AARHUS UNIVERSITY

Coversheet

This is the Preprint of the article.

How to cite this publication

Please cite the final published version:

Casper Thule, Kenneth Lausdahl, Cláudio Gomes, Gerd Meisl, Peter Gorm Larsen, Maestro: The INTO-CPS co-simulation framework, Simulation Modelling Practice and Theory, Volume 92, 2019, Pages 45-61, ISSN 1569-190X, <https://doi.org/10.1016/j.simpat.2018.12.005>.

Publication metadata

Title:	Maestro: The INTO-CPS co-simulation framework, Simulation Modelling Practice and Theory
Author(s):	Casper Thule, Kenneth Lausdahl, Cláudio Gomes, Gerd Meisl, Peter Gorm Larsen
Journal:	Simulation Modelling Practice and Theory
DOI/Link:	https://doi.org/10.1016/j.simpat.2018.12.005
Document version:	Preprint

General Rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognize and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

If the document is published under a Creative Commons license, this applies instead of the general rights.

Maestro: The INTO-CPS Co-Simulation Orchestration Engine

Casper Thule^{a,*}, Kenneth Lausdahl^{b,1}, Peter Gorm Larsen^a, Gerd Meisl^c

^a*Digit, Aarhus University, Finlandsgade 22, 8200 Aarhus N, Denmark*

^b*Mjølner Informatics A/S, Finlandsgade 10, 8200 Aarhus N, Denmark*

^c*TWT GmbH Science & Innovation, Ernstthaldenstraße 17, 70565 Stuttgart, Germany*

Abstract

Enabling interoperability between different simulators can be a challenge. Here the Functional Mockup Interface (FMI) standard is an attempt of standardising interfaces between independent simulation units adhering to FMI. In this paper, we present the open-source orchestration engine called Maestro that performs co-simulation using such units. Maestro was developed with the goal of supporting challenges in co-simulation faced by industry, which other engines did not address. Therefore, Maestro was used by industry partners in several industrial case studies and is one of the few FMI orchestration engines supporting both Linux, Windows, and Mac, along with implementing distributed co-simulation across platforms and architectures. We present Maestro and its features along with using it for a case study of a Line Following Robot. Maestro is capable of performing co-simulation using a fixed or variable (adaptive) iteration scheme as well as hierarchical co-simulations following version 2.0 of the FMI standard including a few extensions.

Keywords: Co-Simulation, Hybrid Co-Simulation, Discrete-Event Simulation, Continuous-Time Simulation, Distributed Simulation, Functional Mock-up Interface

2010 MSC: 00-01, 99-00

*Corresponding author.

Email addresses: `casper.thule@eng.au.dk` (Casper Thule), `kg1@mjolner.dk` (Kenneth Lausdahl), `pg1@eng.au.dk` (Peter Gorm Larsen), `g.meisl@mytum.de` (Gerd Meisl)

¹Employed at Digit, Aarhus University at the time of writing.

1. Introduction

Our daily life increasingly involves interaction with cyber elements controlling physical processes, e.g. cars, trains, and building automation. Such systems are referred to as Cyber-Physical Systems (CPSs). As technology is evolving, these systems are becoming increasingly complex, which in turn also makes the development process more challenging. To assist in the development of CPSs, it can be useful to create models of the constituent components that jointly shape the system. A model is an abstract description of a constituent component ignoring irrelevant properties and fidelity is a measure of how close a model is to the real component. It is possible to describe these models using different paradigms, and the case study presented later in the paper uses Discrete-Event (DE) models, representing cyber elements, and Continuous-Time (CT) models, representing physical processes. These models can then be employed in a collaborative simulation, a co-simulation, which is capable of coupling models created in different formalisms [1]. This enables the possibility of simulating the entire system by simulating the constituent components and couple their respective inputs and outputs in a sound fashion.

Maestro was developed as part of the INTO-CPS project, which concerned enabling co-simulation using the Functional Mock-up Interface (FMI) standard and realising the benefits in integrated tool chains [2, 3]. The main purpose of Maestro was to support different challenges that industry needed support for in co-simulation orchestration engines, which did not exist at the time. The development of Maestro was therefore driven by requirements from industry. To ensure that Maestro successfully satisfied the needs of industry, it was applied by industry partners in different domains such as automotive, railways, maritime, building automation, and agriculture (e.g., [4, 5, 6, 7, 8, 9]). In order to illustrate the majority of the tool, this paper features a small case study where a Line-Following Robot (LFR) is used, which is representative of the mentioned challenges.

- 30 The main contribution of this work is the industry tested co-simulation orchestration engine called Maestro with the following features/characteristics:
- Co-simulation using the FMI 2.0 standard.
 - Standalone orchestration engine with REST-like API and single shot simulation.
 - 35 • Open-source, cross-platform, and cross-architecture (Win32, Win64, Linux32, Linux64, Mac64).
 - Distributed co-simulation across platforms and architectures.
 - Fixed and variable (adaptive) step size.
 - Internal calculation of derivatives, if these are not provided by simulation
 - 40 units.
 - Constraints such as: Bounded difference, Zero-crossing, FmuMaxStep-Size [10]), and sampling rate.
 - Passed crosscheck performed by the FMI organisation using 323 FMUs exported by other tools across the supported platforms.

45 This combination of features and the degree of verification (crosschecking and industry usage) makes Maestro unique compared to other engines.

The remaining part of this article begins with the necessary background information in Section 2. This section will also present related work consisting of other tools supporting FMI. The core of the article describes the co-simulation

50 orchestration engine in Section 3. Afterwards, the small LFR case study is used to illustrate features of Maestro in Section 4. Then Section 5 provides an overview of the expected future work. Finally, the article is completed in Section 6 with concluding remarks, which will also position Maestro compared to the other tools described in Section 2.

55 2. Background

The FMI standard is a tool independent standard for the exchange of models and co-simulation, where we will restrict ourselves to co-simulation in this article. It is the result of the ITEA2 European Project called MODELISAR, which

concerned improving the design of systems and embedded software in vehicles
 60 [11]. The concept is to couple a number of simulation units that each contain
 their own solvers and the data exchange is restricted to discrete communication
 points. In practice, the standard provides and describes C-interfaces and the
 structure of a static description file. A component implementing the C-interfaces
 and providing a static description file is then referred to as an Functional Mock-
 65 up Unit (FMU). Figure 1 shows an example of four FMUs that are connected
 via their input/output ports.

Figure 1: Example of a simulated CPS with dependencies between slaves (the gray boxes) via their respective ports (the black ellipses) (taken from [10]).

It is the responsibility of a Master Algorithm (MA) to orchestrate the simulation. This includes satisfying the dependencies of the FMUs by retrieving the outputs and distributing them to the corresponding inputs. The MA also allows
 70 the individual FMUs to progress for determined time steps, which is carried out by invoking the function `doStep` on the FMUs. This is an iterative process that continues until a user defined end time is reached or an unrecoverable error occurs. Some FMUs support getting and setting states, which enables the possibility of performing rollbacks and attempting to perform a step with a smaller
 75 step size. Maestro contains such a MA along with an interface exposed as a REST-like web-service in order to make it easy to use.

