Grant Agreement: 644047

INtegrated TOol chain for model-based design of CPSs

Final Integration of Simulators in the INTO-CPS Platform

Deliverable Number: D4.3b

Version: 1.0

Date: December 2017

Public Document

http://into-cps.au.dk

Contributors:

Adrian Pop, LIU
Victor Bandur, AU
Kenneth Lausdahl, AU
Casper Thule, AU
Marcel Groothuis, CLP
Tom Bokhove, CLP

Editors:

Marcel Groothuis, CLP

Reviewers:

Carl Gamble, UNEW
Frank Zeyda, UY
Erica Zavaglio, UTRC

Consortium:

Aarhus University	AU	Newcastle University	UNEW
University of York	UY	Linköping University	LIU
Verified Systems International GmbH	VSI	Controllab Products	CLP
ClearSy	CLE	TWT GmbH	TWT
Agro Intelligence	AI	United Technologies	UTRC
Softeam	ST		

Document History

Ver	Date	Author	Description
0.1	05-08-2017	Marcel Groothuis	Initial document version
0.2	27-10-2017	Kenneth Lausdahl	Updated document structure
0.3	30-10-2017	Kenneth Lausdahl	Added initial COE description
0.4	30-10-2017	Casper Thule	Updated 2 and 3.1
0.5	03-11-2017	Marcel Groothuis	Updated 1, 3.2 and 4
0.6	06-11-2017	Marcel Groothuis	Finish text chapter 4 and add conclusions
0.7	07-11-2017	Marcel Groothuis	Add pictures to chapter 4 and write conclusions
0.8	08-11-2017	Adrian Pop	Update the OpenModelica part
0.9	09-11-2017	Marcel Groothuis	Last edits for the internal review
0.10	21-11-2017	Kenneth Lausdahl	Added Section 2.4 and performance data
0.11	22-11-2017	Casper Thule	Updated Section 2
0.12	22-11-2017	Casper Thule	Updated Section 2 and Appendices A, B and C.
0.13	12-12-2017	Marcel Groothuis	Updated abstract, Section 1, 3.2, 4
0.14	14-12-2017	Adrian Pop	Updated OpenModelica Section
1.0	15-12-2017	Marcel Groothuis	Ready for Review

Abstract

This deliverable contains the design specifications for integration of simulators (OpenModelica, Overture, 20-sim) and the rapid prototyping tool 20-sim 4C with the Integrated Tool Chain for Model-based Design of Cyber-Physical Systems (INTO-CPS) co-simulation orchestration engine (COE) at the end of the project. The integration of the simulation tools into the COE uses the Functional Mockup Interface (FMI) with INTO-CPS extensions. In the third year of the project, FMI-based co-simulation has been extended to include Hardware-In-the-Loop (HIL) simulation support.

Contents

1 Introduction

This deliverable contains the design for the integration of the simulators OpenModelica, Overture and 20-sim with the co-simulation orchestration engine (COE) using the FMI 2.0 standard for co-simulation. Next to the integration of the above mentioned simulators, this deliverable also describes the integration of HIL simulation in the COE using 20-sim 4C and real-time targets.

The integrated simulators in this project are:

- OpenModelica [Fri04], `https://openmodelica.org`
- Overture [LBF+10], `http://overturetool.org/`
- 20-sim [KGD16], `http://www.20sim.com/`

The integration of HIL simulation with the COE is done using:

- 20-sim 4C (Rapid prototyping)[Kle13], `http://www.20sim4c.com/`
- a Raspberry Pi 3 (Embedded computing board) [Ras17], `https://www.raspberrypi.org/`
- the Xenomai real-time Linux extension [Xen17], `http://xenomai.org/`

This deliverable is a continuation of Deliverable D4.2b [PBLG16] and it describes the updates that have been developed in the third year of the INTO-CPS project.

Chapter 2 describes the INTO-CPS Co-simulation Orchestration Engine (COE). This is a FMI 2.0 compliant master that is used to integrate the above-mentioned tools. Chapter 3 summarizes the FMI tool support status in November 2017 with a focus on the INTO-CPS Year 3 improvements and changes for Overture (Section 3.1), 20-sim (Section 3.2) and OpenModelica (Section 3.3). Further details can be found in the previous deliverables D4.2b [PBLG16] (Year 2) and D4.1b [PBLG15] (Year 1). Chapter 4 contains the design and implementation of the new HIL simulation feature. Chapter 5 summarizes the tool enhancements from the final year of the INTO-CPS project.

1.1 Requirements

The high-level requirements from the INTO-CPS requirements report in deliverable D7.7 [LPO+17] with focus on the INTO-CPS Integration of Simulators are presented below for the different baseline tools.

- Requirement **0007** - The COE must be able to monitor the overall stability of a co-simulation based on the suggested step-size, step-size tolerance as well as input values (min, max and nominal) of FMUs.
  Status **100%**: requirement is met.
- Requirement **0008** - The COE must have algorithms in place to increase the overall stability of a co-simulation.
  Status **100%**: requirement is met.
- Requirement **0009** - The OpenModelica tool must provide an INTO-CPS FMI tool wrapper that is compliant with the COE.
  Status **98%**: requirement is almost met.
- Requirement **0010** - The 20-sim tool must provide an INTO-CPS FMI tool wrapper that is compliant with the COE.
  Status **100%**: requirement is met.
- Requirement **0011** - The Overture tool must provide an INTO-CPS FMI tool wrapper that is compliant with the COE.
  Status **100%**: requirement is met.
- Requirement **0042** - It must be possible to generate a HIL configured FMU from an existing 20-sim model FMU using 20-sim 4C.
  Status **95%**: requirement is almost met; minor fixes are needed to the import process for source code FMUs from other tools than 20-sim.

1.2 Related Work

Several approaches have been proposed in the past dealing with integration of simulators at different levels:

- Simulator-tool level - the tools are called as slaves by a master (covered by FMI for Model Exchange);
- Model-export level - the tool can export a model that can be imported in another tool (covered by FMI for Co-Simulation) and
- Source-code level - the tool can export source code that can be integrated with source code exported from other tools.

