# Maestro: The INTO-CPS Co-simulation Framework

Casper Thule[a,*], Kenneth Lausdahl[b,1], Cláudio Gomes[c], Gerd Meisl[d], Peter Gorm Larsen[a]

[a] *DIGIT, Aarhus University, Finlandsgade 22, 8200 Aarhus N, Denmark*
[b] *Mjølner Informatics A/S, Finlandsgade 10, 8200 Aarhus N, Denmark*
[c] *University of Antwerp, Prinsstraat 13, 2000 Antwerpen, Belgium*
[d] *TWT GmbH Science & Innovation, Ernsthaldenstraße 17, 70565 Stuttgart, Germany*

## Abstract

Cyber-Physical Systems (CPSs) often operate in a critical context where it is crucial that they behave as intended. However, the heterogeneous nature of CPSs makes them inherently challenging to develop. To assist in the development process, one can perform co-simulation, where models of constituents of a CPS are coupled to jointly simulate the full system. The challenge herein is to combine heterogeneous formalisms in a sound fashion and address practical needs such as stability, performance, platform compatibility and so forth. To address this, Maestro is a tool for co-simulation using models adhering to the Functional Mock-up Interface standard for Co-Simulation. Its development was driven by needs from different industry domains such as railways, agriculture, building automation and automotive. It supports both a fixed and variable constraint-based iteration scheme along with platform distribution capabilities. The tool is open-source as an attempt to increase adoption of co-simulation and encourage researchers to collaborate. Maestro has been validated by industry through application in the aforementioned domains. It is a step in the direction of the two-folded long-term goals: ensure trustworthy co-simulation results and make co-simulation a technology taken for granted.

[*] Corresponding author.

*Email addresses:* `casper.thule@eng.au.dk` (Casper Thule), `kgl@mjolner.dk` (Kenneth Lausdahl), `Claudio.GoncalvesGomes@uantwerpen.be` (Cláudio Gomes), `g.meisl@mytum.de` (Gerd Meisl), `pgl@eng.au.dk` (Peter Gorm Larsen)

[1] Employed at Aarhus University at the time of writing this article.

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

*2010 MSC:* 00-01, 99-00

## 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 matures, it enables the development of more complex systems, which in turn creates pressure for better development processes. To assist in the development of such CPSs, it is desireable to perform full system evaluation.

An approach to carry out such evaluation is to simulate the system. However, a CPS consists of multiple constituent components, whose behavior is best computed by different simulators [1]. To get the overall behavior of the system, these simulators need to cooperate, in what we denote *co-simulation*. Co-simulation "...consists of the theory and techniques to enable global simulation of a coupled system via the composition of simulators" [2]. This coupling is carried out in a straightforward way: by connecting the outputs of one simulator to the inputs of other simulators (see section 2), and executing each simulator in tandem with the other simulators.

Despite the apparent simplicity of this procedure, there is still no solution that ensures that the results produced by the co-simulation can be trusted. This is because the communication delays, numerical approximations, and simulator coordination, all contribute to the errors in the results. As a result, researchers have set out to understand this topic, and there is already an extensive body of knowledge, as can be concluded from recent surveys [2, 3, 4, 5].

These surveys identify a number of challenges all related to ensuring trustworthy results. However, these works do not give sufficient attention to the practical challenges faced by industry, when using co-simulation to evaluate complex CPSs. This has been tackled by a more recent empirical survey [6]. In this work, the authors applied the Delphi method to interview practitioners and researchers on perceived challenges regarding the Functional Mock-up Interface (FMI) for co-simulation. The FMI 2.0 standard for co-simulation defines how to package, implement, and provide information on a simulator. Essentially, the results in [6] provide a ranking of academic challenges, and challenges that have not received enough attention.

The following challenges where identified as barriers to the adoption of co-simulation:

- "There is insufficient documentation and a lack of examples, tutorials, etc."
- "There is not enough cooperation and exchange (theoretical/numerical, implementation, application/industry) in defining and developing the FMI standard."
- "There is a lack of (scientific) community, forums, groups."
- "There is a lack of tools that sufficiently support FMI."

Beside the challenges above, the communication between the Integrated Tool Chain for Model-based Design of Cyber-Physical Systems (INTO-CPS) project and its industrial follower group have highlighted that:

- multi-platform deployment of the same simulator is a barrier to the use of co-simulation; and
- variable step size orchestration algorithms need to allow for fine-grained tunning, as each co-simulation problem has particular characteristics that only the domain experts know about.

Maestro, the open-source co-simulation framework presented in this work, is an attempt to address the challenges described above. It was developed as part of the INTO-CPS project to enable co-simulation using the FMI standard and increase the benefits in integrated tool chains [7, 8, 9]. In this project, the main purpose of Maestro was to address the challenges that industry faced when applying co-simulation. The maintenance and further development of Maestro has been transferred to the non-profit INTO-CPS Association[2]. It is available at https://github.com/INTO-CPS-Association/maestro and the protocol is described in [10] and https://git.io/fNpaq.

Maestro is a co-simulation framework that has been validated extensively in wildly different fields such as automotive, railways, maritime, building automation, and agriculture (e.g. [11, 12, 13, 14, 15, 16]). Additionally, we report on lessons learned and future research directions.

The remaining part of this article begins with the necessary background information in section 2, which introduces a running case study of a Line-Following Robot (LFR) to help guide the reader through various concepts. Afterwards, Maestro is described in section 3 followed by section 4 presenting the applicability of Maestro, validating the tool and reflecting on lessons learned. Next, section 5 presents similar co-simulation tools and positions Maestro relative to these. Section 6 concerns the future of Maestro and how we aim to improve its applicability in the domain of co-simulation. Finally, section 7 concludes this work.

## 2. Background

In this section we introduce the main concepts required to understand the motivation behind some of the features of Maestro, and how these are implemented. This is carried out by first introducing modelling and simulation concepts before continuing on to co-simulation. To make the explanation of these concepts clearer, we resort to a well known running example, introduced below.

### *2.1. Running Example: The Line Follower Robot*

The LFR is one of the academic pilot studies of the INTO-CPS project[3]. It was originally developed in the DESTECS project and presented in [17].

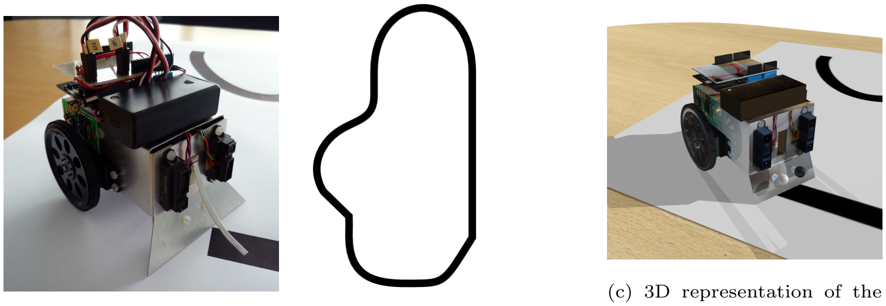
[2] http://into-cps.org/ Visited on November 7, 2018.

[3] The 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

The robot, shown in fig. 1a is supposed to follow a line painted on the ground (an example line is shown in fig. 1b). The line contrasts with the background and the robot uses infrared sensors to detect light and dark areas on the ground.

Figure 2 shows the conceptual subsystems and their interconnections. The robot has two wheels, each powered by individual motors that allow the robot to change direction. The number, and position, of the sensors may be configured in the model.

To quickly predict the behavior of the robot under different sensor configurations and lines, a model of the robot has been built, a 3D image of it is shown in fig. 1c. The behavior of this model is computed using co-simulation.



(a) The physical LFR

(b) A line-follow path

(c) 3D representation of the LFR

Figure 1: The Line-Following Robot (LFR) and path

### 2.2. Modelling and Simulation Concepts

We will adopt the nomenclature in [2], and use the illustration in fig. 3 to exemplify the following concepts. A model of an original system, which is an existing system or system that does not yet exist, is an abstraction whose behavior should match the original system behavior, with respect to some goal. As the example illustrated in fig. 3 shows, the original system is the LFR, whose behavior can be measured by experimentation. The model of the physical LFR (a collection of differential equations) is sketched at the bottom of the figure. Its behavior can be obtained via simulation, that is, by using an algorithm which interprets the model and produces a behavior trace. We call such an algorithm

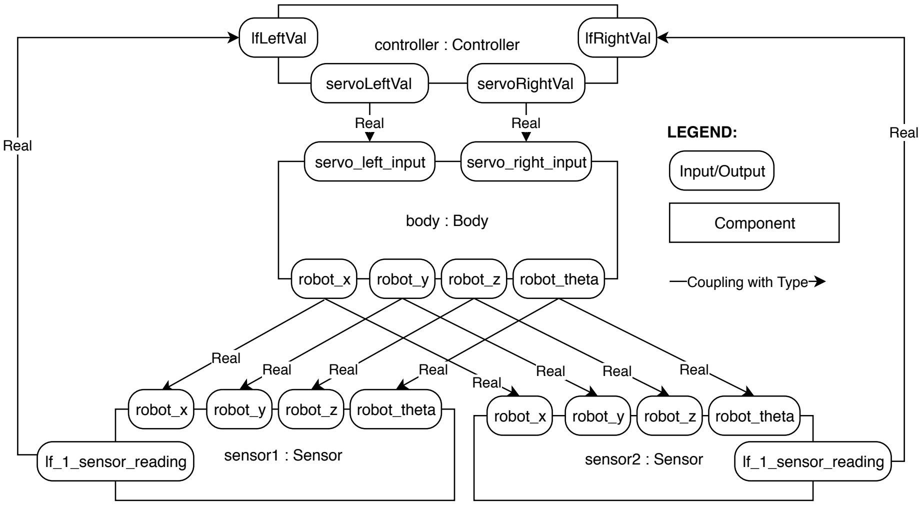


Figure 2: The Connections Diagram for the LFR.