The Integrated Tool Chain for Model-based Design of Cyber-Physical Systems (INTO-CPS) project uses the FMI standard for unifying the tools in a

tool suite [12]. The tool suite offers a chain of interlinked tools, that support the development of CPSs from requirements to realisation in mechanical- and electronic hardware and software. The overall workflow is shown in Figure 2, where Maestro is used for Model-in-the-Loop (MiL)- and Hardware-in-the-Loop (HiL)/Software-in-the-Loop (SiL) simulation. MiL, HiL, and SiL mean that one or more FMUs engaging in the co-simulation are respectively a model, the real hardware or the real software. Four industrial case studies ranging over railways, building automation, agriculture, and automotive drove the development of the MA to ensure that it addressed industrial needs. Furthermore, these case studies (as well as a collection of smaller academic pilot studies) were used to demonstrate and evaluate the tool chain. Maestro has also been used for other projects such as maritime [9], manufacturing [13], and cooperative swarms of drones [14].

Figure 2: Overall workflow and services offered by the INTO-CPS tool chain.

In order to understand the sections below, a brief introduction to some of the tools of INTO-CPS is required. The connections between constituent models/-components that jointly form the system is expressed in a special CPS profile on top of SysML using the Modelio Tool [15]. SysML has the concept of profiles, which is a way of providing extensions for particular domains. Therefore, a SysML profile has been created for CPSs to connect models in an easy manner using model descriptions from FMI [16]. The tools 20-sim [17] and OpenMod-

elica [18] are used to model CT parts of a system, such as mechanical and
 100 electrical components. The DE parts of a system are modelled using the no-
 tation VDM-RT [19]. VDM-RT extends VDM-SL [20], such that it can model
 and analyse Real-Time embedded and distributed systems, object-orientation
 and concurrency. The open-source tool Overture can export VDM-RT models
 as FMUs [21, 22, 23].

105 The FMI website contains a Tools page², which shows tools supporting FMI
 1.0 and/or FMI 2.0 either as slaves (produces FMUs) or masters (performs sim-
 ulations using FMUs). This page also shows, how many FMUs a given master
 has been crosschecked with. A successful crosscheck means, that a master has
 successfully imported and simulated an FMU that is exported from another
 110 tool. According to this page, Maestro³ is the only tool that has been success-
 fully crosschecked on Windows, Linux, and Mac. According to the Tools page
 over 100 tools support or plan to support FMI either as slaves or masters. In
 Table 1 we attempt to briefly present a few tools, that we believe represents
 the diversity of the master tools. It is out of scope of this paper to elaborate
 115 and compare all tools performing co-simulation using FMI. The reason for this is
 that co-simulation is a large domain and the feature set of different engines vary
 substantially. Furthermore, several of these engines are proprietary, requires
 writing source code using a particular framework, or are not standalone but
 integrated into different environments, e.g. Eclipse. The tool descriptions can
 120 therefore not be considered exhaustive, as it would be a study in itself. Gomes
 has led the development of a survey and state of the art on co-simulation,
 a taxonomy to classify co-simulation approaches, and classifies several co-
 simulation approaches/tools/frameworks both with and without FMI [1, 24].

²<http://fmi-standard.org/tools/>. Visited on August 16, 2018.

³Maestro is called “INTO-CPS Co-simulation Orchestration Engine (COE)” on the Tools
 page.

Product	Description
DACCOSIM [25]	Eclipse-Plugin with additional features; Companion tool DacRun to compile, run, and collect results using DACCOSIM; Open-source; Cross-platform; FMI Tools website shows no cross-check results; Multithreaded and distributed architecture; Approximates discrete events [26], e.g. events in DACCOSIM occurs in a time interval, but these are to be considered instantaneous [27]. Maestro both approximates and allows the FMUs to report their specific synchronisation time using an FMI extension, which is described in Section 3.1.
DYMOLA	Proprietary; Windows only; FMI Tools website shows successful crosscheck results; Tool for both modeling and simulation; Both slave and master; Approximates discrete events in a fashion similar to DACCOSIM [26].
FIDE [26]	IDE for building applications using FMUs (Maestro is standalone); open-source; cross-platform; Based on the open-source Ptolemy II framework [28] for analysis and design of heterogeneous systems; Supports many extensions to FMI 2.0 [10, 29] including the one mentioned under DACCOSIM.; FMI Tools website shows planned FMI 2.0 support ⁴ .
PyFMI ⁵	Python package for loading and interacting with FMUs and thereby not a standalone tool; Open-source; FMI Tools website shows crosscheck results for Win32 only; Based on FMILibrary ⁶ .
C2WT [30]	C2WT leverages the High Level Architecture (HLA), a general purpose architecture for distributed simulation systems, to perform co-simulation. The work presents a novel approach that integrates FMUs within the HLA-based simulations.

Table 1: Tools with FMI.

⁴Publications indicate that FMI 2.0 is supported.⁵<https://jmodelica.org/pyfmi/index.html> Visited on Aug. 15, 2018⁶<https://jmodelica.org/fmil/FMILibrary-2.0.3-htmldoc/index.html> Visited on Aug. 15, 2018. FMILibrary is a C library serving as foundation for applications interfacing FMUs

The tools in Table 1 show, that existing tools are adopting FMI and new tools are being developed specifically for utilising FMI. Moreover, the work carried out in C2WT on integrating FMI and High Level Architecture (HLA) is particularly interesting, as HLA is a widely used simulation standard used in e.g. NATO [31] and the US Department of Defense [32], and the work on extending C2WT was sponsored by the US Department of Defense. Another example is [33], where FMI is compared to another framework called Functional Digital Mock-Up (FDMU). Furthermore, it is proposed how to combine these approaches. Several papers also address the shortcomings of the FMI standard, e.g. [34], which addresses the lack of thread safety across multiple instances of the same FMU. Broman also presents several proposals for future hybrid (Mixed Discrete Event and Continuous Time) co-simulation standards and guidance for the development of FMI. These proposals mainly concern the handling of instantaneous events and the representation of time. Cremona et. al. [29] takes this further by actually realising these proposals in code as extensions to the FMI standard. The following sections will present Maestro and afterwards in Section 6 Maestro will be related to the tools above in a similar format and commented on.

3. The Maestro Co-Simulation Orchestration Engine

As mentioned, Maestro was developed as part of the INTO-CPS project. When the INTO-CPS project ended in December 2017, the maintenance and further development of Maestro was transferred to the INTO-CPS Association⁷. It is therefore actively being maintained and developed. Maestro is open-source with no proprietary parts and is available at <https://github.com/INTO-CPS-Association/maestro>. It is possible to download the tool on the Releases page and launch it locally. The protocol is described in [35] and directly from the git repository at <https://git.io/fNpq>. The preferred way to try it out is to download the cross-platform INTO-CPS Application at <https://into-cps-association.github.io/download/>. It features a download manager that can be used to install Maestro, and features to configure and launch a co-simulation using Maestro. Furthermore, it contains an examples downloader, which downloads and configures examples. These examples are also available on github in the INTO-CPS Association at <https://github.com/INTO-CPS-Association> and

⁷<http://into-cps.org/> Visited on Aug. 16, 2018.

presented in [36]. Additionally, [37] presents guidelines on how to use the INTO-CPS technology. Finally, <https://git.io/fNjvJ> is a step-by-step tutorial on how to use the INTO-CPS technology to model a system in SysML, Create and Export FMUs, utilise additional tooling of the INTO-CPS tool chain, and perform co-simulation.