At the *simulator-tool level* several tools (MSC. Adams, OpenModelica, etc) were integrated using co-simulation within the SKF BEAST tool [SNF05], [Sie10]. Also integration of Overture and 20-sim has been achieved [GMF12] before in the DESTECS [DES09], [LRV+11] EU project at the *simulator tool-level*.

At the *model-export level*, for example, 20-sim can export Matlab S-functions. Similarly, 20-sim can import external model equations as precompiled DLL using its built-in `dlldynamic()` function [KGD16].

At the *source-code level*, 20-sim is able to generate C-code and C++-code based on templates that enables it to embed the code in a bigger framework. Please see deliverable D5.1d – *Design Principles for Code Generators* [HLG+15] for related work into integration at the *source-code level*.

In this project the integration is performed at *model-export level* where models are exported from tools as FMUs for co-simulation based on the FMI 2.0 standard. The exported FMUs can then be co-simulated using the COE.

Model export is realized in this project in the following tools:

- Overture: FMI 2.0 co-simulation toolwrapper FMU and FMI 2.0 co-simulation standalone/source code FMU;
- OpenModelica: FMI 1.0/2.0 model exchange and co-simulation FMU with source code;
- 20-sim: FMI 2.0 co-simulation toolwrapper FMU and FMI 2.0 co-simulation standalone/source code FMU;
- 20-sim 4C: FMI 2.0 co-simulation toolwrapper FMU and

- RTTester: Test Automation module export as FMI 2.0 co-simulation FMU. Details for RTTester can be found in Deliverables D5.2a and D5.2b [PLM16, BLM16].

Model-import (FMI 2.0 master) is realized in this project in the following tools:

- The COE (INTO-CPS Co-simulation Orchestration Engine);
- OpenModelica (FMI 2.0 FMU import);
- 20-sim (FMI 2.0 co-simulation FMU import) and
- 20-sim 4C (FMI 2.0 co-simulation FMU import; source code FMUs only; for HIL-simulation purposes).

Related tools that implement FMI compliant model-import and model-export features can be found on the FMI standard website [Blo14].

In year 3, 20-sim 4C has been extended with source code FMU import support with the goal to run FMUs in real-time on embedded targets with the possiblity to do HIL-simulation experiments. Related work on running FMUs in real-time for HIL-simulation is done by dSPACE for its VEOS and SCALEXIO platforms [dSP17].

2 Co-simulation Orchestration Engine

The Co-simulation Orchestration Engine (COE) [TLL18] is a fully Functional Mock-up Interface (FMI) 2.0 co-simulation compliant Master supporting both fixed and variable step size simulations. The COE is designed as a simulation service provider as described in Section 2.1 and is one of the FMI Masters available on the most platforms and architectures[1]. Not only does it support the major platforms, it is also capable of supporting distributed co-simulations with combinations of platform and architecture specific FMUs as described in Section 2.2. It has been used with FMUs exported from Overture, 20-sim, OpenModelica, Dymola, Modelon, SimulationX, 4Diac etc. [PLS+17, OLF+17, NZL+17, Tec16, Con13][2]. Furthermore, the COE also supports parallel execution to reduce overall simulation time of demanding co-simulations both on consumer computers and high performance clusters as described in Section 2.3. While some simulation scenarios benefit from increased simulation speed other cases do not, for example, cases where hardware is in the loop and requires that a simulation does not execute faster than real-time. This is the case for instance when using the COE in combination with Real-Time Operating Systems (RTOSs) as a Hardware-In-the-Loop (HIL)-simulation. To support this, a delay is performed if a step completes faster than the specified step size related to Wall-clock Time (WCT), which in this case is given in seconds.

In Section 2.4 an approach is described that enabled sub-systems to be encapsulated as an FMU, and an approach that can reduce simulation time when using models that require different step sizes.

The FMI Master supports both fixed and variable step size. A co-simulation configured to use fixed step size will progress using fixed steps as long as no FMUs fail. If a FMU fails, then a recovery algorithm attempts to resolve the issue, and during this recovery, an alternate step size may be taken before the entire simulation either fails or continues with the previously determined fixed step size. A co-simulation configured for variable step size will progress with a step size that is restricted initially only by the end time of the simulation but may be further restricted by the following constraints:

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

[1] See `http://fmi-standard.org/tools/` for tool availability.
[2] See `http://fmi-standard.org/tools/` for FMI Cross Check Results.

that a zero crossing is revealed as 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. A solver in a tool such as Simulink can adjust the step size using iterative approaches, but in a co-simulation, a roll-back of the participating models' internal states would be required. This, however, is in general not possible or efficient. Hence the variable step size calculator bases its step size adjustments on the prediction of a future zero crossing. It uses extrapolation and derivative estimation to estimate changes and therefore reduce the need for roll-back.

Bounded Difference: A bounded difference constraint is also a continuous constraint. It 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). The bounded difference problem is distinguished 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).

Sampling Rate: A sampling rate constraint is a discrete constraint. It constrains the step size such that repetitive, predefined time instants are exactly hit. This can be useful in co-simulation, for instance, when a modelled control unit reads a sensor value every x milliseconds.

FMU Max Step Size: This constraint implements the `getMaxStepSize` method from [BBG+13, CLB+16] providing a prediction of the maximal step size that a given FMU instance can perform at a given point in time. It limits the need to roll back a simulation, because each FMU participates in deciding the step size, and therefore all of them are capable of performing the determined step size. It is enabled by default and it constrains the step size as follows:

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

As previously described in Deliverable 4.2b [PBLG16], stability is an important aspect of any simulation both for the detection and ability to handle otherwise unstable co-simulations. The COE has the ability to detect some cases that may lead to an unstable model. In Year 3 it has also been upgraded to handle some models that it previously could not simulate because the model became unstable. Further detail on the stabilization techniques are described in Section 2.5 and in the *Mass Spring Damper* example i Deliverable D3.6 [MGP+17].

2.1 Client Interface

The COE is designed as a simulation service without any user interface. Its initial design was explained in detail in Deliverable D4.1d [LLW+15]. The Application Programming Interface (API) consists of JavaScript Object Notation (JSON) over Hypertext Transfer Protocol (HTTP) for simulation control and Web-sockets for live simulation progress information as shown in Figure 1.