"the solver". The combination of a solver and a model forms a simulator. The difference between a simulator and a solver is that the solver requires a model and its inputs signals, in order to compute the behavior of the model. The simulator, on the other hand, just needs the input signals.

We distinguish three kinds of models: Continuous-Time (CT), Discrete-Event (DE), and hybrid. CT models describe how the abstracted state of the system evolves continuously over time, whereas DE models describe how the state evolves as a reaction to events. These evens can be internal (triggered by the passage of time), or external (caused by the inputs). Hybrid models combine characteristics of CT and DE models. Here, the state means a valuation of all variables in the model. Each state is associated with a point in time.

The solver typically computes the behavior trace iteratively, by taking the current state and inputs to the model, and estimating the state and outputs at the next timepoint.

The computation of the complete initial state of the system is called initialization. For example, if the user defines the (incomplete) initial state of the LFR model as moving at $1m/s$ in a straight line, the solver has to determine

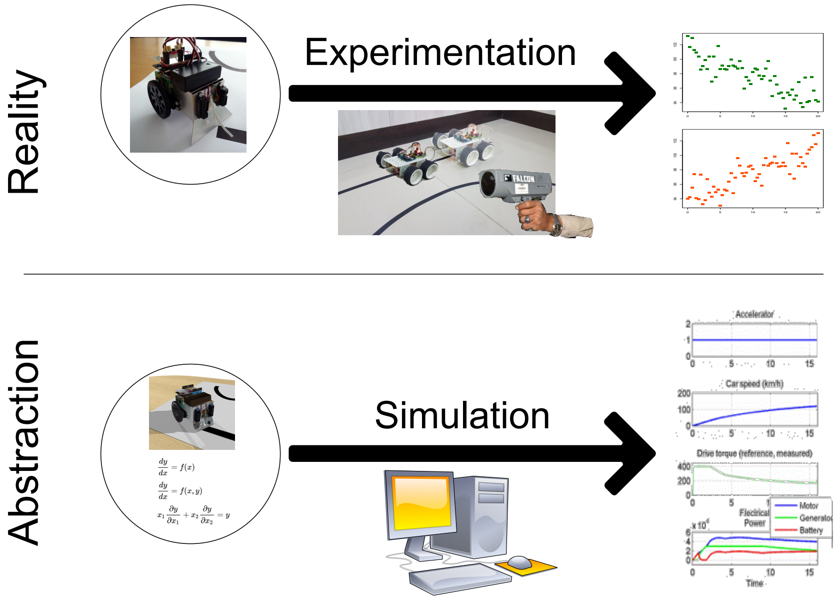


Figure 3: Illustration of simulation concepts.

what the angular velocity of the wheels is, before being able to compute the next state of the LFR model.

Since the solver often makes approximations of the behavior trace, we need a measure of how well the behavior traces produced by the solver correspond to the idealized behavior traces of the model. We denote such a measure as accuracy. Typically, the farther apart (in time) the states estimated by the solver are, the less accurate the solver is.

A variable time-step size solver will adaptively vary the time-step, the time distance between estimated states, so as to keep a prescribed accuracy. For example, in the LFR simulation, a variable time-step size solver will use smaller time-steps when the robot is moving faster, and larger time steps otherwise.

In real-time simulation, the solver computes the state at time $t$ roughly after $t$ seconds (in wall-clock time) have passed since the beginning of the simulation. For example, a 3D animation of the LFR will look more realistic if done in real-time.

Since simulation can be used to automate the evaluation of a given model, it enables the possibility of evaluation with many variations of a base model, in order to select the best design based on a given objective. To the generation of such variations, and the evaluations of each one, we call Design-Space Exploration (DSE). The design space is the set of possible solutions for a given design problem [18]. For example, in the LFR:

- the objective might be to minimize the time required to traverse a given map, energy consumption, or maximum deviation from the line;
- the design space represents the possible configurations of sensor placement and controller implementation.

Figure 4 illustrates the path followed by two different sensor configurations and the same controller.

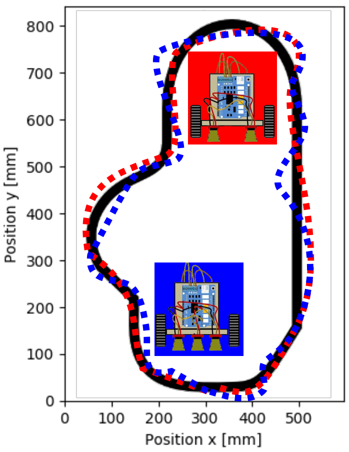


Figure 4: Illustration of design space exploration on the LFR.

### 2.3. Co-simulation Concepts

The need for co-simulation arises because the original system is composed of different subsystems, each pertaining to a specialized domain, and the system environment. For example, in fig. 5, we have highlighted two subsystems: the electro mechanical components, and the software execution. The environment is the sheet printed with a line. Each of these subsystems, and the environment, can be modelled by abstractions. For example, the electro-mechanical components can be modelled by differential equations, the software execution by a state machine, and the environment by a table mapping a position to an intensity of reflected light.

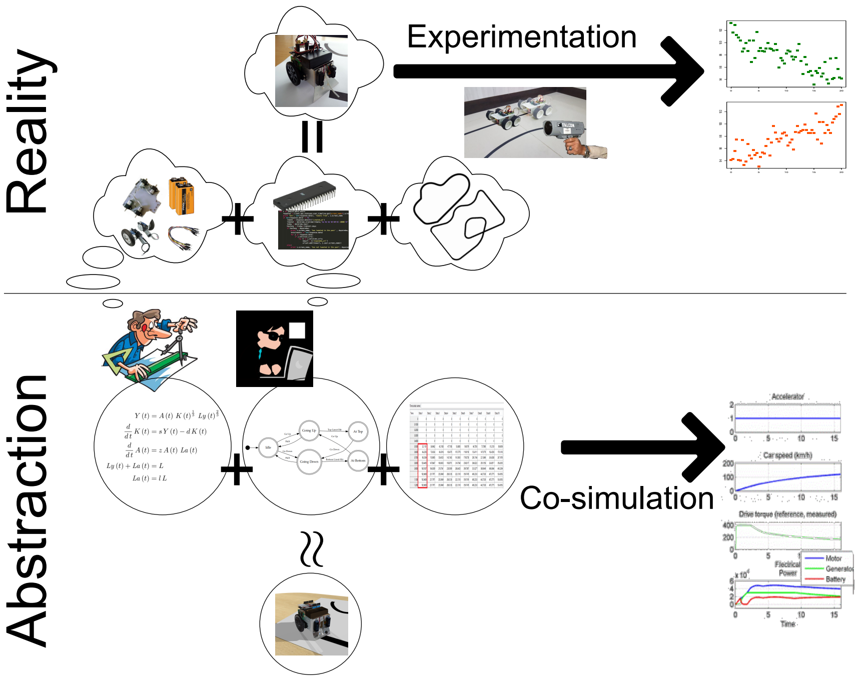

Reality
Experimentation
Abstraction
Co-simulation
$Y(t) = A(t)\,K(t)^{\frac{1}{3}}\,Ly(t)^{\frac{2}{3}}$
$\frac{d}{dt}K(t) = s\,Y(t) - d\,K(t)$
$\frac{d}{dt}A(t) = z\,A(t)\,La(t)$
$Ly(t) + La(t) = L$
$La(t) = l\,L$
Accelerator
Car speed (km/h)
Drive torque (reference, measured)
Electrical Power
Motor
Generator
Battery
Time

Figure 5: Illustration of co-simulation concepts.

Models can be built by different people, specialized in the domain of the subsystem being modelled. For example, in fig. 5 a mechanical engineer might build differential equations to model the movement of the robot, whereas a software engineer may build a state machine to model the software execution. We denote the term *coupled model* to be the combination of these models.

We assume that, if the inputs of each subsystem model are known, then its behavior can be computed. However, the subsystems interact, that is, the input of each subsystem is the output of some other subsystem, modelled in a different abstraction. This means that, to compute the coupled model behavior, all subsystem models need to be simulated in tandem.

We denote co-simulation to the act of computing the subsystem model behaviors in tandem, so as to obtain the coupled model behavior. A typical co-simulation algorithm will compute each subsystem model state and output at some time, set these as inputs to the other relevant subsystems, and repeat the process for the next time point. We denote such an algorithm by the term *orchestrator*, or *master*. We will denote by *co-simulation framework* the tool that implements a master and other related features such as visualization of results.

In order to ensure good separation of concerns, the master typically delegates the responsibility of computing each subsystem model behavior to a dedicated solver. In the example of fig. 5, the differential equations are simulated by a numerical solver, whereas the state machine is simulated by a state machine solver. Therefore, the master's responsibility is to set inputs, get outputs, and coordinate the execution of the simulators.