160 Maestro is implemented in a combination of Java, Scala, and C. Java and Scala is used for the main logic and C is used to interface with FMUs. The tool is cross-platform and available on Mac 64-bit, Linux 32 and 64-bit, and Windows 32 and 64-bit⁸. Maestro is implemented as a REST-like web service and therefore does deliberately not feature a User Interface (UI). The reason for this is that we are not
165 interested in performing any sort of lock-in to the INTO-CPS tool chain. It should be possible to use the individual parts of the developed technology in different workflows, tool chains, and contexts. However, as mentioned above, other technology for INTO-CPS has been developed, including a decoupled cross-platform UI for Maestro called the INTO-CPS Application. This tool is also open-source and available at Github
170 within the INTO-CPS Association.

A simulation is carried out by first sending a configuration to Maestro. This configuration contains the FMUs that are part of the simulation, their parameters, the connections between FMUs, and the simulation setup, which will be described below (this configuration can be generated from Modelio if one has used the SysML CPS
175 profile). The model descriptions of the FMUs are statically checked against an FMI validation schema and these along with the configuration file are additionally checked against semantic rules defined in the FMI standard that are not part of the FMI validation schema [38]. Once the configuration is accepted by Maestro, the simulation can be started. During the simulation it is possible to hook in to live streaming of
180 outputs from the FMUs. Once the simulation is finished, the results can be retrieved. This is the general flow for carrying out a simulation using Maestro, but there are additional options not presented in this article [35]⁹.

The simulation setup contains the simulation algorithm, or MA, to use. A simula-

⁸A cross-compile service for FMUs, where the source code is available, was also developed and it compiles to Mac, Linux 32- and 64-bits and Windows 32- and 64-bit. The service can be accessed at <https://sweng.au.dk/fmubuilder/>.

⁹These options are e.g. multiple sessions, parallelisation [39], stabilisation, and making Maestro perform in real-time.

tion can be executed using fixed step size or variable (also referred to as adaptive) step
 185 size. Using fixed step each **doStep** is performed with the same step size throughout
 the simulation. By lowering the step size a more accurate simulation can be achieved
 but the time the co-simulation takes will be prolonged. Accuracy is a measure of how
 close a simulated behavior trace is to an analytical behavior trace [1]. It is not nec-
 essarily required to run with the same step size throughout an entire co-simulation to
 190 achieve a usable result, thus the motivation for variable step is to combine a accurate
 simulation with a time-efficient simulation. It is necessary to constrain a simulation
 using variable step size, such that the important points in time are chosen.

The core of a co-simulation iteratively performs **doSteps**. The iteration method
 used by Maestro is based on the Jacobi communication approach [40, 41]. This ap-
 195 proach iterates as follows: All FMUs perform a **doStep**, then the dependencies are
 resolved by getting and setting outputs/inputs on the FMUs, and this process iterates
 until a predetermined end-time is reached. This iteration method was selected as it
 supports parallelisation well and worked for all the case studies.

The next section describes the variable step functionality and related constraints.
 200 Afterwards, the concept of hierarchical co-simulation is explained and motivated, as
 Maestro has support for it. The challenge of executing the constituent FMUs of a
 co-simulation on different platforms is addressed in Section 3.3 and finally Section 3.4
 will present use cases on using Maestro as a service.

3.1. Variable Step

205 When using the variable step functionality of Maestro it is necessary to limit the
 step size using max step size or other constraints. This is because the algorithm seeks
 to perform the maximal step possible, and if no constraints are specified then the
 maximal step is equal to the end time. The constraints inform the algorithm about
 what is important in the models, such that it will step near interesting points in
 210 time. In practice, the variable step process is carried out as follows: Before a step is
 taken, an entity referred to as the Variable Step size Calculator (VSC) is invoked and
 calculates the step size based on the current time, the previous step size, the current
 output values, the (estimated) output derivatives of the FMUs, and possibly based
 on one or more constraints. If multiple constraints are specified, then the chosen step
 215 size will be the minimum of the step sizes calculated based on constraints. After a
 step has completed, the VSC is invoked to validate whether any of the constraints

have been violated. If this is the case, then a rollback is initiated (if supported by the FMUs) and the VSC provides a new and reduced step size. This can occur i.e. when derivatives are missing, the signals are changing in a unpredictable fashion, or
220 incorrect configuration. The mathematical foundation and details of the constraints are elaborated on in [35] and <https://git.io/fNpaq>.

Currently four kinds of constraints are available. In order for these constraints to work as intended, the second-order derivatives are required. If these are not provided by the FMUs, then they are estimated. This also makes it possible to utilize second-order
225 derivatives for FMUs that do not offer these. The kinds of constraints are described below:

Zero Crossing: A zero crossing constraint is a continuous constraint. A zero crossing occurs at the point where the value of a function changes its sign. In simulation, it can be important to adjust the step size such that a zero crossing is revealed as
230 accurately as possible. For instance, a ball should rebound from a wall exactly when the distance between the ball and the wall hits zero and not before or after. As performing rollbacks is not efficient and might not be possible, the VSC bases its step size adjustments on the prediction of a future zero crossing. It uses extrapolation and derivative estimation to estimate changes and therefore
235 reduce the need for performing rollbacks.

Bounded Difference: A bounded difference constraint is also a continuous constraint. A bounded difference ensures that the minimal and maximal value of a set of values do not differ by more than a specified amount (the underlying assumption is that this difference becomes smaller when the step size is reduced).
240 The bounded difference problem is distinct from the zero-crossing problem in that there is not a specific time instant (the zero crossing) to hit, but rather a specific time difference (the step size that keeps the difference bounded). A concrete example of use is to bound the error of a function calculating the heatflow between two models where each of them only have a prediction of the flow from
245 the other model.

Sampling Rate: A sampling rate constraint is a discrete constraint. It constrains the step size such that repetitive, predefined time instants are exactly chosen without any jitter. This can be useful in co-simulation, for instance, when a modelled control unit reads a sensor value every x milliseconds. The chosen

250

step size is either the time difference between the current time and the time instant of the next sampling, or the maximal stepsize, whichever is smaller. Thus by using this constraint it is possible to gain some of the properties from fixed step.

FMU Max Step Size: This constraint implements the proposed FMI extension `getMaxStepSize` from [10]. The motivation behind implementing this functionality is to avoid the need for rollbacks and choose the largest step size possible. The extension proposes that each FMU implements a function called `getMaxStepSize`, which returns the largest possible step it can perform at the given point in time. Maestro then chooses the minimum of the largest possible step sizes as the next step size as follows:

$$size = \min(\{getMaxStepSize(i) \mid i \in instances\})$$

255

Each FMU is guaranteed to be able to perform this step, as it must be able to perform any steps smaller than its maximum step according to the FMI 2.0 standard. This feature is not part of the FMI 2.0 standard, but FMUs generated by tools in the INTO-CPS tool chain implement this functionality. Additionally, this functionality allows for FMUs to decide the step size, which can help to precisely synchronise at desired points in time, e.g. when events occurs.

