of them only required half an hour of work.

This paper presents an extension that also brings automated DSE to CoHLA. A co-simulation definition in CoHLA can easily be extended with a specification of a design space and a set of performance metrics to support system designers to make design decisions. CoHLA generates a configuration file for each design space and set of metrics. These configuration files can be used to let the run script automatically execute all system configurations in the design space and output the metric results to a file.

After testing our approach on a sample system, we found that DSE was easy to use and it clearly showed the impact of selecting different initialisation procedures.

In future work, we will improve the usability of CoHLA for performing automated robustness tests by adding fault injection. We will also aim for a method that allows easy distribution of the simulation executions. Although distributed simulation is supported by HLA, CoHLA currently does not provide easy setup of such a co-simulation. Additionally, CoHLA currently focuses on attribute sharing across federates instead of interaction by means of messages. It may be interesting to add support for this type of communication.

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] 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.

[6] A. Falcone and A. Garro. The SEE HLA starter kit: Enabling the rapid prototyping of HLA-based simulations for space exploration. In *Proceedings of the Modeling and Simulation of Complexity in Intelligent, Adaptive and Autonomous Systems 2016 (MSCIAAS 2016) and Space Simulation for Planetary Space Exploration (SPACE 2016)*, MSCIAAS '16, pages 1:1–1:8. Society for Computer Simulation International, 2016.

[7] J. Fitzgerald, C. Gamble, P. G. Larsen, K. Pierce, and J. Woodcock. Cyber-physical systems design: Formal foundations, methods and integrated tool chains. In *2015 IEEE/ACM 3rd FME Workshop on Formal Methods in Software Engineering*, pages 40–46, May 2015.

[8] A. Garro and A. Falcone. On the integration of HLA and FMI for supporting interoperability and reusability in distributed simulation. In *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, DEVS '15, pages 9–16. Society for Computer Simulation International, 2015.

[9] P. G. Larsen, J. Fitzgerald, et al. Integrated tool chain for model-based design of Cyber-Physical Systems: The INTO-CPS project. In *2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)*, pages 1–6, April 2016.

[10] N. Mühleis, M. Glaß, L. Zhang, and J. Teich. A co-simulation approach for control performance analysis during design space exploration of cyber-physical systems. *SIGBED Rev.*, 8(2):23–26, June 2011.

[11] T. Nägele and J. Hooman. Rapid Construction of Co-Simulations of Cyber-Physical Systems in HLA Using a DSL. In *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 247–251, Aug 2017.

[12] H. Neema, J. Gohl, Z. Lattmann, J. Sztipanovits, G. Karsai, S. Neema, T. Bapty, J. Batteh, H. Tummescheit, and C. Sureshkumar. 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.

[13] H. Neema, Z. Lattmann, P. Meijer, J. Klingler, S. Neema, T. Bapty, J. Sztipanovits, and G. Karsai. Design space exploration and manipulation for cyber physical systems. In *IFIP First International Workshop on Design Space Exploration of Cyber-Physical Systems (IDEAL'2014), Springer-Verlag Berlin Heidelberg*, 2014.

[14] J. Sztipanovits, T. Bapty, S. Neema, L. Howard, and E. Jackson. *OpenMETA: A Model- and Component-Based Design Tool Chain for Cyber-Physical Systems*, pages 235–248. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[15] 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.

[16] O. Topçu, L. Yilmaz, H. Oğuztüzün, and U. Durak. *Distributed Simulation*. Springer, 2016.

[17] 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.