One example master, applied to the LFR, is illustrated in fig. 6. As the figure indicates, at simulated time $t_i$, the master requests the outputs of the body subsystem (the electro-mechanical subsystem) computed by the numerical simulator and provides these as inputs to the DE software simulator (represented by $u(t_i)$. Then the master does the same for the software simulator. After both simulators exchanged inputs/outputs, the master asks them to compute their outputs at the next time point $t_i + H$, and so on.

For other master algorithms, include those that minimize the synchronization error at communication points and solve algebraic loops between FMUs, we refer the reader to [19].

Since co-simulation is just a special class of simulation, the same concepts introduced in section 2.2 can be extended to co-simulation, with the following highlighted differences:

**Continuous Time/Discrete Event/Hybrid Coupled Model** However, often *the coupled model is Hybrid.* We denote hybrid co-simulation when the master is applied to a hybrid coupled model, i.e. consists of a DE and CT model. One of the challenges in producing accurate hybrid co-simulation results is on how to ensure that all simulators are synchronized at the point

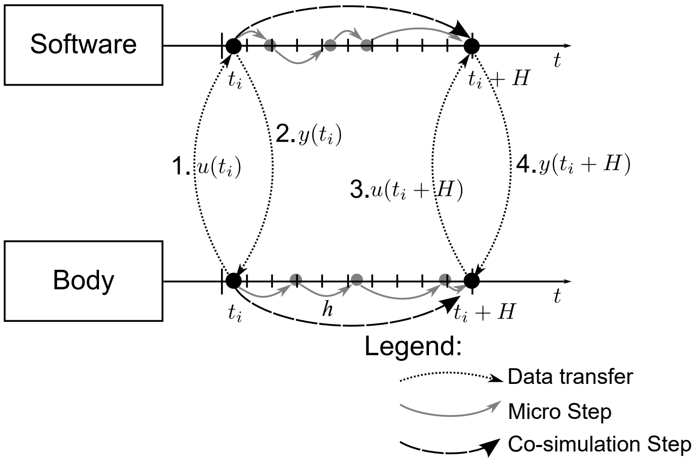


Figure 6: Example master: Jacobi [20].

of an event. Solutions to these typically involve retrying co-simulation steps to find out the correct event time [21, Section 5.1].

**Initialization** is the process performed by *the master and the simulators* to compute a complete initial state of the coupled model.

**Variable step** is a *kind of master* that adapts the communication step size ($H$ in the example of fig. 6). The simulators can too adapt their internal step size ($h$ in fig. 6). Because in general the master has little knowledge about how the simulators of each subsystem model work, providing a good variable step master requires some fine tunning. This is where the domain knowledge of the system engineer is important. In the LFR, a poorly chosen step size can cause the software controller to cross the line without being aware that it did so.

Moreover, the decoupling between the solvers of each subsystem model enables the inclusion of physical subsystems that interact with the other subsystem simulators. For example, in the LFR, one could run a co-simulation where a microprocessor running the software controller is connected to a body and environment simulators. Through the appropriate interfaces, the master will coordinate the computation of the body and environment simulators, and set/get data to/from the software execution simulator. This is called hardware-in-the-loop co-simulation.

Co-simulation has been applied extensively, and using many different simulators [3]. It is important that the same master is independent of the simulators being used to run a co-simulation. To achieve this, the community has proposed a standard for the simulator interface—the FMI standard.

The FMI standard is a tool independent standard for the exchange of models and co-simulation. It is the result of the ITEA2 European Project called MODELISAR [22]. 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, referred to as an Functional Mock-up Unit (FMU), is essentially a zip file containing a binary library that can be loaded by a co-simulation framework. The static description file contains information on inputs, outputs and parameters of an FMU including whether an output is dependant on an input. Furthermore, it informs whether it is possible to retrieve and set a state on the FMU: important capabilities for hybrid co-simulation, which can be used to perform *rollback* of an FMU by setting it to a previous state.

## 3. The Maestro Co-Simulation Framework

Maestro is a co-simulation framework that supports hybrid, distributed, and parallel, co-simulation, with accuracy and stability control mechanisms, and real-time support. These features, and corresponding rationale, are detailed in this section, while their applicability is discussed in the next section.

At the technical level, Maestro offers a RESTful web service Application Programming Interface (API), implemented in a combination of Java, Scala, and C, and available on all major platforms.

To aid in the presentation of Maestro, we will make use of parts from the 4+1 View Model of Architecture [23]. The developmental view of Maestro is depicted in fig. 7. It is split into two main projects: the *FMI Interfacing* contains the logic required to interface with FMUs; and the *Orchestration Engine* contains co-simulation related logic, including data exchange through the web service (HTTP Server), parsing and validation of the information regarding the simulators (ModelDescriptionParser and ModelDefinitionChecker), and the master (COE).

The architecture presented in fig. 7 separates the master from the interfaces required to communicate with the simulators. It reflects some of the lessons learned during the INTO-CPS project: the FMI standard currently lacks some features that are useful to fully support hybrid co-simulation (see, e.g., [24]); and the co-simulation algorithm encodes knowledge that may outlive the interfacing standards. Technically, the object facilitating the communication with individual FMUs is constructed using the factory Pattern [25]. This separates the construction logic of communication objects from the master, thereby making it possible to alter the construction of communication objects, or add other objects communicating in a different fashion, without making larger changes to the architecture. The usage of the factory pattern is demonstrated in subsection 3.2 and subsection 3.3.

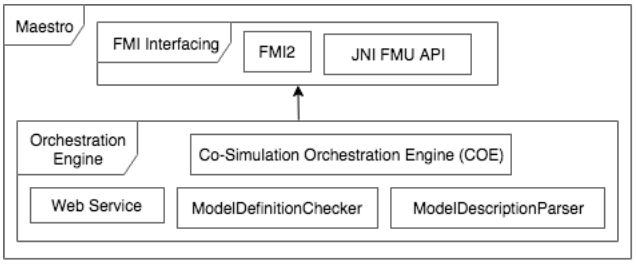


Figure 7: Developmental View of Maestro

Figure 8 shows the typical requests required to execute a co-simulation using Maestro[4].

*Required information.* The configuration of the co-simulation is transferred to Maestro as a JavaScript Object Notation (JSON) configuration. One such con-

[4] The reader is referred to [10] and `https://git.io/fNpaq` for more information on these requests and the API.

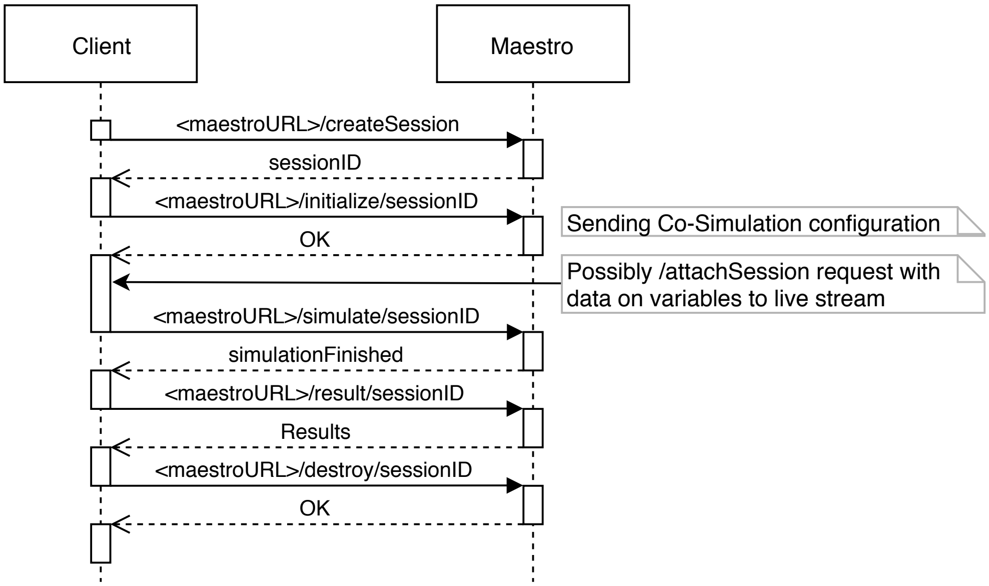

Client
Maestro
<maestroURL>/createSession
sessionID
<maestroURL>/initialize/sessionID
Sending Co-Simulation configuration
OK
Possibly /attachSession request with data on variables to live stream
<maestroURL>/simulate/sessionID
simulationFinished
<maestroURL>/result/sessionID
Results
<maestroURL>/destroy/sessionID
OK

Figure 8: Running a co-simulation with Maestro.

figuration can be seen in listing 1. The configuration includes the FMUs, FMU and co-simulation parameters (e.g., end time), input/output couplings, the master to be used, and its configuration. See [10] and `https://git.io/fNpaq` for additional details on the configuration.

```
{ "fmus":{
    "{x1}":"watertankcontroller-c.fmu",
    "{x2}":"watertank-c.fmu"
  },
  "connections":{
    "{x1}.controller.valve":["{x2}.tank.valve"],
    "{x2}.tank.level":["{x1}.controller.level"]
  },
  "parameters":{
    "{x1}.controller.maxLevel":8,
    "{x1}.controller.minLevel":2
  },
  "algorithm": {
    "type":"var-step",
    "size":[1E-10, 1.0],
    "initsize":1E-4, "constraints":{
      "maxstepsize": {
        "type": "fmumaxstepsize"}}}}
```

Listing 1: Example of a co-simulation configuration in JSON. This example contains two FMUs, their connections, initial parameters and the master to use.