260

3.2. Hierarchical Co-Simulation

Maestro has been extended with a feature enabling the concept of Hierarchical co-simulation, which is illustrated in Figure 3. Here, FMU A and B, the connections between them and the simulation settings are imploded into one FMU, thereby making this FMU a fully configured co-simulation. This is referred to as a Hierarchical FMU and it can be used as a regular FMU alongside FMU C in a co-simulation using Maestro. Thus Maestro supports both importing FMUs and co-simulation scenarios. The feature is motivated by two different use cases, one concerning co-simulation performance and the other concerning employing co-simulation.

270

To illustrate the first use case consider Figure 3 again with FMU A and B being tightly coupled FMUs and therefore need to synchronize often. However, they are loosely coupled with FMU C and thereby do not need to synchronize with C as often.

Figure 3: Hierarchical co-simulation

Using hierarchical FMU it is possible to co-simulate FMU A and B with small steps while allowing for C to take larger and fewer steps, thus achieving a performance enhancement of the co-simulation.

The second use case is because the FMI standard does not specify how full modularization can be achieved for a subsystem, but is limited to providing a convenient way of sharing and encapsulating single models. To exemplify this, consider the scenario depicted in Figure 4. System A is being developed, and this system uses subsystem B from another company, which uses most of its components from third-parties. The produce of the subsystem can make use of any FMUs provided from their third-party suppliers and perform co-simulation. However, they cannot produce an FMU for the complete subsystem based on their co-simulation because there is no straightforward way to share and encapsulate co-simulation scenarios containing simulation settings and connections between the FMUs. It can be time-consuming and challenging to configure such a co-simulation scenario for the engineers of system A, as they might not have knowledge of the internal structure of the subsystem. Therefore, it is desirable to provide a means to encapsulate and share co-simulation scenarios, such that they can be utilized as a component in another co-simulation. This is possible if Subsystem B is turned into a hierarchical FMU.

3.3. Distributed Co-Simulation

The FMI standard enable models to be shared as FMUs but in practice this only works if all FMUs taking part in a co-simulation support the same combination of platform and architecture. This limits how these FMUs can be shared e.g. an FMU exported as Windows-32 bit cannot be used in a co-simulation with another FMU that only supports Windows 64-bit. The same limitation applies if the platform is

Figure 4: System with subsystem that uses third-party components

different, which was the case for MAN Diesel & Turbo [9], where a control system was developed for a water handling system. The system was used to clean the exhaust gas using exhaust gas recirculation for a large two-stroke maritime combustion engine.

300 The physical model was developed in MATLAB 64-bit for Windows and the control system was developed in an internal framework for Linux 32-bit with full support for HiL simulation. Thus one of the main challenge was to perform co-simulation across platforms and architectures.

To overcome this challenge with mixed platforms and architectures a plug-in was developed for Maestro. The plug-in solved both the platform and architecture issue and improved performance for some system architectures. It implements a distributed factory that enable FMU instances to be relocated to remote simulation daemons, which then perform the simulation as shown in Figure 5. The communication is carried out using Java-Remote Method Invocation (JAVA-RMI) and therefore a performance gain can only be achieved if the added communication overhead is outweighed by the benefit from utilising the extra resources on the remote host. However, the concern is not performance but simply to allow co-simulation by enabling interoperability.

3.4. Using Maestro as a Service

As mentioned above, Maestro exposes a web service with a Representational State Transfer (REST) Application Programming Interface (API) and therefore does not include a UI. The motivation behind this decision is to provide a core simulation service using a standardised and well known interface approach in a way that lets others embed it into their workflows, thereby supporting batch processing, cluster deployment, etc.

Figure 5: Distributed Extension Overview

One of these workflows is Design-Space Exploration (DSE) [42], which can be described as: “An activity undertaken by one or more engineers in which they build and evaluate models in order to reach a design from a set of requirements” [43]. In the context of co-simulation within the FMI standard this can be considered as trying different models, or different parameters to models, that represent alternative solutions to a given problem. The results are then analysed and an approach is singled out based on an objective that is important to the user, e.g. cost or performance [37]. This is the methodological aspect, whereas there is another aspect of actually performing this search. Searching the design space can be carried out in various fashions, as Maestro does not dictate a specific approach. “The design space is the set of possible solutions for a given design problem” [43]. In the LFR case study, presented in Section 4, Python is used, but this is up to the developer performing the DSE. The process is to complete a co-simulation, gather the results, change the parameters of the FMUs or change entire FMUs and perform another co-simulation, gather the results again and so forth until the design space has been explored. Then the different results can be compared and used to select the ‘best’ design alternatives based on a given objective. As Maestro supports live-streaming of co-simulation results it is possible to terminate a co-simulation prematurely if earlier co-simulations yielded better results. This is also an example of Maestro supporting batch processing.

TWT GmbH has used Maestro to create co-simulations from an FMU taking part in another co-simulation. They have modelled a trip assistant for electric vehicles as a co-simulation, which features a route prediction component. The role of the route prediction FMU is to provide a realistic remaining state-of-charge of the vehicle’s

battery during and at the end of a trip along with choosing a route that minimises the overall time for travelling and recharging. If notable deviations are present between the predicted values and the actual values (e.g. if the driver chooses an alternative route) then the route prediction FMU creates an entirely new co-simulation that calculates a new prediction, which is then used in the original co-simulation. This approach alters the usual notion of a co-simulation being based on a predetermined scenario and makes it dynamic. It is different from hierarchical co-simulation in the sense that when an FMU is asked to perform a step, then it spawns a new co-simulation, which finishes in the step of the FMU. In hierarchical co-simulation, the hierarchical FMU does not complete an entire co-simulation in a single step, but steps the subsystem FMUs in coordinate with the rest of the co-simulation.

4. Case Study: Line-Following Robot

The LFR is one of the academic pilot studies of the INTO-CPS project¹⁰, several more are available as mentioned in Section 3. It was originally developed in the DESTECs project and presented in [44]. The model simulates a robot that can follow a line painted on the ground. The line contrasts from the background and the robot uses infrared sensors to detect light and dark areas on the ground. The robot has two wheels, each powered by individual motors to enable the robot to make controlled changes of the direction. The number and position of the sensors may be configured in the model. A controller takes input from the sensors to make outputs to the motors.

Figure 6 provides an overview of different aspects of the example: The real robot, an example path the robot will follow, and a 3D representation in 20-sim.

The robot is modelled as four FMUs: A discrete controller model representing the logical decisions controlling the motors and two continuous model representing the physical aspects of the robot such as movement on the map and the reading of sensor values. Lastly, the fourth, optional, FMU visualises the robot in a 3D environment.