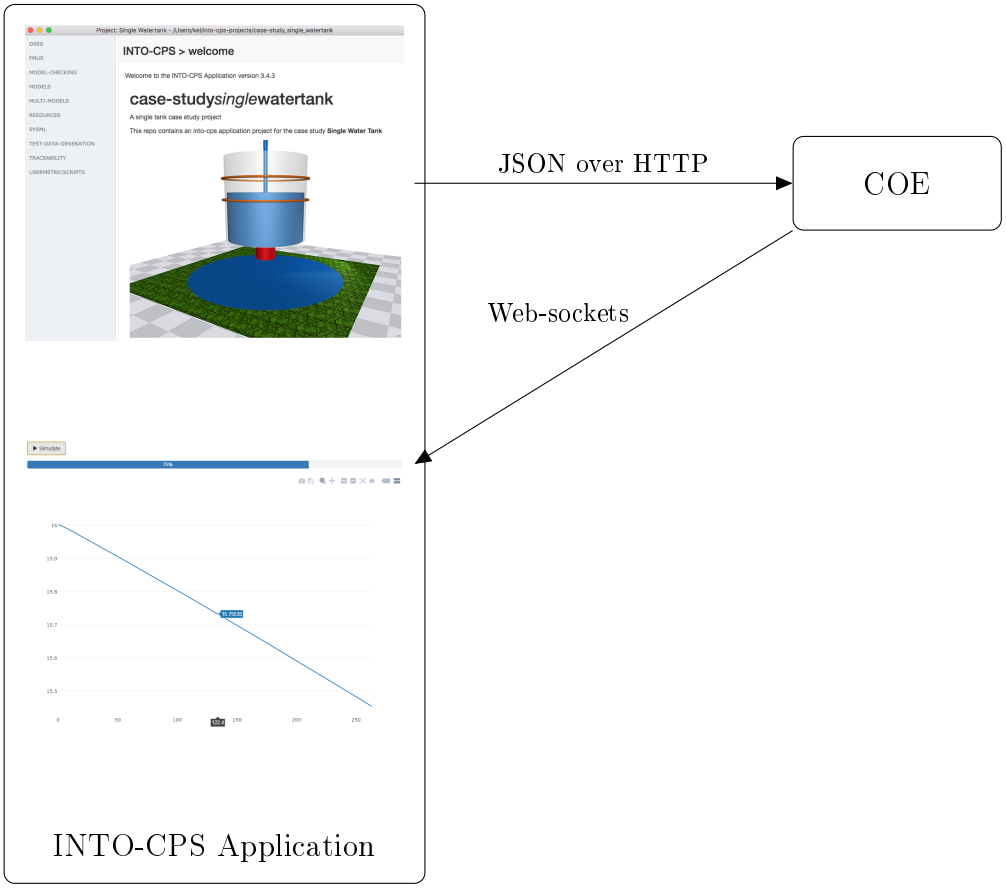

Figure 1: INTO-CPS Application interaction with the COE

To perform a co-simulation, a user can use a front-end user interface like the INTO-CPS Application, which can be used to create the configurations required by the COE and perform co-simulations. The INTO-CPS Application then creates a session for the given co-simulation in the COE and sends the configuration data to the COE. The COE then processes the configuration and informs of any issues using HTTP response. A detailed

description of the configuration can be found in Appendix B, Appendix C, and Appendix D.

The recent versions of the COE (0.2.16 and newer) enable the user to specify which FMU scalar variables should be included in the simulation result. Additionally, it also enables the user to create any number of graphs combining any output or local scalar variable from the FMUs in the co-simulation. To maintain performance, a filter can be enabled that reduces the frequency of data transmitted to the graphs, which is important for simulations with very small step sizes, where the graphs are only used to obtain an overview of the system behaviour. The graphs are also configurable to allow a floating time window view of the system, which is useful for long-running simulations. Simulation results are unaffected by these filters and always contain all data points generated in the co-simulation, thereby allowing post analysis.

2.2 Distributed Co-Simulation

The FMI standard enables models to be shared as FMUs, but in practice this only works if all FMUs taking part in a simulation support the same combination of platform and architecture. This limits the ways these FMUs can be shared. For example, an FMU exported as Windows 32-bit cannot be used in a co-simulation where another FMU only supports Windows 64-bit. The same limitation applies if the platform is different, which was the case in [PLS+17], where a control system was developed for a Water Handling System (WHS). The system was used to clean the exhaust gas using Exhaust Gas Recirculation (EGR) 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.

To overcome the challenge of mixed platforms and architectures, a custom plug-in was developed for the COE. This solved both the platform and architecture issue and improved performance for some system architectures[3]. The plug-in implements a distributed factory that enable FMU instances to be relocated to remote simulation daemons, which then perform the simulation as shown in Figure 2. 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

[3]Performance increase could be seen for architectures with less cores than the amount of work to be performed.

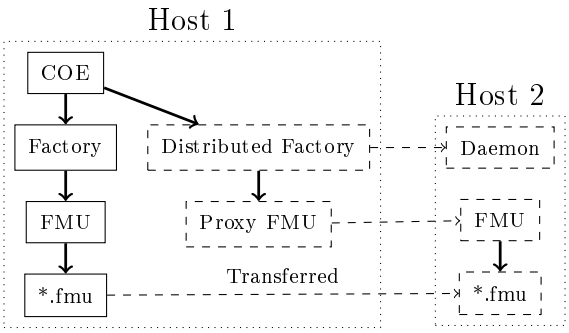

Figure 2: Distributed Extension Overview

from utilizing the extra resources on the remote host.

2.3 Co-Simulation Performance

It is always desirable to reduce the time it takes to perform a co-simulation, especially when searching the design space for optimal combinations as described in [Gam17]. Therefore the COE is designed to be capable of utilizing all system cores available for the inherent parallel operations of co-simulation. Depending on the configuration of the COE, the following operations are executed in parallel: `setX`, `doStep`, and `getX` for all instances of all FMUs that are part of a given co-simulation.

Initial exploratory research was carried out in [TL16], which indicated that not all simulations benefit from completely concurrent execution and that it is highly dependent on the number of FMUs, the number of instances and the time taken to perform `doStep` on each FMU instance. However, our internal tests have shown that there are significant performance improvements for some systems. Table 1 shows the test results of simulations using standard (*std.*) settings and parallel (*par.*) execution. The used FMUs are written in the Modelica language and exported using Dymola, see Appendix E for the full model. It performs intensive computation and it was required, that the computation was carried out such that it could not be removed by optimization. The *nLoop* parameter is used to adjust the internal computation of the FMUs.