The upcoming paragraphs detail the internal behavior of the Maestro, during the interactions depicted in fig. 8. The behavior is summarized in fig. 9.

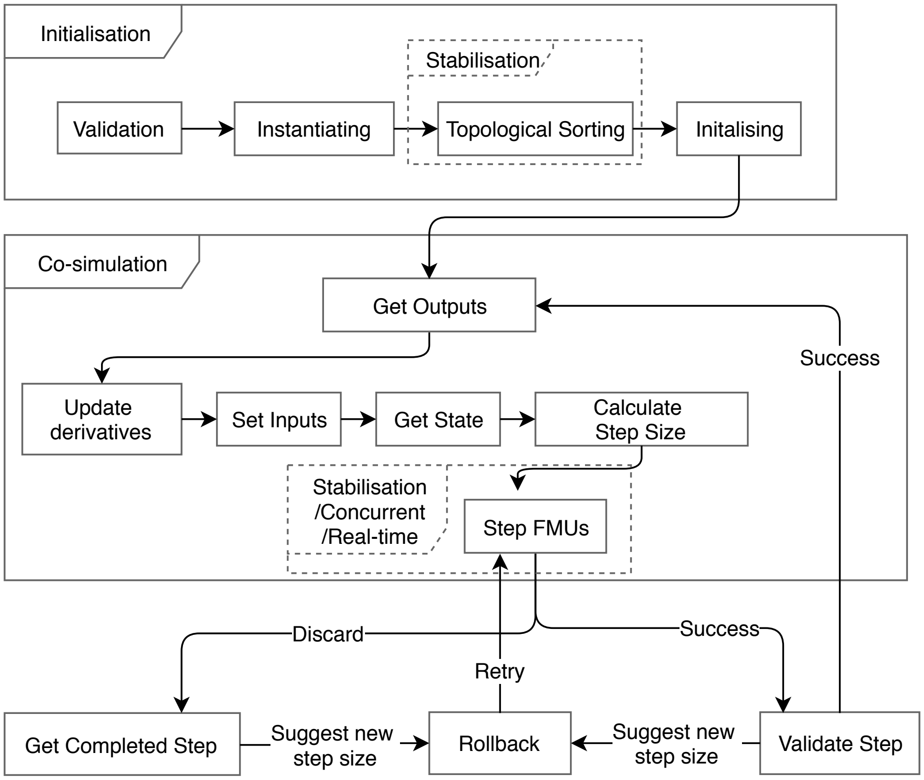


Figure 9: The Logical view of Maestro

*Initialisation.* The initialisation phase tries to ensure that the FMUs and configuration of the co-simulation are valid. Experience has shown that FMU providers do not always correctly implement the FMI standard. As such, the validation of each FMU comprises checking for the existence of compatible binary files and sound input/output descriptions, and trying to load/initialize the FMU. This procedure, which heavily depends on the FMI specification, is implemented and validated in VDM-SL [26]. This allows us to easily extend it in future standard specifications.

The initialization of the input/output couplings is done by mapping them onto a graph, where variables map to nodes and dependencies map to edges, and using a topological sorting algorithm, to determine the order of data exchange. This resulting order is used to synchronize the FMUs inputs/outputs (recall step 1 in fig. 6).

*Co-Simulation.* Overall, the iteration method used by Maestro is based on the Jacobi Iteration approach as presented in section 2. This method was chosen because it allows the FMU to be run concurrently. The entities "Update derivatives", "Validate Step", and "Calculate Step Size" in Figure 9 are related to variable step size and described in section 3.1.

As shown in fig. 9, when an FMU discards a co-simulation step of size $H$, it is prompted for the how much of the step, $H'$, it was able to complete. All the other FMUs participating in the co-simulation are then rolled back to the previous state and prompted to perform a step of size $H'$. If a fixed step size has been set, Maestro will attempt to perform the next step with the configured step size.

If the stabilization configuration is enabled, the master will repeat the same co-simulation step multiple times, in order to ensure a more accurate data exchange between the FMUs, thereby avoiding instabilities [27]. In particular, this is realised by simulating the FMUs to time $t_{i+1}$ retrieving outputs, rolling them back to time $t_i$ and stepping them again, but using the outputs from time $t_{i+1}$ as inputs. This continues until the outputs from two consecutive steps converge according to $|a - b| \leq atol + rtol \times |b|$, where $atol$ is absolute tolerance, $rtol$ is relative tolerance, $a$ is an output variable at the current state and $b$ is the corresponding output value at the previous state. $atol$ and $rtol$ are configuration parameters.

If real-time constraints are set, the master will impose a delay at the end of a co-simulation step to slow down the simulation to real-time.

The following sections delve deeper into more advanced features of Maestro: variable communication step size, hierarchical, and multi-platform co-simulation.

### *3.1. Variable Step*

Traditional variable step size simulation algorithms allow the user to control the local error made during the co-simulation, under the assumption that controlling the local error allows the global error to be controlled. Roughly, for this assumption to hold, one needs at least to ensure that the co-simulation is numerically stable, and that the coupled model forms a Lipschitz continuous ODE [28]. In practice, most of the industrial case studies where Maestro has been applied, do not obey the Lipschitz continuity, because they include discontinuities. As such, the variable step size functionality needs to go beyond the traditional step size algorithms.

The component responsible for computing the communication step size during a co-simulation is the Variable Step size Calculator (VSC). It does so based on: the current time, the previous step size, the current output values, the output derivatives of the FMUs, the current local error estimate [3, Section 4.3.4], and co-simulation specific constraints. If derivatives are not provided by the FMUs, then they are estimated.

These constraints can be specified by the user, and reflect the experience collected from the industrial case studies:

**Zero Crossing:** A zero crossing constraint instructs the VSC to synchronize all FMUs at a time where a given signal is zero or two signals intersect. This constraint is useful in the co-simulation of hybrid systems (e.g., collisions). Maestro avoids the use of rollback functionalities by using the derivatives of the signal to estimate when it will cross the zero. This works well because in practice most FMUs providers do not support rollback functionalities.

**Bounded Difference:** A bounded difference constraint ensures that the difference between signals or two consecutive observations of the same signal is bounded by a pre-defined value. This constraint can be used, e.g., to assure that a simulator does not differ on its input by more than the prescribed value, or to support quantization [29, 30]. The underlying assumption is that the signal being observed is continuous, that is, the smaller the observation interval (communication step size), the smaller the difference between two consecutive observations. Concrete examples of this feature include bounding the error on the computation of heatflow between two simulators where each of them only have a prediction of the flow from the other model.

**Sampling Rate:** A sampling rate constraint makes sure all FMUs synchronize at pre-determined time points. This can be used in co-simulation that include software models that are supposed to run at some frequency. Note that it does not force all communication step sizes to be of a fixed step size: it forces the synchronization at those times, but other synchronizations outside those times can happen.

**FMU Max Step Size:** This constraint, well known in discrete event co-simulation, and first proposed as an FMI extension `getMaxStepSize` in [31], aims at avoiding the need to rollback the co-simulation. The constraint requires each FMU to implement a function `getMaxStepSize`, which returns the largest possible step it can perform at the given point in time. The VSC then chooses the minimum of the step sizes proposed by all FMUs.

When multiple constraints have been specified, the chosen step size will be the minimum of the step sizes satisfying the constraints. However, there are cases that need special attention. To demonstrate these, we present two scenarios in the following paragraphs.

Consider the case where constraint A is activated and increases the step size by a factor $\rho_A$. This is referred to as relaxing the step size and $\rho_A$ is the relaxation factor. This might activate constraint B, because constraint B cannot relax the step size by more than factor $\rho_B < \rho_A$. Thus, constraint B is activated, not because of a direct violation, but because it cannot relax the step size as much as constraint A. To prevent this case, the constraints have the same maximum relaxation factor. Thus, this factor is not a property of the individual constraint, but of the VSC.

Now consider the second scenario: a discrete constraint entails a large reduction in the step size, while the other constraints tolerate a larger step size. In the ensuing co-simulation steps, there might not be any discrete events, but the step size will not be increase immediately. This is addressed in different ways for the different constraints. More details are available in [32, Sec. B.7]

After a step has completed, the VSC is invoked to validate whether any of the constraints have been violated. If that is the case, a rollback is initiated and the VSC provides a new and reduced step size. More details can be found in [10] and `https://git.io/fNpaq`.

### *3.2. Hierarchical Co-Simulation*

Maestro has been extended with a feature enabling the concept of Hierarchical co-simulation, demonstrated at `https://youtu.be/MKZb3HkyVtc`, and conceptually illustrated in fig. 10.

In the figure, FMU A and B, the connections between them and the co-simulation settings are *imploded* into one FMU, thereby making this FMU a fully configured co-simulation. The resulting FMU can be then used in a co-simulation.