The signals are as follows:

- `lfLeft` and `lfRight` are real sensor values from the optical sensors,

¹⁰The project configuration specifically used for this case study is available at <https://git.io/fNpK0>, and the data is available at: <https://git.io/fNpi9>

Figure 6: The Line-Following Robot (LFR) and path

Figure 7: The SysML Connections Diagram for the LFR

- `lfServoLeft` and `lfServoRight` are servo Pulse-Width Modulation (PWM) signals, and
- `robot_*` are 3D position information

4.1. Establishing a Network of FMUs

375 The first step in creating a case study using the INTO-CPS tool chain is to use the connections diagram from Modelio as shown in Figure 7 and the INTO-CPS application to create a multi-model. The connections are extracted from the connections

diagram and inserted into the multi-model and thereby the application can be used to view them. The INTO-CPS application can also be used to edit the connections, instances of the FMUs, and set initial parameters. The multi-model also configures the connections between the created FMUs and the symbolic names used. A summary of this step is: Extract a multi-model from the SysML model created in Modelio, connected the created FMUs and set the initial parameters. Several multi-models can be created in a project, e.g. one with a 3D simulation FMU and one without, which is the case for this case study. For example, the connections diagram as shown in Figure 7 does not contain a 3D FMU, but a 3D FMU showing the simulation as it progresses as pictured in Figure 6c can be added as well.

4.2. Simulation

The next step is to set up a co-simulation configuration, which is coupled to the multi-model created in the previous step. Here, it is possible to create a fixed configuration specifying the end time and the step size. Another option is choosing variable step, where one has to specify initial step size, minimum step size, maximum step size and constraints, which were described in Section 3.1.

The LFR was co-simulated using two different configurations, a fixed and a variable, which both had an end time of 20 seconds. The variable co-simulation uses variable step with a minimum step size of 0.000001s, a maximum step size of 2s, an initial step size of 0.01s and the constraint `getMaxStepSize`. The step size of the variable co-simulation is decided by the discrete-event component, as long as it stays in the interval mentioned above. The fixed co-simulation uses fixed step with a step size of 0.1s. The plot in Figure 8a shows the map and the position of the robot, where the red solid line represents the variable co-simulation and the blue dashed line represents the fixed co-simulation. The variable co-simulation consists of 1992 observations and took 1 minute and 36 seconds, whereas the fixed co-simulation consists of 200 observations and took 22 seconds. Thus the fixed simulation is faster in this case, because it consists of less steps. However, the variable simulation is more accurate, as the synchronisations happen exactly when the events occur. Furthermore, if the controller is modelled with a period of x time units between synchronisations, then there is presumably a reason for why the developer chose this period.

DSE was also performed using the LFR and the experiments concerned changing

410 the robot speeds, the sensor positions, and changing both¹¹ [36]. The metrics used to calculate a ranking based on changing the sensor positions were: distance travelled, energy consumed, deviation area and maximum deviation. It showed, that placing the sensors on the front of the robot with a displacement of 10 mm to each side of the centre-line provided the best overall result [45].

(a) Co-simulation of LFR using a fixed step configuration and variable step configuration with a `getMaxStepSize` constraint. (b) Co-simulation of the LFR using a fixed step configuration and variable step configuration with a zero crossing constraint.

Figure 8: A map to follow (gray) overlaid with co-simulations of the LFR.

415 4.3. Extension: Zero Crossing

The interaction between the analogue optical sensors and the controller can be improved such that potential delays, which may be introduced by the variable step algorithm are reduced by adding a zero crossing constraint.

420 As described in Section 3.1, the constraint can ensure that the variable step algorithm will perform a step as close to the zero crossing as possible. For the LFR the zero crossing is represented as the threshold in the controller where the analogue values

¹¹A video of DSE experiments using the LFR is available at: <https://youtu.be/A95YGKpMTuM?t=1m8s>

from the sensors change between detecting black and white. By adding this constraint the controller is stepped with the new value of the sensors as close to the point where they change as possible under the constraints for the simulation. This allows the controller to react immediately in the subsequent step while a variable step simulation without this constraint could be delayed by any variable time between its current time and the maximal step size. This presents an alternative to using `getMaxStepSize`, which is an extension to FMI and therefore not supported by all FMUs, where the variable step functionality is still utilised.

An example of employing zero crossing instead of `getMaxStepSize` in a variable co-simulation is shown in Figure 8b, represented by the blue dashed line, along with a plot of a fixed step algorithm represented by the solid red line. The step size is set to 0.4s for the fixed co-simulation and for the variable co-simulation the maximum step size is set to 0.4s, the initial step size is set to 0.1s, the minimum step size is set to 0.00001s and safety is set to 100¹². The fixed co-simulation consists of 50 observations and took 14 seconds while the variable co-simulation consists of 604 observations and took 55 seconds. It can be seen in the figure that the fixed co-simulation fails to follow the line and performs a loop, whereas the co-simulation with the zero crossing follows the line. This shows that zero crossing is an alternative when applying the variable step algorithm and can react even a value, i.e. the sensor reading, changes rapidly. It is important to note that the actual behaviour of the LFR is to follow the line, and therefore the blue dashed line is the correct simulation. However, if the actual sample rate of the robot was 0.4 seconds, then the red line is more accurate.

4.4. Extension: HiL and Distributed

Maestro supports HiL simulation which can be used to improve the fidelity of the simulation model before deployment. This can be carried out by replacing the CT FMUs with a proxy FMU to the actual physical actuator or sensor. Maestro will then run a co-simulation where some of the FMUs are based on the models and the others are physical entities. The LFR controller model can be validated against the actual robot by providing proxy FMUs for the optical sensors and the two wheel servos. These proxies can either be hand-made or made using the 20-sim-4c tool [35]. The

¹²Safety adjusts the conservatism of the zero crossing prediction. See [35] for details.

simulated actuator outputs can then be fed both to the physical robot and the virtual representation for visual comparison.

4.5. *Extending the Line-Following Robot Case Study*

455 The LFR is a case study that is continuously extended to exemplify new features of Maestro or technology in general. Thus several extensions could be performed to extend the case study:

DSE on Raspberry Pies: This idea is to use cheap hardware to perform several co-simulations simultaneously. Newcastle University has purchased 50 Raspberry
460 Pies and is attempting to use them all for running DSE. Thus 50 co-simulations will run simultaneously, each on a Raspberry Pie. This requires Maestro to be cross-compiled to the architecture used on the Raspberry Pie, which has been carried out manually. The design of Maestro is suited for this as Java is supported on the Raspberry Pie and the C-code for the native part of Maestro
465 is written in a portable fashion.

Swarm of LFRs: Autonomous vehicles are being tested for adaption in the taxi industry and therefore it could be interesting to co-simulate a swarm of autonomous LFRs. Furthermore, these would interact to notify one another of positions, pick-ups and similar. Hierarchical co-simulation could be employed
470 to co-simulate such a system. Each LFR would be imploded into a hierarchical FMU, which internally would be co-simulated with a step size small enough to faithfully co-simulate the LFR. However, the interaction between the robots does not have to happen as frequently, and therefore a larger step size can be used for this.