It can be seen from Table 1 that the simulation speed is reduced to about 40% when the parallel execution is enabled, which is expected for this type of simulation with high load and few connections.

	Init	Sim	Avg. CPU	Total	Avg. Total
5 FMU, Sim: 10s, Step: 1s (fix), nLoop = 10M					
Std.	19.384	1497.871	25-50	1517.255	1517.26
Par.	19.421	641.031	100	660.452	660.45
Relative Difference					-56 %
10 FMU, Sim: 10s, Step: 1s (fix), nLoop = 10M					
Std.	46.933 55.718	3030 3018.665	25-50	3076.957 3074.383	3075.67
Par.	40 47.089	1097 1144.411	100	1137 1191.5	1164.25
Relative Difference					-62 %
15 FMU, Sim: 10s, Step: 1s (fix), nLoop = 1M					
Std.	5.988 6.749 6.914	450.19 480.992 473.835	25-50	456.178 487.741 480.749	474.89
Par.	5.83 7.004 6.897	159.72 161.604 163.509	100	165.55 168.608 170.406	168.19
Relative Difference					-65 %

Table 1: Performance Time Measurements.

While the COE, is designed as a service it can also be executed as a one-shot simulation from a Command-line Interface (CLI), which has proven useful in Design Space Exploration (DSE) scenarios [GMB17] where large clusters have been used to perform scheduled simulations on cluster nodes.

2.4 Hierarchical Co-Simulation

The FMI standard provides a convenient way to share and encapsulate models, however it does not specify how full modularization can be achieved for a sub-system. To exemplify this, the Water Treatment System (WTS) from [PLS+17] is used. The system developed by the company MAN Diesel reduces emissions using EGR. As part of this system, MAN Diesel buys a sub-system called WHS from another company, which then buys most of the sub-system components from third-parties. This means, that the company producing the WHS can use FMI to model the component they sell to be used as a sub-system in another product. They can make use of any FMUs provided from their third-party suppliers and perform co-simulation. However, they can not produce an FMU for the complete WHS product 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 end-customer, and therefore it is desireable to provide a means to encapsulate and share co-simulation scenarios, such that they can be utilized as a component in another co-simulation.

Based on the case mentioned above the COE has been extended with an FMU interface allowing it to act as the executable part of an FMU, which thereby makes the FMU a fully configured co-simulation. This is exemplified in a general setting in Figure 3. The approach taken is that a co-simulation configuration in the INTO-CPS Application can be imploded into an hierarchical FMU. This hierachical FMU includes the FMUs that are part of the co-simulation configuration, the COE and the required configurations to perform the internal co-simulation. The hierarchical FMU has the union of all inputs of the internal FMUs that are not internally connected and the union of all outputs of the internal FMUs. This approach enables the user to configure a sub-system and, once satisfied, simply remove the sub-system driver and implode the simulation, which generates a new hierarchical FMU. This new hierarchical FMU can be used as any other FMU in future simulations. Another potential gain is a performance enhancement, which may apply when the sub-system performs smaller steps than the simulation it is being used in. This allows parts of the co-simulation to take fewer but larger steps, which is faster than all components taking many small steps.

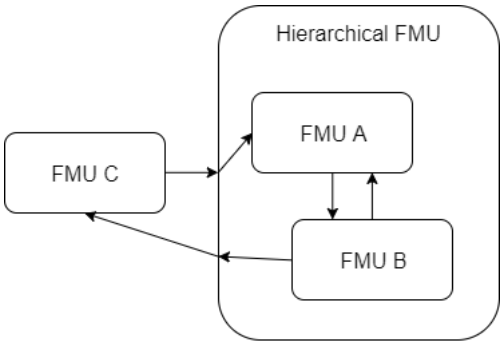

Figure 3: Example of an FMU C connected with a hierarchical FMU containing two FMUs: A and B

2.5 Co-Simulation Stability

To support co-simulation for a greater range of models, a number of improvements were added to the COE in Year 3 to not only improve stability in forms of detection but also to attempt to overcome stability challenges. This is based on [PBLG16], and directly related to the two requirements:

- Requirement **0007** - The COE must be able to monitor the overall stability of a co-simulation based on the suggested step-size, step-size tolerance as well as input values (min, max and nominal) of FMUs.
- Requirement **0008** - The COE must have algorithms in place to increase the overall stability of a co-simulation.

The COE issues warnings if a bound is specified for a scalar variable and it is violated. This takes place when the variable is retrieved from the FMU instance through the FMI function `getX`, or when a value is set using `setX`. A more advanced technique to overcome stability challenges is to use successive substitution. This enables models with cyclic dependencies to be simulated because it will attempt to stabilize the system by repeating a given step until the signals are close enough in relation to the tolerances specified. A simulation with stabilization enabled will be slower than a standard simulation. This is because the FMU instances that support rollback will be rolled-back every step and attempt to take the same step again, but this time with the values obtained from the previous step. If the signal values are not within a defined tolerance of the previous signal values, then another step and rollback will be performed until they either stabilize or the maximum number[4] of stabilization steps is reached. A mass spring damper example is shown in Deliverable D1.3a [OLF+17] and Deliverable D3.6 [MGP+17], which can be simulated with stabilization enabled in the COE. This model has cyclic dependencies and is unstable if simulated without stabilization enabled.

[4]The default bound on the stabilization is 5 steps.

3 Integration of Simulators

Integration of simulators in the INTO-CPS COE is achieved via the FMI 2.0 standard for co-simulation. Each of the simulators has implemented support for the FMI 2.0 standard. The next sections summarize the FMI 2.0 for co-simulation support and features of each simulator. Section 3.1 describes Overture, Section 3.2 describes 20-sim and Section 3.3 describes OpenModelica.

3.1 Overture

Years 1 and 2 of the INTO-CPS project saw the development of FMI support for Overture, including a tool-wrapper FMU exporter, as detailed in the Year 2 version of this document, Deliverable D4.2b [PBLG16]. Year 3 focused on FMI support in Overture toward deployment to hardware and toward HIL-simulation. This is achieved through standalone FMU export.