The hierarchical co-simulation feature is motivated by two scenarios, encountered in some of the industrial cases of the INTO-CPS project: the co-simulation includes FMUs that are *tightly coupled* (that is, they need to synchronize often), and the interaction between suppliers and system integrators demand a more efficient way of reproducing co-simulation results. Additionally, recent work has

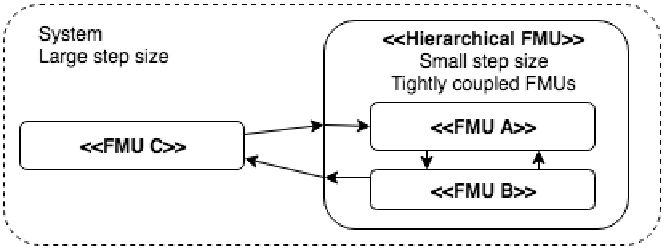


Figure 10: Hierarchical co-simulation

made use of this feature to change the FMU capabilities [33].

To illustrate the first scenario, consider fig. 10. Since FMUs A and B need to communicate often, Maestro will synchronize all other FMUs at the same rate. This is clearly wasteful. With hierarchical co-simulation, FMUs A and B can be turned into the same FMU and synchronize independently of the other FMUs.

The second scenario is illustrated in fig. 11: System A is being developed, and this system uses subsystem B from another company, which uses most of its components from third-parties. The supplier of subsystem B uses the FMUs provided by third-party suppliers to evaluate subsystem B. However, the developers of system A have no easy way of obtaining the co-simulation parameters required to correctly reproduce the co-simulations run by the provider of subsystem B. Indeed, the developers of system A might not even have the knowledge to adequately configure the co-simulation. Thus, with hierarchical co-simulation, the co-simulation of the subsystem can be configured by the supplier of subsystem B and transferred.

Our experience is corroborated by a recent empirical survey [6], where the configuration of the co-simulation parameters was identified as one of the challenges faced by the practitioners. This is aggravated by the fact that Maestro provides more flexibility in specifying how the synchronization of the FMUs should be performed (recall section 3.1). These issues are mitigated if subsystem B is turned into a hierarchical FMU.

The realisation of hierarchical FMU revolves around the fact that the factory

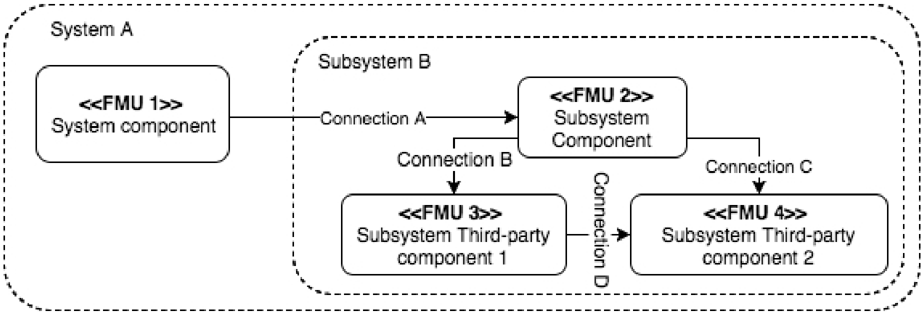


Figure 11: System with subsystem that uses third-party components

pattern, mentioned in section 3, is employed to not only create FMU communication objects, but also to create master objects. The information on which type of FMU to construct is contained within the `fmus` object of a co-simulation configuration. For example, `"watertank-c.fmu"` in listing 1 on line 3 refers to the default FMU, whereas the alternative `"coe:hierarchicalWaterTank"` refers to a hierarchical FMU as it begins `coe:`. The logical view of the hierarchical co-simulation feature is presented in fig. 12. In this figure, the `HierarchicalCOE` and the `COE` entities have the same type, but are constructed differently. The static information file of the `HierarchicalFMU` is the union of the static information files of the inner FMUs, namely `FMU A` and `FMU B`. Furthermore, the file contains a mapping that allows the `HierarchicalFMU` to distinguish between scalar variables, such that they remain unique in the resulting static information file.

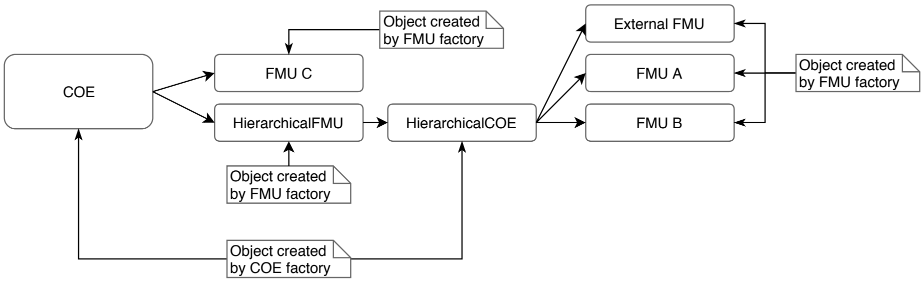


Figure 12: Logical View of the Hierarchical Feature

In fig. 12, the communication between inner FMUs and external FMUs from the view of the hierarchical COE is carried out using an entity representing the external ports, denoted `External FMU`. When the Hierarchical FMU is asked to perform a co-simulation step, a barrier synchronization mechanism is used to ensure that all inner FMUs complete the co-simulation step. This is illustrated in fig. 13, where `SC` and `SW` denote the two semaphores used.

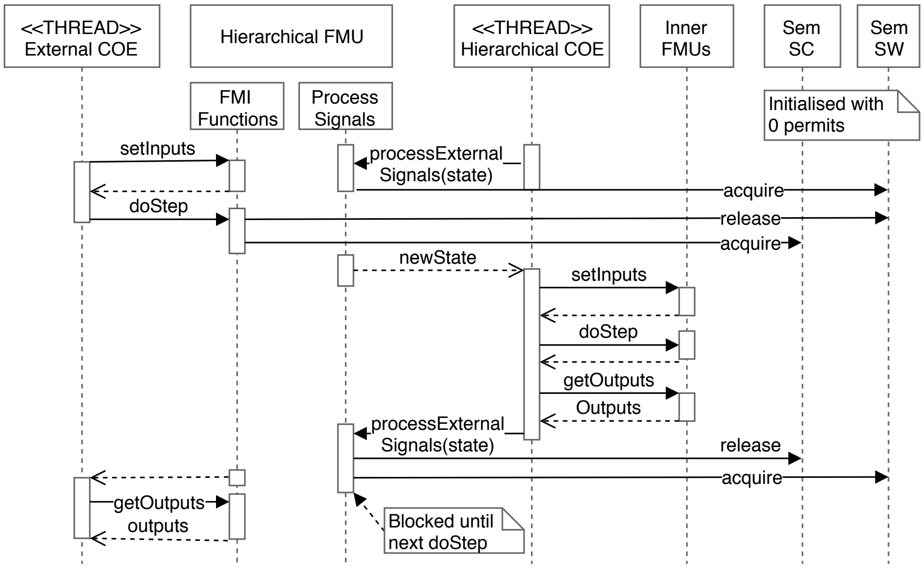


Figure 13: Sequence diagram of synchronisation in relation to hierarchical co-simulation.

### 3.3. Distributed Co-Simulation

The FMI standard enables models to be shared as FMUs. In practice, however, because the tools used to model and simulate are heterogeneous, we found that implementing the FMI standard was insufficient. The major problems concerned the platforms and architectures supported by the FMUs. A co-simulation can only be run if all FMUs support the same combination of platform and architecture.

To overcome this challenge, the distributed co-simulation extension was provided. This feature is illustrated in fig. 14 and also makes use of the factory pattern to create FMUs as was carried out in context of hierarchical co-simulation

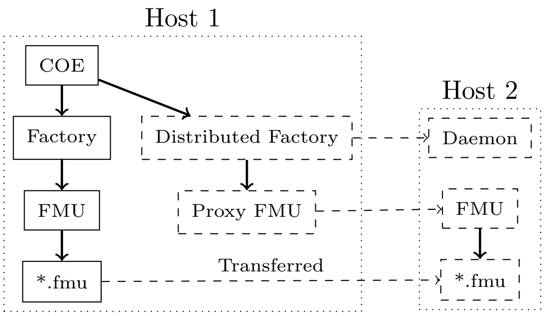


Figure 14: Distributed Extension Overview

in section 3.2. In the figure, during the co-simulation, the FMU that is not compatible with host 1 is replaced by a compatible proxy FMU. The proxy FMU communicates with host 2, where the actual FMU has been transferred to. A daemon ensures the correct transfer and initialization of the actual FMU in host 2.

## 4. Validation, Applicability and Lessons Learned

This section discusses the lessons learned during the development of Maestro and its application to industrial case studies. Maestro has been applied extensively, and it is outside the scope of this work to detail every use case where it has been applied. For some published examples, refer to [34, 35, 36, 37, 38, 39, 40, 41, 42, 13, 43].

The features described in this work where developed in response to the specific needs of the case studies. As a result, Maestro has become quite usable and robust. This is corroborated by anecdotes from the companies using Maestro:

- "Throughout year 2 of INTO-CPS, the COE has become more stable, which improved usability", TWT GmbH [39];
- "... co-simulation with the INTO-CPS COE was a very simple task . . . The simulation is fast and gives immediate results.", ClearSy [39].

The maturity of Maestro is also demonstrated on the FMI Tools page[5], where Maestro has passed cross-check using 183 FMUs exported by other tools across the supported platforms. Furthermore, it is the only tool that has been successfully cross-checked on Windows, Linux on Mac across architectures. A successful cross-check means that a master has successfully imported and simulated an FMU exported from another tool.