475 **Dynamically Adjusting:** As a consequence of co-simulating a swarm of LFRs there would be several robots on the same paths. Thus one robot could interfere with the path of another robot, thereby extending the time it takes for a robot to reach a destination point, e.g. a pickup point. To predict a new arrival time would require a dynamic adjustment during the co-simulation. As seen in the
480 TWT example in Section 3.4 a new co-simulation could be spawned to calculate the new destination time.

5. Future Work

In the Modelio transformation towards the composition of FMUs that shall be included in a co-simulation, it is now possible to also include information about the units. It is envisaged that this will be taken into account in the Maestro COE including automating conversions between alternative units. It is also envisaged that more general semantic adaptations that enhances FMUs, e.g. by providing better input extrapolations, will be incorporated [46]. Both of these initiatives will ensure easier adaptation between FMUs and ensure that the units being exchanged conform to each other.

The FMI standard is currently at version 2.0 but it is expected that version 3.0 will be released during 2018 and Maestro will be updated to take advantage of the different improvements in the new version of the standard. The FMI community also have a number of working groups for how to best organise multi-models in a co-simulation setting. In order to make it easier to influence each FMU with different parameters for example for Design Space Exploration support it is envisaged that support from the group working on System Structure and Parameterization (SSP) will be incorporated. This will make it easier to generate different configurations to be used in exploring alternative candidate solutions and based on that finding a Pareto front where the most optimal candidates are placed. The Maestro development team has also taken part in suggesting how the FMI standard can be altered to improve co-simulation performance in terms of simulation speed by treating pointers differently [47].

Additionally, it is envisaged that it will make sense to combine Maestro with other types of simulation frameworks such as HLA in order to enlarge the group of potential users. Initial work on this kind of combinations has already started [48, 49]. Furthermore, the project “Application of co-simulation to support test and operations (ACOSIM)” [50] is ongoing and concerns employing FMI in the space industry. In this project, it is planned to integrate Maestro within the simulation framework used by The European Space Agency and support Simulation Model Portability 2.

Currently, Fortiss¹³ is working on adding a feature for co-simulation using Maestro for AutoFOCUS3, which “... is a highly integrated model-based tool covering the complete development process from requirements elicitation, deployment, the modelling of

¹³<https://www.fortiss.org/home/> Visited on August 16, 2018

the hardware platform to code generation.” [51]. Additionally, 4DIAC, a framework for distributed industrial automation and control, is integrating their tool within the INTO-CPS tool chain and thereby making use of Maestro and FMI [52].

6. Concluding Remarks

In this paper a Co-Simulation Orchestration Engine called Maestro based on the FMI standard 2.0 for co-simulation has been presented. The tool offers functionality to statically validate co-simulation configurations and perform centralised or distributed co-simulations using a fixed or variable step algorithm. The variable step algorithm can be constrained in several ways to ensure an efficient co-simulation and usable results. Furthermore, Maestro enables hierarchical co-simulation where FMUs can be grouped in subsystems with different co-simulation requirements and still engage in a full system co-simulation. Maestro offers co-simulation as a service and can therefore be used for purposes such as DSE or launching a nested co-simulation to predict, for example, a future position.

The functionality of Maestro has been exemplified in a case study, where a LFR consisting of four FMUs was co-simulated. The results show, that the robot can be faithfully co-simulated using both the fixed- and variable step algorithm. Furthermore, it has been described how the case study can be extended and improved by utilising additional features in Maestro.

To finish this paper, some remarks are given to position Maestro relative to the other tools presented in Section 2. It is a standalone orchestration engine, only focusing on performing the co-simulation. Related tooling such as creating FMUs, configuring co-simulation scenarios, and processing simulation results is not part of Maestro, but does exist. The philosophy is to offer individual tools that can be integrated in other workflows. Thus, Maestro and the INTO-CPS offers a full tool chain consisting of individual tools, that can be employed on a needs basis. Furthermore, Maestro is open source, cross-platform, and industrially tested. It is the only orchestration engine on the FMI Tools page that has been crosschecked for both Linux, Windows, and Mac¹⁴. Additionally, it makes use of an FMI-extension that allows FMUs to decide the step

¹⁴It is called the INTO-CPS Orchestration Engine on this website: <https://fmi-standard.org/tools/>

size, which allows for precise synchronisation. As such we finish our presentation of Maestro with an open invitation to download the tool, try it, use it in your tooling, and help improve it.

545 Acknowledgements

The authors would like to thank Cláudio Gomes for proofreading this paper and providing valuable feedback. Additionally, we would like to thank the anonymous reviewers for reviewing and providing ideas on how to improve the paper. The work presented here is partially supported by the INTO-CPS project funded by the European Commission's Horizon 2020 programme under grant agreement number 664047.
550 We would like to thank all the participants of those projects for their efforts making this a reality.

References

- [1] C. Gomes, C. Thule, D. Broman, P. G. Larsen, H. Vangheluwe, Co-simulation: a
555 Survey, *ACM Comput. Surv.* 51 (3) (2018) 49:1–49:33.
- [2] J. Fitzgerald, C. Gamble, P. G. Larsen, K. Pierce, J. Woodcock, Cyber-Physical Systems design: Formal Foundations, Methods and Integrated Tool Chains, in: *FormaliSE: FME Workshop on Formal Methods in Software Engineering, ICSE 2015, Florence, Italy, 2015*.
- 560 [3] P. G. Larsen, J. Fitzgerald, J. Woodcock, R. Nilsson, C. Gamble, S. Foster, Towards semantically integrated models and tools for cyber-physical systems design, in: T. Margaria, B. Steffen (Eds.), *Leveraging Applications of Formal Methods, Verification and Validation, Proc 7th Intl. Symp.*, Vol. 9953 of *Lecture Notes in Computer Science*, Springer International Publishing, 2016, pp. 171–186.
- 565 [4] J. Fitzgerald, C. Gamble, R. Payne, P. G. Larsen, S. Basagiannis, A. E.-D. Mady, Collaborative model-based systems engineering for cyber-physical systems, with a building automation case study, *INCOSE International Symposium* 26 (1) (2016) 817–832. doi:10.1002/j.2334-5837.2016.00195.x.