Unlike tool-wrapper FMUs, which contain a combination of simulation tool and model, standalone FMUs contain code generated from the model with Overture's C code generator, VDM2C [BHPG16]. The code is compiled for Windows, Mac and Linux platforms as static libraries. The generated code is generic, in the sense that it is intended for deployment outside an FMI setting, but it is specialized by Overture's FMU exporter using wrapper code that provides the FMI interface.

The FMI wrapper code contains two main features. First, communication between the COE and the generated code is achieved via buffer variables. When the COE needs to write updated inputs to the FMU, it does so by writing directly into the buffer variables. Internally, the FMU reads these buffers and forwards the values to the model code. After an invocation of `doStep`, the wrapper code synchronizes the recalculated model variables back with the buffer variables, which are then read by the COE. Second, the wrapper code implements both fixed and variable step-size co-simulation. For a given step size requested by the COE, the thread execution mechanism determines how many times each of the threads of the FMU can be executed, based on their declared period values. Those threads which fit an integral number of times in the step duration are executed the corresponding number of times, and the output of their execution is made visible to the COE. Those threads whose period is such that it either does not fit inside one step, or does not fit inside a step an integral number of times, are executed, but their

outputs are only made available to the COE once an execution duration finishes inside a given step. For instance, a thread with a period of 3 will execute when `doStep` is called with a step size of 2, but its outputs will be made available to the COE only at the end of the second call to `doStep`. This can be viewed as a form of hysteresis in the output values in such cases.

In order to make the generated code compatible with many embedded hardware platforms, the generated C code is compliant with the C89 standard.

3.2 20-sim

3.2.1 Introduction

20-sim is a modeling and simulation program for mechatronic systems and control engineering on the Windows operating system. With 20-sim, multi-domain dynamic models can be analyzed in the time and frequency domain for modeling and control purposes. For rapid prototyping and HIL-simulation purposes, C-code generation support is available using C-code templates for various C-code objectives.

With respect to simulation, 20-sim supports continuous time simulations, discrete time simulations and hybrid simulations. For variable step-size integration method support, simulation back-stepping is available, but not for discrete time systems. External interfacing to 20-sim is available using scripting (XML-RPC), custom DLL functionality and CSV file variables.

3.2.2 FMI Support

20-sim has tool support for both import and export of FMI:

Import FMU import is supported in the following ways:

Interface Definition An FMI `modelDescription.xml` can be imported into 20-sim. This results in an empty 20-sim block with the corresponding FMI interface. After adding the wanted model implementation, it can be exported again as an FMU.

Co-simulation 20-sim can import an existing FMI 2.0 co-simulation

FMU as a 20-sim block. 20-sim then acts as an FMI master simulator.

Export FMU export is supported in the following ways:

Tool Wrapper The 20-sim simulator has a built-in co-simulation feature based on XML-RPC calls. A tool wrapper FMU was developed for INTO-CPS in Year 2 that uses this interface to allow an FMI co-simulation with a 20-sim model running inside the 20-sim simulator. The FMU itself acts as an FMI wrapper for the existing XML-RPC co-simulation interface.

Standalone FMU export for FMI 1.0 and FMI 2.0 *Co-Simulation* is supported in 20-sim since version 4.6. The FMU export template is based on the platform-independent, standalone, ANSI-C source code template included in 20-sim. In contrast to the tool wrapper FMU, this FMU type has no dependencies on the 20-sim tool. FMI 2.0 *ModelExchange* is not supported.

3D Animation 20-sim can export an existing 20-sim 3D animation as visualization FMU. This special type of FMU does not contain any model, but instead it shows a 3D animation window. This FMU only has inputs, and thus no outputs. Furthermore, this type of FMU is currently only supported on the Windows platform (32-bit and 64-bit).

Year 3 updates to the 20-sim FMI support are:

- A new cross-platform FMI visualization solution based on Unity [Tec16] has been developed in year 3 that can replace the above-mentioned Windows only 3D animation FMU. To make it easier to move to Unity, a translation tool from the 20-sim 3D animation format to a Unity scene has been developed. This allows for a largely automated conversion from an existing 20-sim 3D animation scenery to a Unity visualization.
- Time-event and frequency event support for standalone FMU export has been added.
- The standalone FMUs are now compatible with dSPACE VEOS.
- Vode-Adams variable step-size method support has been improved for interaction with an FMI variable step-size co-simulation algorithm.
- Support for storing and restoring the FMU state has been added. This allows for restarting a co-simulation from a particular moment in time. It can be used for rollback purposes in a co-simulation when one of the

other FMUs could not take the proposed step selected by the master algorithm. When all other FMUs support restoring a previous state, the master algorithm can retry with a different step size.

- Various bugfixes and improvements based on user feedback and FMI-crosscheck results.

3.2.3 Mapping between 20-sim and FMI

20-sim 4.6 supports FMU model export using a dedicated C-code generation template (standalone FMU). The mapping from a submodel in 20-sim to an FMI description is a one-to-one translation of the sorted model equations to the corresponding ANSI-C code lines. The FMU code generation template adds an FMI 1.0 or 2.0 co-simulation interface around the generated code. Detailed information on the 20-sim code generation process can be found in deliverable D5.1d [HLG+15].

All 20-sim model inputs and outputs are mapped to FMI input and output variables. 20-sim parameters are mapped to FMI parameters and all 20-sim variables are mapped to FMI scalar variables. Matrices and vectors are flattened to a list of FMI scalar variables following the FMI structured naming convention [Blo14]. All 20-sim variables are accessible from the FMI interface. 20-sim C-code generation only supports variables of type double. This means that the available 20-sim types integer and boolean will also be converted to type double in the generated code. In the FMU interface, the original type as specified in the 20-sim model is used, which means that the FMI variable set and get functions will do a conversion to and from double values internally.

Unimplemented FMI Functionality Only co-simulation FMI is supported. Model exchange FMI will not be supported in 20-sim within the scope of the INTO-CPS project. An overview of the FMI functions that are not implemented is listed in Table 2.