The remaining part of this section describes how Maestro's features where used and lessons learned concerning the following areas: API, Feedback, Extensibility, Distributed Co-Simulation, Real-time Simulation and Variable step. It concludes by presenting how Maestro and the INTO-CPS project address some of the challenges in the adoption of FMI, identified in [6].

*API.* Maestro is accessible via a RESTful web service API because the contexts in which it has been used (e.g., DSE [44], and from a Graphical User Interface (GUI) [45]) require a programming-language agnostic interaction. Furthermore, the API is kept minimalistic as experience has shown that some researchers are not used to writing code that interacts with web services.

The API of Maestro also allows for different simultaneous co-simulations by using a session mechanism. This was utilised by TWT GmbH to create co-simulations from an FMU taking part in another co-simulation, as depicted in Figure 15 [42]. The role of the *sub-co-simulation* is to provide a realistic remaining state-of-charge of the vehicle's battery during and at the end of a trip. 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 the sub-co-simulation runs to completion.

*Rapid Feedback.* Even though Maestro does not require a GUI, the INTO-CPS Application[6] was crucial to obtain rapid feedback on the use of Maestro. One

[5] https://fmi-standard.org/tools/, visited November 7, 2018

[6] Available at the INTO-CPS Association Github: `https://github.com/INTO-CPS-Association/`.

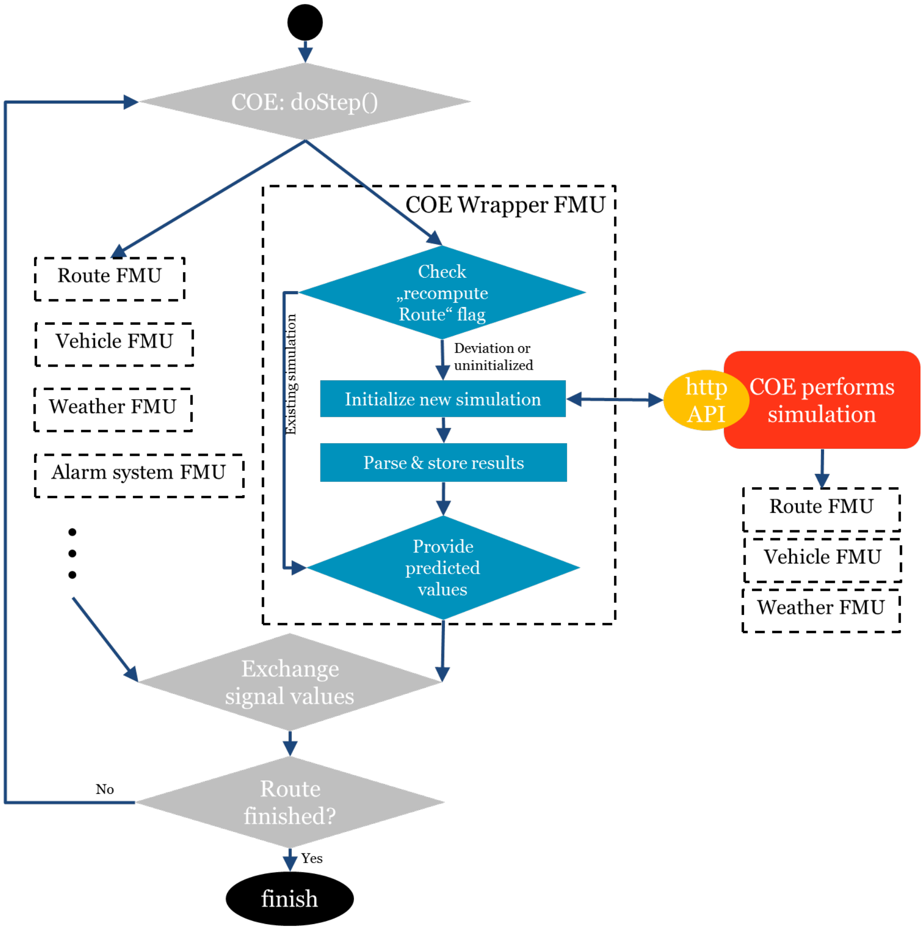


Figure 15: Trip Assistant Case from TWT GmbH

of the results of such feedback is the live streaming feature of Maestro. This feature, which consists of making available the co-simulation results as they are being computed, allows engineers to quickly spot whether the coupled model is behaving as intended, or whether something has gone wrong with the co-simulation. This saved time in long running co-simulations (e.g., one of the companies had co-simulations that took at least two weeks to complete). The live streaming feature was also used during DSE, where it was possible to evaluate the fitness of a particular coupled model without waiting for the co-simulation to end. Currently, this feature does not allow setting external inputs during a co-simulation.

*Extensibility.* The architecture of Maestro allowed multiple extensions to be developed with minimal changes and decouples the co-simulation logic from the interaction with the FMUs. This extensibility will be put to the test with the upcoming versions of FMI. This subject is extended upon in section 6.

*Distributed Co-Simulation.* This feature was applied to solve a problem faced by MAN Diesel & Turbo [16]. They were developing a control system for a water handling system, used to clean the exhaust gas using exhaust gas recirculation for a large two-stroke maritime combustion engine. 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. Furthermore, the distribution feature also mitigates the problem that some models could only be executed on a local computer with a specific environment (due to licensing).

*Real-time Simulation.* This feature was motivated in the co-simulation of agricultural robots. Maestro was used to test and develop an interface for a robot with real-time components [13, 43].

*Variable Step.* Variable Step has been applied to increase simulation speed, improve the precision of co-simulation results and support DE models. Moreover, the bounded difference constraint was applied to bound the error in the co-simulation of the heat flow between FMUs. For more details see [10].

*Teaching Advanced Features.* It was observed that most applications of Maestro use fixed step rather than variable step, although variable step has the potential to improve performance or accuracy. When asked why, Industry Partners explained that they believe in the potential of variable step, but there is a challenge of convincing users to actually use it. Furthermore, it was reported that the behaviour in terms of time steps can be different than expected. They propose tutorials that show how to use it, under which conditions to use it, which option is best suited for which use case, and what can be gained from it. Furthermore, they propose tooling to provide suggestions for the parameters of the co-simulation, as is done in [46]. The INTO-CPS project reacted to these queries by publishing step-by-step tutorials, a user manual, and several examples. This subject is touched upon again in section 6.

*FMI Barriers.* To conclude this section we will relate the development of Maestro and the INTO-CPS technology to the barriers of FMI uncovered by the aforementioned empirical survey [6]. Many of the same barriers were noted by researchers involved in INTO-CPS project, who also contributed to the development of the standard by relaying information and examples at an FMI user meeting[7] [47]. Below, the relation is carried out for several barriers

**FMI has limited support for discrete co-simulation and it is not easily applicable:**

> The public example "Ether" simulates a network protocol using strings and booleans [48] and is also applied in co-simulating a swarm of UAVs [37]. The supported Get Max Step Size [31] extension allows a DE FMU to convey its preferred step size [49].

**The standard does not support certain requirements that would be widely needed by industry and academia:**

> The extension Get Max Step Size was implemented by Maestro. Furthermore, it was reported at an FMI user meeting that compilation information is missing for source-code FMUs and functionality to retrieving additional resources of interest generated by FMUs during a co-simulation is lacking, e.g. to perform model checking.

**There is insufficient documentation and a lack of examples, tutorials, etc.:**

> Maestro and INTO-CPS offer regularly updated resources on the INTO-CPS Association's Github page: step-by-step tutorial, several examples

[7] Related 1-page document available on request.

[48], step-by-step user manual [50], guidelines on using the INTO-CPS technology [51], protocol and more. The INTO-CPS Application makes it simple to use the technology by having an examples and tools download manager and can be used to configure and launch a co-simulation using Maestro.

**Lack of transparency in features supported by FMI tools:**

The features of Maestro are clearly described in the user manual [50].

**FMI has limited support for hybrid co-simulation and it is not easily applicable:**

This information was also unearthed as part of applying Maestro and INTO-CPS technology to various challenges as presented at the FMI User Meeting and in [35]. The supported FMI extension Get Max Step Size mitigates part of this challenge.

**There is a lack of (scientific) community, forums, groups:**

The INTO-CPS Association has been formed to offer this.

**There is a lack of tools that sufficiently support FMI:**

Maestro is one such tool and also has an associated GUI called The INTO-CPS Application.

**There is not enough cooperation and exchange (theoretical/numerical, implementation, application/industry) in defining and developing the FMI standard:**

Maestro (and INTO-CPS) is a result of academia and industry working together to produce tools, methodologies, theoretical novelties and more. More specifically, both Aarhus Univerity and TWT GmbH contributed to development of Maestro. The 48 publications[8], the industrial follower group of 79 members[9], the INTO-CPS Association[10], and the attendance

[8] `http://into-cps.org/publications/`, visited on November 7, 2018.

[9] `http://projects.au.dk/into-cps/industry/industry-follower-group/`, visited on November 7, 2018.

[10] `http://into-cps.org`, visited on November 7, 2018

at the FMI User Meeting and the reporting of an issue[11] bears witness of cooperation and exchange of knowledge.