- [5] P. G. Larsen, J. Fitzgerald, J. Woodcock, T. Lecomte, Trustworthy Cyber-Physical Systems Engineering, Chapman and Hall/CRC, 2016, Ch. Chapter 8: Collaborative Modelling and Simulation for Cyber-Physical Systems, ISBN 9781498742450.
- [6] F. Foldager, P. G. Larsen, O. Green, Development of a Driverless Lawn Mower using Co-Simulation, in: 1st Workshop on Formal Co-Simulation of Cyber-Physical Systems, Trento, Italy, 2017.
- [7] L. D. Couto, S. Basagianis, A. E.-D. Mady, E. H. Ridouane, P. G. Larsen, M. Hasanagic, Injecting Formal Verification in FMI-based Co-Simulation of Cyber-Physical Systems, in: The 1st Workshop on Formal Co-Simulation of Cyber-Physical Systems (CoSim-CPS), Trento, Italy, 2017.
- [8] M. Neghina, C.-B. Zamrescu, P. G. Larsen, K. Lausdahl, K. Pierce, A Discrete Event-First Approach to Collaborative Modelling of Cyber-Physical Systems, in: Fitzgerald, Tran-Jørgensen, Oda (Ed.), The 15th Overture Workshop: New Capabilities and Applications for Model-based Systems Engineering, Newcastle University, Computing Science. Technical Report Series. CS-TR- 1513, Newcastle, UK, 2017, pp. 116–129.
- [9] N. Pedersen, K. Lausdahl, E. V. Sanchez, P. G. Larsen, J. Madsen, Distributed Co-Simulation of Embedded Control Software with Exhaust Gas Recirculation Water Handling System using INTO-CPS, in: Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2017), Madrid, Spain, 2017, pp. 73–82, ISBN: 978-989-758-265-3.
- [10] D. Broman, C. Brooks, L. Greenberg, E. Lee, M. Masin, S. Tripakis, M. Wetter, Determinate composition of FMUs for co-simulation, in: Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on, 2013, pp. 1–12. doi:10.1109/EMSOFT.2013.6658580.
- [11] ITEA Office Association, Itea 3 ? project ? 07006 modelisar, <https://itea3.org/project/modelisar.html> (December 2015).
- [12] P. G. Larsen, J. Fitzgerald, J. Woodcock, P. Fritzson, J. Brauer, C. Kleijn, T. Lecomte, M. Pfeil, O. Green, S. Basagiannis, A. Sadovykh, Integrated Tool

- 600 Chain for Model-based Design of Cyber-Physical Systems: The INTO-CPS
Project, in: CPS Data Workshop, Vienna, Austria, 2016.
- [13] M. Neghina, C.-B. Zamfirescu, P. G. Larsen, K. G. Lausdahl, K. Pierce, Multi-
Paradigm Discrete-Event Modelling and Co-simulation of Cyber-Physical Sys-
tems, *Studies in Informatics and Control* 27 (1) (2018) 33–42. doi:10.24846/
605 v27i1y201804.
- [14] G. Zervakis, K. Pierce, C. Gamble, Multi-modelling of Cooperative Swarms, 2018,
Accepted for publication in Proceedings of the 16th Overture Workshop.
- [15] A. Bagnato, E. Brosse, I. Quadri, A. Sadovykh, SysML for Modeling Co-
simulation Orchestration over FMI: the INTO-CPS Approach, *Ada User Journa*
610 37 (4) (2016) 215–218.
- [16] N. Amálio, R. Payne, A. Cavalcanti, J. Woodcock, Checking SysML Models for
Co-Simulation, *Lecture Notes in Computer Science*, Springer, 2016.
- [17] C. Kleijn, Modelling and Simulation of Fluid Power Systems with 20-sim, *Intl.*
Journal of Fluid Power 7 (3).
- 615 [18] P. Fritzson, Principles of Object-Oriented Modeling and Simulation with Modelica
2.1, Wiley-IEEE Press, 2004.
- [19] M. Verhoef, P. G. Larsen, J. Hooman, Modeling and Validating Distributed Em-
bedded Real-Time Systems with VDM++, in: J. Misra, T. Nipkow, E. Sekerinski
(Eds.), *FM 2006: Formal Methods*, *Lecture Notes in Computer Science* 4085,
620 Springer-Verlag, 2006, pp. 147–162. doi:10.1007/11813040\11.
- [20] P. G. Larsen, B. S. Hansen, H. Brunn, N. Plat, H. Toetenel, D. J. Andrews,
J. Dawes, G. Parkin, et al., Information technology – Programming languages,
their environments and system software interfaces – Vienna Development Method
– Specification Language – Part 1: Base language (December 1996).
- 625 [21] P. G. Larsen, K. Lausdahl, N. Battle, J. Fitzgerald, S. Wolff, S. Sahara, M. Ver-
hoef, P. W. V. Tran-Jørgensen, T. Oda, The VDM-10 Language Manual, Tech.
Rep. TR-2010-06, The Overture Open Source Initiative (April 2010).

- [22] K. Lausdahl, J. W. Coleman, P. G. Larsen, Semantics of the VDM Real-Time Dialect, Tech. Rep. ECE-TR-13, Aarhus University (April 2013).
- 630 [23] K. Lausdahl, J. W. Coleman, P. G. Larsen, Towards a co-simulation semantics of VDM-RT/Overture and 20-sim, in: N. Plat, C. B. Nielsen, S. Riddle (Eds.), Proceedings of the 10th Overture Workshop, no. CS-TR-1345 in Technical Report Series, Computing Science, Newcastle University, 2012, pp. 30–37.
URL <http://www.cs.ncl.ac.uk/publications/trs/papers/1345.pdf>
- 635 [24] C. Gomes, C. Thule, D. Broman, P. G. Larsen, H. Vangheluwe, Co-simulation: State of the art, CoRR abs/1702.00686. [arXiv:1702.00686](https://arxiv.org/abs/1702.00686).
URL <http://arxiv.org/abs/1702.00686>
- [25] V. Galtier, S. Vialle, C. Dad, J.-P. Tavella, J.-P. Lam-Yee-Mui, G. Plessis, FMI-Based Distributed Multi-Simulation with DACCOSIM, in: Spring Simulation
640 Multi-Conference, Society for Computer Simulation International, Alexandria, Virginia, USA, 2015, pp. 804–811.
- [26] F. Cremona, M. Lohstroh, S. Tripakis, C. Brooks, E. A. Lee, FIDE – An FMI Integrated Development Environment, in: Symposium on Applied Computing, 2015.
- 645 [27] D. Broman, L. Greenberg, E. A. Lee, M. Masin, S. Tripakis, M. Wetter, Requirements for hybrid cosimulation standards, in: Proceedings of 18th ACM International Conference on Hybrid Systems: Computation and Control (HSCC), ACM, 2015, pp. 179–188.
- [28] C. Ptolemaeus (Ed.), System Design, Modeling, and Simulation using Ptolemy II, Ptolemy.org, 2014.
650 URL <http://ptolemy.org/books/Systems>
- [29] F. Cremona, M. Lohstroh, D. Broman, E. A. Lee, M. Masin, S. Tripakis, Hybrid co-simulation: it’s about time, Software & Systems Modeling [doi:10.1007/s10270-017-0633-6](https://doi.org/10.1007/s10270-017-0633-6).
655 URL <https://doi.org/10.1007/s10270-017-0633-6>
- [30] H. Neema, J. Gohl, Z. Lattmann, J. Sztipanovits, G. Karsai, S. Neema, T. Bapty, J. Batteh, H. Tummescheit, C. Sureshkumar, Model-based integration platform