3.2.4 Additional Simulator Capabilities

Tool Wrapper Approach Instead of implementing the model internally in the FMU, the FMU can also interact with a running instance of the 20-sim simulator tool. This is called the tool wrapper approach. The tool wrapper approach has the advantage that the co-simulated (sub)model can

FMI 1.0	FMI 2.0
fmiGetString	fmi2GetString
fmiGetStringStatus	fmi2GetStringStatus
fmiSetString	fmi2SetString
fmiGetRealInputDerivatives	fmi2GetRealInputDerivatives
fmiSetRealInputDerivatives	fmi2SetRealInputDerivatives
fmiGetRealOutputDerivatives	fmi2GetRealOutputDerivatives
	fmi2SerializedFMUstateSize
	fmi2SerializeFMUstate
	fmi2DeSerializeFMUstate
	fmi2GetDirectionalDerivative

Table 2: FMI functions currently not implemented

be inspected from within 20-sim itself. Also, results of a co-simulation can be inspected in 20-sim. After the co-simulation has finished, the FMU will close the connection with the tool, but the simulation results can still be inspected within 20-sim.

External Monitoring To inspect the progress of a 20-sim simulation by an external application, 20-sim has been extended with functionality to transfer large data sets of the most recent simulation data on request. This so-called monitoring extension has been written specifically for the INTO-CPS project. This means that an external tool can register a list of monitor variables in 20-sim. During simulation, the external application can retrieve the values for these registered variables. The monitoring action itself does not interrupt the simulation in 20-sim. One application of this monitoring functionality is to pass 20-sim data to software that can visualise this data in a 3D scenery. A connection to Unity [Tec16] (a game engine) is setup to animate a 3D-scenery based on variables obtained from 20-sim. Unity can make visually appealing sceneries, and the visualisation can be done on multiple operating systems. This enables distributed visualisation on separate computers. This can be used for training simulator purposes, for example. The interface is built upon the XML-RPC interface, which enables other tools to use the monitoring functionality for their own purposes.

3.3 OpenModelica

OpenModelica [Fri04] is an open-source Modelica-based modeling and simulation environment. Modelica [FE98] is an object-oriented, equation-based

FMI 1.0	FMI 2.0
	`fmi2GetFMUstate`
	`fmi2SetFMUstate`
	`fmi2FreeFMUstate`
	`fmi2SerializedFMUstateSize`
	`fmi2SerializeFMUstate`
	`fmi2DeSerializeFMUstate`

Table 3: FMI functions currently not implemented

language to conveniently model complex physical systems containing, e.g., mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents. The Modelica language (and OpenModelica) supports continuous, discrete and hybrid time simulations. OpenModelica always compiles Modelica models into FMU, C or C++ code for simulation. OpenModelica supports Windows, Linux and Mac Os X.

OpenModelica has support for static and dynamic debugging of Modelica models [PSA+14]. Static debugging helps the user understand how his model has been optimized and solved by the compiler via an equation browser. Dynamic debugging is currently available for algorithmic Modelica code and supports breakpoint-based debugging.

3.3.1 FMI Support

OpenModelica supports FMI 1.0 for model-exchange import and export and FMI 2.0 export and import both for model-exchange and co-simulation. In Year 3 the focus was on making the FMI export more stable and fix issues found during usage (errors in dependencies inside model structure, setting of discrete inputs, default nominal attributes, missing values of final parameters, missing dependent dlls, etc.) . Support for analytic jacobians via `fmi*GetReal*Derivatives` has been implemented and is being tested.

An overview of the FMI functions that are not implemented is listed in Table 3. All the other functions that are given in the FMI standard are implemented.

4 HIL-simulation support

4.1 Introduction

One of the goals in Year 3 for the INTO-CPS project was to extend the FMI co-simulation usage towards partially running on real hardware. This is also known as Hardware-In-the-Loop (HIL) simulation. A HIL-simulation experiment is a variant of a co-simulation experiment in which one or more parts (e.g. models) run on real hardware. To achieve this goal, CLP has extended 20-sim 4C to import source code FMUs to allow running them on real hardware. To perform HIL-simulations with an FMU running on the real hardware, a special tool-wrapper FMU has been developed that acts as a communication link between the COE and the FMU running on the Raspberry Pi. This addresses INTO-CPS requirement **0042**.

This chapter describes the implementation details for running FMUs in real-time as standalone tasks or as part of a HIL-simulation. Section 4.2 describes the extension of 20-sim 4C that allows for importing FMUs and running them on real hardware. Section 4.3 describes the selected hardware target for demonstrating HIL-simulation with 20-sim 4C. HIL-simulation puts particular demands on the used models, FMUs and the communication protocol. This chapter concludes in Section 4.4 with a summary of guidelines and limitations for the current HIL implementation.

4.2 20-sim 4C

4.2.1 Purpose

20-sim 4C is a rapid-prototyping environment that enables you to run models (generated as ANSI-C-code) on various targets like PCs, Bachmann PLCs and various embedded boards. 20-sim 4C can be used for:

- **Measurement and Calibration:** Run models that read sensors;
- **Machine Control:** Run models to control the operation of machines;
- **Rapid Prototyping:** Start and stop the model; change parameters during run-time; monitor signals during run-time; log signals for off-line analysis and import the log data back in 20-sim to finetune the model.

20-sim 4C can import generated C-code from multiple sources:

1. 20-sim
2. Matlab Simulink
3. Scilab/Xcos
4. FMI 2.0 co-simulation source code FMU (Year 3 INTO-CPS addition)

The next section focusses on Item 4, the FMI 2.0 import extension.

4.2.2 FMI 2.0 Import

The input for 20-sim 4C is a set of ANSI-C source files, together with an XML-file that describes the model configuration. This model configuration contains a description of the inputs, outputs, variables and parameters contained in the source code.

The implemented steps to import an FMU are:

1. Extract the FMU
2. Check for a valid FMU (FMI 2.0 *Co-Simulation* with source code)
3. Generate a 20-sim 4C compatible XML model configuration file from the FMI model description
4. Generate 20-sim 4C project
5. Open 20-sim 4C project

The result is a 20-sim 4C project that can be further configured for a specific target. This process is shown at the top in Figure 4. See also [BLL+17] (Section 5.3) for the corresponding FMU import manual.