## 5. Related Work

Co-simulation is a large field that is difficult to do justice in this section. See [2, 4, 5, 52] for some surveys on the topic. In this section, we focus on co-simulation tools that represent the diversity of FMI co-simulation frameworks and share the same goal as Maestro. These are summarized in table 1.

Several of the summarized tools are proprietary, require writing source code using a particular framework, or are not standalone but integrated into different environments, e.g. Eclipse. Maestro, compared to these, is a standalone orchestrator, 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 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 orchestrator on the FMI Tools page that has been crosschecked for Linux, Windows, and Mac across architectures[12].

[11] `https://trac.fmi-standard.org/ticket/338`, visited on November 7, 2018.

[12] The results as of November 7, 2018. It is called the INTO-CPS Orchestration Engine on this website: `https://fmi-standard.org/tools/`.

| **Product** | **Description** |
|---|---|
| DACCOSIM [53] | 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 crosscheck results; Multithreaded and distributed architecture; Approximates discrete events [54], e.g. events in DACCOSIM occurs in a time interval, but these are to be considered instantaneous [55]. 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 [54]. |
| FIDE [54] | IDE for building applications using FMUs (Maestro is standalone); open-source; cross-platform; Based on the open-source Ptolemy II framework [56] for analysis and design of heterogeneous systems; Supports many extensions to FMI 2.0 [31, 57] including the one mentioned under DACCOSIM.; FMI Tools website shows planned FMI 2.0 support[13]. |
| PyFMI[14] | 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[15]. |
| C2WT [58] | 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.

[13] Publications indicate that FMI 2.0 is supported.

[14] `https://jmodelica.org/pyfmi/index.html` Visited on November 7, 2018.

[15] `https://jmodelica.org/fmil/FMILibrary-2.0.3-htmldoc/index.html` Visited on November 7, 2018. FMILibrary is a C library serving as foundation for applications interfacing FMUs

## 6. Future Work

Since Maestro is envisioned as a stable research tool for co-simulation, improving its extensibility is one of the top priorities. One of the ways to achieve this is to support external plugins. For example, a plugin could implement a Gauss-Seidel master [59], or support an FMI extension, and so forth. Furthermore, version 3.0 of the FMI standard is expected to be released during 2019 and Maestro will be updated to support it.

It is also of interest to allow the usage of FMUs in context of other standards for simulation architectures such as High Level Architecture (HLA). Initial work has been carried out on this topic [60, 61].

Maestro will continue to be improved as it will be used in future projects, such as "Application of co-simulation to support test and operations (ACOSIM)" [62]. In this project, Maestro will be integrated within the simulation framework (Simulation Model Portability 2) used by The European Space Agency.

To increase the adoption of co-simulation in both research and industry, it is desireable to make improvements on the accessability of Maestro. Currently, the approach is to download the INTO-CPS Application, use the built-in download manager to download Maestro, and then perform co-simulations. To avoid the need for downloading and hosting it locally, we are investigating a cloud version of the INTO-CPS Application, which would allow public access to a running instance of Maestro. There are several challenges involved in this: resources, as some co-simulations are demanding; sandboxing, as FMUs execute native code; a new API with security measures to ensure that co-simulations are available to the permitted clients only; and others.

The final goal of our research is that the user simply provides the simulators and their coupling, pushes a button and gets valid results. In order to realise this, the simulators must provide additional meta-data. For example, the shortest timed reaction of a DE FMU implementing a timed automata and the propagation delay of other FMUs in the system could aid in automatically calculating the step size or algorithm to use [63]. Another example is the property of energy conservation, where the flow of energy between the coupled simulation units are considered, and energy residuals are an expression of coupling errors [64]. This requires information on the physical power quantities of coupling variables. If such meta-data was exported from the tools used to create the simulation units, it would be possible to automatically disqualify certain algorithms and qualify others. If several mutual exclusive algorithms were qualified, these could be employed in a DSE-like approach and the appropriate algorithm could be chosen. This would also help to mitigate the challenge of using variable step introduced in section 4, as the orchestrator would automatically apply the most appropriate algorithm and configuration.

## 7. Concluding Remarks

In this paper, the open source, cross-platform, co-simulation framework called Maestro 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 accurate co-simulation 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, for example, predict a future position.

The applicability of Maestro has been presented by examples, industrial case studies and academic case studies, which also served to validate Maestro's approach to co-simulation. Additionally, Maestro has passed 183 cross-checks in total, and is the only tool to have passed cross-checks on Linux, Mac and Windows across architectures[16]. Furthermore, the challenges faced and lessons learned during the development and usage of Maestro have been presented.

[16] As of November 7, 2018.

These can be summarized as follows:

- The API of an orchestrator should be language agnostic, as researchers use a variety of programming languages and might not have a software development background.
- Rapid feedback during a co-simulation is important, as errors might be exposed before the co-simulation has finished, allowing users and tools to react faster. Furthermore, it is important during tool development to support the desired features.
- Extensibility is important for future research.
- Due to simulation units supporting different platforms and architectures, it should be possible to perform a co-simulation across platforms and architectures.
- Advanced features of co-simulation, e.g. configuring variable step, should be accompanied by teaching material such as examples and tutorials.

It is our goal to turn Maestro into a tool for conducting research within the domain of co-simulation to ensure trustworthy co-simulation results. Furthermore, by providing a stable and validated tool it is a step in the direction of turning co-simulation into a technology taken for granted.

## Acknowledgements

The authors 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, and the Agency for Innovation by Science and Technology in Flanders (IWT, dossier 151067). We would like to thank all the participants of those projects for their efforts making this a reality. Furthermore, we would like to thank United Technology Research Centre Ireland for providing feedback on using the technology and Nick Battle for proofreading the paper.

## References

## References

[1] H. Vangheluwe, J. De Lara, P. J. Mosterman, An introduction to multi-paradigm modelling and simulation, in: AI, Simulation and Planning in High Autonomy Systems, SCS, 2002, pp. 9–20.

[2] C. Gomes, C. Thule, D. Broman, P. G. Larsen, H. Vangheluwe, Co-simulation: a Survey, ACM Comput. Surv. 51 (3) (2018) 49:1–49:33.

[3] C. Gomes, C. Thule, D. Broman, P. G. Larsen, H. Vangheluwe, Co-simulation: State of the art, Tech. rep. (Feb. 2017).
URL `http://arxiv.org/abs/1702.00686`

[4] I. Hafner, N. Popper, On the terminology and structuring of co-simulation methods, in: Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, ACM Press, New York, New York, USA, 2017, pp. 67–76. `doi:10.1145/3158191.3158203`.

[5] P. Palensky, A. A. Van Der Meer, C. D. Lopez, A. Joseph, K. Pan, Cosimulation of Intelligent Power Systems: Fundamentals, Software Architecture, Numerics, and Coupling, IEEE Industrial Electronics Magazine 11 (1) (2017) 34–50. `doi:10.1109/MIE.2016.2639825`.

[6] G. Schweiger, C. Gomes, G. Engel, I. Hafner, J. Schoeggl, A. Posch, T. Nouidui, Functional Mock-up Interface: An empirical survey identifies research challenges and current barriers, in: The American Modelica Conference, Cambridge, MA, USA, 2018, p. to be published.

[7] 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.

[8] 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.

[9] P. G. Larsen, J. F. andJim Woodcock, C. König, S. Basagiannis, E. Brosse, C. Gomes, J. Cabral, H. D. Macedo, C. Thule, A. Sadovykh, C.-B. Zamfirescu, M. Neghina, K. Pierce, C. Gamble, R. Payne, The INtegrated TOolchain for Cyber-Physical Systems (INTO-CPS): a Guide, Tech. rep., INTO-CPS Association (October 2018).
URL `www.into-cps.org`

[10] A. Pop, V. Bandur, K. Lausdahl, M. Groothuis, T. Bokhove, Final Integration 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`

[11] 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`.

[12] 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.

[13] 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.

[14] L. D. Couto, S. Basagiannis, 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.

[15] 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.

[16] 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.

[17] 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).
URL `http://destecs.org/images/stories/Project/Deliverables/D34aExamplesCompendium.pdf`

[18] J. F. Broenink, J. Fitzgerald, C. Gamble, C. Ingram, A. Mader, J. Marincic, Y. Ni, K. Pierce, X. Zhang, Methodological guidelines 3, Tech. rep., The DESTECS Project (INFSO-ICT-248134) (October 2012).

[19] C. Gomes, C. Thule, P. G. Larsen, J. Denil, H. Vangheluwe, Co-simulation of Continuous Systems: A Tutorial, Tech. Rep. arXiv:1809.08463 [cs, math] (Sep. 2018). `arXiv:1809.08463`.
URL `http://arxiv.org/abs/1809.08463`

[20] R. Kübler, W. Schiehlen, Two Methods of Simulator Coupling, Mathematical and Computer Modelling of Dynamical Systems 6 (2) (2000) 93–113. doi:10.1076/1387-3954(200006)6:2;1-M;FT093.

[21] C. Gomes, C. Thule, D. Broman, P. G. Larsen, H. Vangheluwe, Co-simulation: A Survey, ACM Computing Surveys 51 (3) (2018) Article 49. doi:10.1145/3179993.

[22] ITEA Office Association, Itea 3 ? project ? 07006 modelisar, https://itea3.org/project/modelisar.html (December 2015).