- for fmi co-simulation and heterogeneous simulations of cyber-physical systems, in:
The 10th International Modelica Conference 2014, Modelica Association, Lund,
660 Sweden, 2014.
- [31] N. A. T. O. Research, T. Organization, Implementation of HLA Compliance Certification Within NATO and NATO/PfP Nations, RTO technical report, N.A.T.O., Research and Technology Organization, 2007.
URL <https://books.google.dk/books?id=jtgFkAEACAAJ>
- 665 [32] J. S. Dahmann, R. M. Fujimoto, R. M. Weatherly, The department of defense high level architecture, in: Proceedings of the 29th Conference on Winter Simulation, WSC '97, IEEE Computer Society, Washington, DC, USA, 1997, pp. 142–149.
doi:10.1145/268437.268465.
URL <http://dx.doi.org/10.1145/268437.268465>
- 670 [33] O. Enge-Rosenblatt, C. Clauß, A. Schneider, P. Schneider, Functional Digital Mock-up and the Functional Mock-up Interface—Two Complementary Approaches for a Comprehensive Investigation of Heterogeneous Systems, in: 8th International Modelica Conference, Linköping University Electronic Press; Linköpings universitet, Dresden, Germany, 2011.
- 675 [34] S. E. Saidi, N. Pernet, Y. Sorel, A. B. Khaled, Acceleration of FMU Co-Simulation On Multi-core Architectures, in: The First Japanese Modelica Conferences, no. 124, Linköping University Electronic Press, Tokyo, Japan, 2016, pp. 106–112.
doi:10.3384/ecp16124106.
- [35] A. Pop, V. Bandur, K. Lausdahl, M. Groothuis, T. Bokhove, Final Integration
680 of Simulators in the INTO-CPS Platform, Tech. rep., INTO-CPS Deliverable, D4.3b (December 2017).
URL http://into-cps.org/fileadmin/into-cps.org/Filer/D4.3b_Integration_of_simulators.pdf
- [36] M. Mansfield, C. Gamble, K. Pierce, J. Fitzgerald, S. Foster, C. Thule, R. Nilsson,
685 Examples Compendium 3, Tech. rep., INTO-CPS Deliverable, D3.6 (December 2017).
URL http://into-cps.org/fileadmin/into-cps.org/Filer/D3.6_Examples_Compendium_3.pdf

- [37] J. Fitzgerald, C. Gamble, K. Pierce, Method Guidelines 3, Tech. rep., INTO-CPS Deliverable, D3.3a (December 2017).
 690 URL http://into-cps.org/fileadmin/into-cps.org/Files/D3.3a_Method_Guidelines_3.pdf
- [38] M. Hasanagić, P. W. V. Tran-Jørgensen, K. Lausdahl, P. G. Larsen, Formalising and Validating the Interface Description in the FMI standard, in: The 21st
 695 International Symposium on Formal Methods (FM 2016), 2016.
- [39] C. Thule, P. Larsen, Investigating concurrency in the co-simulation orchestration engine for into-cps, Vol. 28, Institute for System Programming, Russian Academy of Sciences (ISPRAS), 2016, pp. 139–156, this publication is part of the Horizon
 2020 project: Integrated Tool chain for model-based design of CPSs (INTO-CPS),
 700 project/GA number 644047. doi:10.15514/ISPRAS-2016-28(2)-9.
- [40] J. Bastian, C. Clauss, S. Wolf, P. Schneider, Master for Co-Simulation Using FMI, in: 8th International Modelica Conference, 2011.
- [41] C. Thule, C. Gomes, J. Deantoni, P. G. Larsen, J. Brauer, H. Vangheluwe, Towards the Verification of Hybrid Co-simulation Algorithms, 2018, Accepted for
 705 publication at the CoSim-CPS-18 Workshop.
- [42] C. Gamble, R. Payne, J. Fitzgerald, S. Soudjani, F. F. Foldager, P. G. Larsen, Automated exploration of parameter spaces as a method for tuning a predictive digital twin, Submitted for publication.
- [43] J. F. Broenink, J. Fitzgerald, C. Gamble, C. Ingram, A. Mader, J. Marincic, Y. Ni,
 710 K. Pierce, X. Zhang, Methodological guidelines 3, Tech. rep., The DESTECs Project (INFSO-ICT-248134) (October 2012).
- [44] C. Ingram, K. Pierce, C. Gamble, S. Wolff, M. P. Christensen, P. G. Larsen, D3.4a – Examples Compendium, Tech. rep., The DESTECs Project (INFSO-ICT-248134) (January 2013).
 715 URL <http://destecs.org/images/stories/Project/Deliverables/D34aExamplesCompendium.pdf>

- [45] C. Gamble, K. Pierce, Design space exploration for embedded systems using co-simulation, in: J. Fitzgerald, P. G. Larsen, M. Verhoef (Eds.), Collaborative Design for Embedded Systems, Springer Berlin Heidelberg, 2014, pp. 199–222.
- 720 [46] C. Gomes, B. Mayers, J. Denil, C. Thule, K. Lausdahl, H. Vangheluwe, P. D. Meuliere, Semantic Adaptation for FMI Co-simulation with Hierarchical Simulators, Is accepted for publication on December 31, 2017 in Simulation: Transactions of the Society for Modelling and Simulation International journal.
- [47] K. Lausdahl, C. Thule, P. Larsen, J. Höll, C. König, A. Klueber, M. Pfeil, 725 V. Fässler, The INTO-CPS Co-Simulation Orchestration Engine – Experiences with FMI 2.0 and proposed extensions, in: FMI User Meeting, 2017, FMI User Meeting at the Modelica Conference 2017; Conference date: 15-05-2017.
URL https://svn.fmi-standard.org/fmi/branches/public/docs/Modelica2017/08_20170515_FMI_user_meeting_Prague.pdf
- 730 [48] M. U. Awais, M. Cvetkovic, P. Palensky, Hybrid simulation using implicit solver coupling with HLA and FMI, International Journal of Modeling, Simulation, and Scientific Computing.
- [49] Thomas Nägele and Jozef Hooman, Co-simulation of Cyber-Physical Systems using HLA, in: Proceedings 7th IEEE Annual Computing and Communication 735 Workshop and Conference (CCWC 2017), IEEE, 2017, pp. 267–272.
- [50] corallia, ESTEC awards "ACOSIM - Application of co-simulation to support test and operations" project to EMTECH Space P.C. consortium - Corallia, <http://www.corallia.org/en/si-news/si-cluster-news/4011-estec-acosim-emtech.html>, (Accessed on 08/16/2018) (February 2018).
- 740 [51] V. Aravantinos, S. Voss, S. Teufl, F. Hölzl, B. Schätz, Autofocus 3: Tooling concepts for seamless, model-based development of embedded systems, in: I. Dragomir, S. Graf, G. Karsai, F. Noyrit, I. Ober, D. Torre, Y. Labiche, M. Genero, M. Elaasar (Eds.), Joint Proceedings of the 8th International Workshop on Model-based Architecting of Cyber-physical and Embedded Systems and 1st International Workshop on UML Consistency Rules (ACES-MB 2015 & WUCOR 745 2015) co-located with ACM/IEEE 18th International Conference on Model Driven

Engineering Languages and Systems (MoDELS 2015), Ottawa, Canada, September 28, 2015., Vol. 1508 of CEUR Workshop Proceedings, CEUR-WS.org, 2015, pp. 19–26.

750

URL <http://ceur-ws.org/Vol-1508/paper4.pdf>

- [52] Jose Cabral and Monika Wenger and Alois Zoitl, Enable co-simulation for industrial automation by an fmu exporter for iec 61499 models, 2018, accepted for IEEE 23rd International Conference on Emerging Technologies and Factory Automation.