4.2.3 Running an FMU in real-time with I/O

The next steps to get the FMU running in real-time on a target are the 20-sim 4C steps:

- **Configure**: Select a target and discover it on the network.
- **Connect**: Connect FMU inputs and outputs to target I/O (e.g. AD/DA converters, digital I/O etc.) and extend the FMU code.

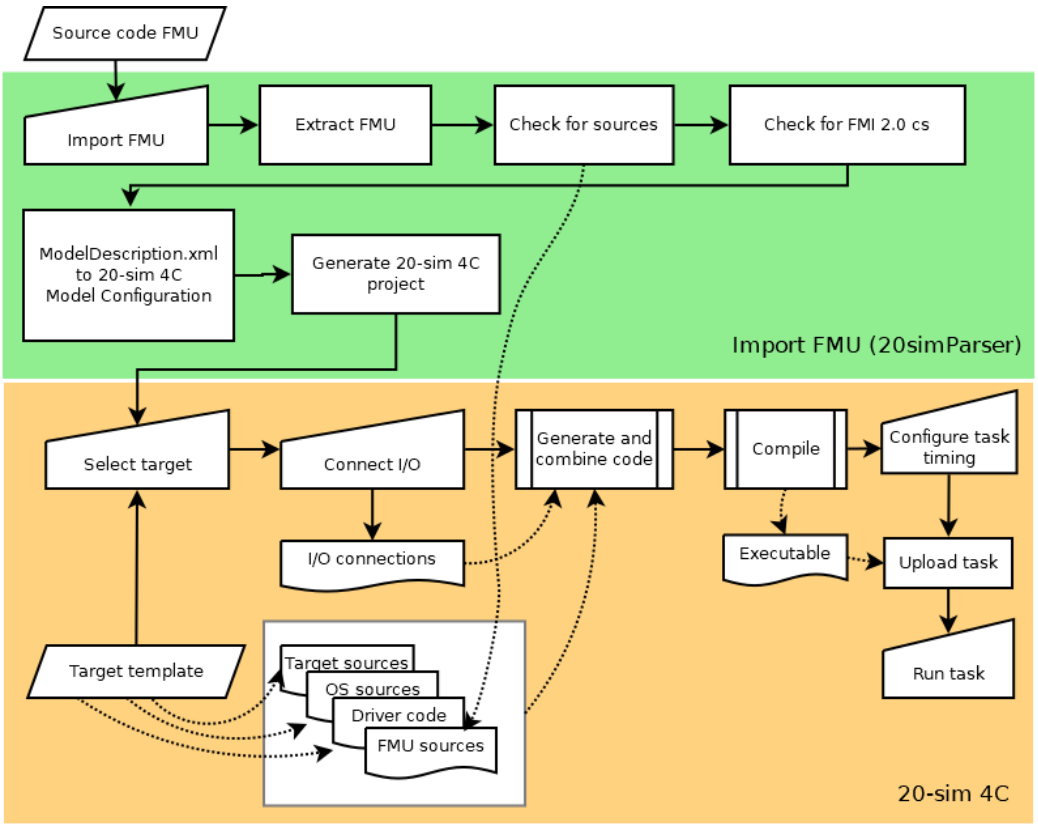

Figure 4: Steps to run a source code FMU in real-time

- **Compile**: Combines and compiles code from various sources to a target executable. The combined code consists of:
  - FMU source code;
  - 20-sim 4C FMI wrapper;
  - connection code (connects FMU inputs/outputs to I/O);
  - real-time task framework (operating system specific code);
  - board support code (board initialization/cleanup) and
  - I/O driver code.
- **Command**: Configure task timing settings, upload the target executable and start the model task

This process is shown in the orange box at the bottom of Figure 4. The result of these steps is a running FMU on the target (see also Figure 5).

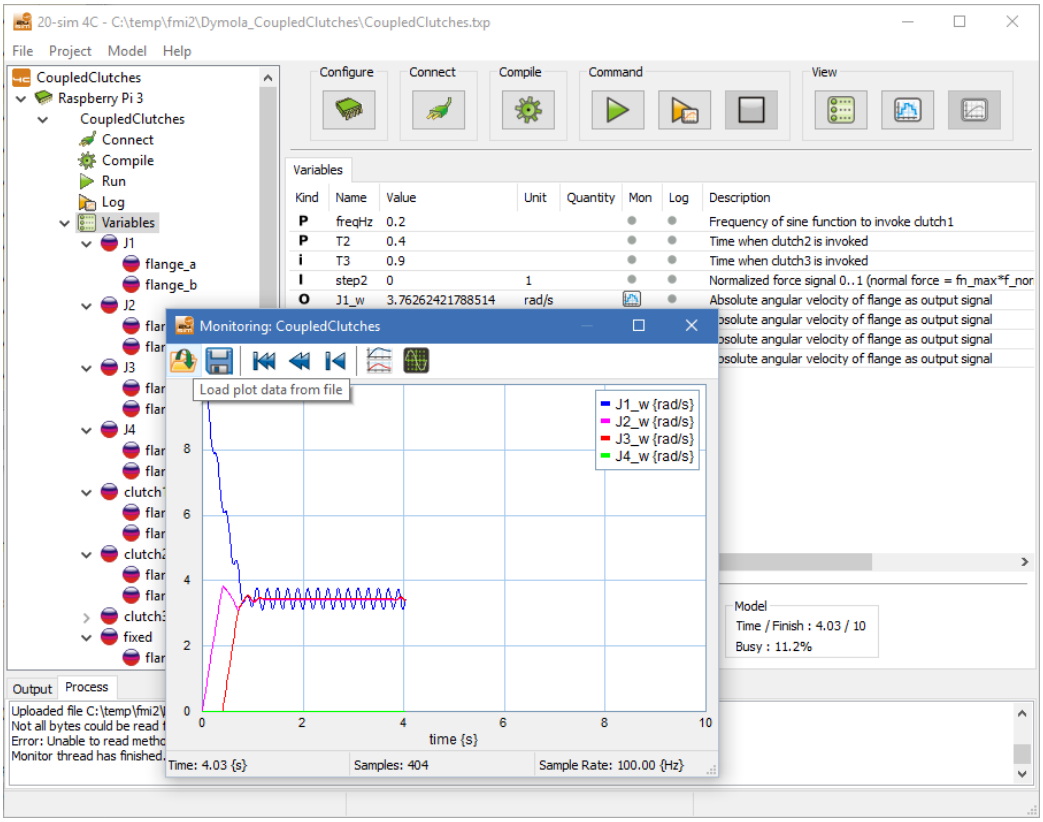

Figure 5: 20-sim 4C running an imported CoupledClutches FMU on a Raspberry Pi 3.