[23] P. B. Kruchten, The 4+1 view model of architecture, IEEE Software 12 (6) (1995) 42–50. doi:10.1109/52.469759.

[24] F. Cremona, M. Lohstroh, D. Broman, E. A. Lee, M. Masin, S. Tripakis, Hybrid co-simulation: It's about time, Software & Systems Modelingdoi:10.1007/s10270-017-0633-6.

[25] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-oriented Software, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[26] 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 International Symposium on Formal Methods (FM 2016), 2016.

[27] M. Busch, B. Schweizer, Numerical stability and accuracy of different co-simulation techniques: Analytical investigations based on a 2-DOF test model, in: 1st Joint International Conference on Multibody System Dynamics, 2010, pp. 25–27.

[28] M. Arnold, C. Clauß, T. Schierz, Error Analysis and Error Estimates for Co-simulation in FMI for Model Exchange and Co-Simulation v2.0, in: S. Schöps, A. Bartel, M. Günther, W. E. J. ter Maten, C. P. Müller (Eds.), Progress in Differential-Algebraic Equations, Springer

Berlin Heidelberg, Berlin, Heidelberg, 2014, pp. 107–125. doi:10.1007/978-3-662-44926-4\\_6.

[29] J.-S. Bolduc, H. Vangheluwe, Mapping ODES to DEVS: Adaptive quantization, in: Summer Computer Simulation Conference, Society for Computer Simulation International, Montreal, Quebec, Canada, 2003, pp. 401–407.

[30] W. P. M. H. Heemels, A. R. Teel, N. van de Wouw, D. Nešić, Networked Control Systems With Communication Constraints: Tradeoffs Between Transmission Intervals, Delays and Performance, IEEE Transactions on Automatic Control 55 (8) (2010) 1781–1796. doi:10.1109/TAC.2010.2042352.

[31] 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.

[32] K. Lausdahl, P. G. Larsen, S. Wolf, V. Bandur, A. Terkelsen, M. Hasanagić, C. T. Hansen, K. Pierce, O. Kotte, A. Pop, E. Brosse, J. Brauer, O. Möller, Design of the INTO-CPS Platform, Tech. rep., INTO-CPS Deliverable, D4.1d (December 2015).

[33] C. Gomes, B. Meyers, J. Denil, C. Thule, K. Lausdahl, H. Vangheluwe, P. De Meulenaere, Semantic Adaptation for FMI Co-simulation with Hierarchical Simulators, SIMULATION (2018) 1–29doi:10.1177/0037549718759775.

[34] M. Neghina, C.-B. Zamrescu, P. G. Larsen, K. Lausdahl, K. Pierce, Multi-Paradigm Discrete-Event Modelling and Co-simulation of Cyber-Physical Systems, Studies in Informatics and Control 27 (1) (2018) 33–42.

[35] L. Diogo Couto, S. Basagiannis, E. H. Ridouane, E. Zavaglio, P. Antonante, H. Saada, S. Falleni, Lessons learned using fmi co-simulation for model-based design of cyber physical systems, in: T. Margaria, B. Steffen (Eds.),

Leveraging Applications of Formal Methods, Verification and Validation. Distributed Systems, Springer International Publishing, Cham, 2018, pp. 488–503.

[36] J. Cabral, M. Wenger, A. Zoitl, Enable co-simulation for industrial automation by an fmu exporter for iec 61499 models, in: IEEE 16th International Conference of Industrial Informatics (INDIN), 2018.

[37] G. Zervakis, K. Pierce, C. Gamble, Multi-modelling of Cooperative Swarms, in: Proceedings of the 16th Overture Workshop, 2018.

[38] F. Hantry, T. Lecomte, S. Basagiannis, C. König, J. Esparza, Case Studies 1, Public Version, Tech. rep., INTO-CPS Public Deliverable, D1.1a (December 2015).

[39] J. Ouy, T. Lecomte, M. P. Christiansen, A. V. Henriksen, O. Green, S. Hallerstede, P. G. Larsen, C. J. ger, S. Basagiannis, L. D. Couto, A. E. din Mady, H. Ridouanne, H. M. Poy, J. V. Alcala, C. König, N. Balcu, Case Studies 2, Public Version, Tech. rep., INTO-CPS Public Deliverable, D1.2a (December 2016).

[40] J. Ouy, T. Lecomte, F. F. Foldager, A. V. Henriksen, O. Green, S. Hallerstede, P. G. Larsen, L. D. Couto, P. Antonante, S. Basagiannis, S. Falleni, H. Ridouane, H. Saada, E. Zavaglio, C. König, N. Balcu, Case Studies 3, Public Version, Tech. rep., INTO-CPS Public Deliverable, D1.3a (December 2017).

[41] J. Fitzgerald, C. Gamble, M. Mansfield, J. Ouy, R. Palacin, K. Pierce, P. G. Larsen, Collaborative modelling and co-simulation for Transportation Cyber-Physical Systems, Elsevier, 2018, Ch. 3, pp. 51–79.

[42] C. F. J. König, G. Meisl, N. Balcu, B. Vosseler, H. Hörmann, J. Höll, V. Fäßler, Engineering of cyber-physical systems in the automotive context: Case study of a range prediction assistant, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and

Validation. Distributed Systems, Springer International Publishing, Cham, 2018, pp. 461–476.

[43] F. Foldager, O. Balling, C. Gamble, P. G. Larsen, M. Boel, O. Green, Design Space Exploration in the Development of Agricultural Robots, in: AgEng conference, Wageningen, The Netherlands, 2018.

[44] C. Gamble, Comprehensive DSE Support, Tech. rep., INTO-CPS Deliverable, D5.3e (December 2017).

[45] C. Gamble, O. Möller, V. Bandur, Integration of Tool Chain Extension Modules with the COE, Tech. rep., INTO-CPS Deliverable, D5.3a (December 2017).

[46] M. Benedikt, F. R. Holzinger, Automated configuration for non-iterative co-simulation, in: 17th International Conference on Thermal, Mechanical and Multi-Physics Simulation and Experiments in Microelectronics and Microsystems (EuroSimE), IEEE, Montpellier, 2016, pp. 1–7. doi:10.1109/EuroSimE.2016.7463355.

[47] K. Lausdahl, C. Thule, P. Larsen, J. Höll, C. König, A. Klueber, M. Pfeil, 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

[48] M. Mansfield, C. Gamble, K. Pierce, J. Fitzgerald, S. Foster, C. Thule, R. Nilsson, Examples Compendium 3, Tech. rep., INTO-CPS Deliverable, D3.6 (December 2017).

[49] K. L. Casper Thule, P. G. Larsen, Overture FMU: Export VDM-RT Models as Tool-Wrapper FMUs, in: Proceedings of the 16th Overture Workshop, 2018.

[50] V. Bandur, P. G. Larsen, K. Lausdahl, C. Thule, A. F. Terkelsen, C. Gamble, A. Pop, E. Brosse, J. Brauer, F. Lapschies, M. Groothuis, C. Kleijn, L. D. Couto, INTO-CPS Tool Chain User Manual, Tech. rep., INTO-CPS Deliverable, D4.3a (December 2017).

[51] J. Fitzgerald, C. Gamble, K. Pierce, Method Guidelines 3, Tech. rep., INTO-CPS Deliverable, D3.3a (December 2017).
URL `http://into-cps.org/fileadmin/into-cps.org/Filer/D3.3a_Method_Guidelines_3.pdf`

[52] C. Gomes, C. Thule, D. Broman, P. G. Larsen, H. Vangheluwe, Co-simulation: State of the art, CoRR abs/1702.00686. `arXiv:1702.00686`.
URL `http://arxiv.org/abs/1702.00686`

[53] 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 Multi-Conference, Society for Computer Simulation International, Alexandria, Virginia, USA, 2015, pp. 804–811.

[54] F. Cremona, M. Lohstroh, S. Tripakis, C. Brooks, E. A. Lee, FIDE – An FMI Integrated Development Environment, in: Symposium on Applied Computing, 2015.

[55] 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.

[56] C. Ptolemaeus (Ed.), System Design, Modeling, and Simulation using Ptolemy II, Ptolemy.org, 2014.
URL `http://ptolemy.org/books/Systems`

[57] 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.
URL https://doi.org/10.1007/s10270-017-0633-6

[58] 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, Sweden, 2014.

[59] J. Bastian, C. Clauss, S. Wolf, P. Schneider, Master for Co-Simulation Using FMI, in: 8th International Modelica Conference, 2011.

[60] 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.

[61] Thomas Nägele and Jozef Hooman, Co-simulation of Cyber-Physical Systems using HLA, in: Proceedings 7th IEEE Annual Computing and Communication Workshop and Conference (CCWC 2017), IEEE, 2017, pp. 267–272.

[62] 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).

[63] C. Thule, C. Gomes, J. Deantoni, P. G. Larsen, J. Brauer, H. Vangheluwe, Towards the Verification of Hybrid Co-simulation Algorithms, 2018, Accepted for publication at the CoSim-CPS-18 Workshop.

[64] S. Sadjina, L. T. Kyllingstad, S. Skjong, E. Pedersen, Energy conservation and power bonds in co-simulations: non-iterative adaptive step size control and error estimation, Engineering with Computers 33 (3) (2017) 607–620.

doi:10.1007/s00366-016-0492-8.

URL https://doi.org/10.1007/s00366-016-0492-8