The above-mentioned 20-sim 4C FMI wrapper implements an FMI 2.0 co-simulation master algorithm. Both the co-simulation master and the co-simulation slave (FMU) are combined in a single executable to be able to run a single FMU as standalone program.

4.2.4 HIL-simulation support

The result of the previous section is a real-time FMU running on a target. Doing a HIL-simulation requires a connection between our real-time FMU and a simulation environment. The simulation environment here is the INTO-CPS COE co-simulating one or more FMUs on a PC. This connection has been realized by means of a special real-time tool-wrapper FMU. 20-sim 4C has been extended with a tool-wrapper export function that generates this real-time tool-wrapper FMU for the selected model and target.

This tool-wrapper FMU has multiple goals:

1. It provides a standard FMI 2.0 co-simulation interface for importing in the COE and other FMI 2.0 co-simulation engines;
2. It provides an XML-RPC communication interface between the PC and the real-time target;
3. It sends FMU inputs to target inputs;
4. It receives FMU outputs from target outpus;
5. It allows for run-time parameter updates (tunable parameters) for the real-time FMU and
6. It provides real-time synchronization (wall-clock sync).

With this tool-wrapper FMU, one can set-up a HIL-simulation experiment as depicted in Figure 6.

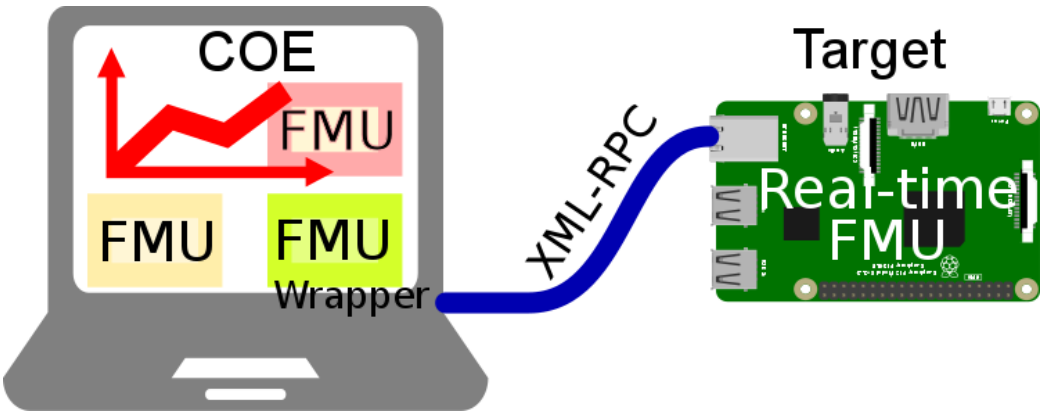

Figure 6: HIL simulation on the Raspberry Pi 3

4.3 Target Platform

The above-mentioned FMI extension for 20-sim 4C allows in principle to run FMUs on all target platforms supported by 20-sim 4C (Bachmann PLCs, industrial PCs and various embedded boards). The selected hardware platform for running FMUs in real-time for INTO-CPS is a Raspberry Pi 3 [Ras17] running Xenomai real-time Linux [Xen17].

4.3.1 Raspberry Pi 3

The selected target platform for running FMU is the Raspberry Pi [Ras17]. The Raspberry Pi boards are small, affordable and powerful embedded ARM boards suitable for educational, hobby and (small) industrial projects. The Raspberry Pi is a good compromise between a small embedded microcontroller like the Arduino (AVR) and a full scale x86/x64-based (industial) PC. The Raspberry Pi provides on-board digital I/O, PWM outputs and serial buses like uart, SPI and i2c. Other I/O can be connected via the available serial buses. The Raspbery Pi boards provide at least 256 MB of RAM, which allows us to log many FMU and I/O signals simultaneously for offline analysis later on.

4.3.2 Xenomai Real-time Linux

Several flavors of real-time Linux exist with different features and different timing accuracies. We have selected the Xenomai 2.6 real-time framework [Xen17] as real-time extension for the mainline Linux kernel. Xenomai is suitable for running industrial applications with stringent response time requirements alongside regular Linux applications. Xenomai is based on a dual kernel approach with a nano-kernel called ADEOS running next to a patched Linux kernel. The regular Linux kernel is patched with an *interrupt pipeline* (I-pipe) that can reroute hardware interrupts to the real-time part. In this way, real-time applications can get the interrupts (including timers) first, giving them a higher priority than the entire Linux system. This assures predictable and stable task response times. For a Xenomai task with a frequency of 1000 Hz, we have measured on the Raspberry Pi 3 a jitter on the task start time of only 7μs (0.7% of the task period). This was measured by means of toggling a digital output pin and measuring the jitter on an oscilloscope.

The Xenomai patches for the Linux kernel are available for several hardware architectures like x86, x64 and ARM. Support for ARM is available for a limited set of boards (like the Raspberry Pi 1, 2 and 3) because all dedicated board support packages need a patched interrupt pipeline.

4.3.3 20-sim 4C support

To make the Raspberry Pi suitable for 20-sim 4C usage and HIL-simulation, Controllab has created a dedicated SD-card image with a modified version

of the default Raspbian Linux installation [Ras17] using a Xenomai-patched kernel. Furthermore, the installation is extended with two daemon applications:

- Controllab Discovery daemon: allows 20-sim 4C to discover the board on the network;
- Controllab XML-RPC daemon: provides an XML-RPC scripting interface that provides functions for:
  - uploading tasks and related files;
  - starting real-time tasks;
  - modifying parameters;
  - monitoring and logging of I/O signals and task variables and
  - HIL-simulation (write unconnected inputs/read outputs).

4.4 Limitations

4.4.1 Models and FMUs

HIL-simulations require that the selected models can run in real-time, since part of the co-simulation experiment is running in real-time on real hardware. This limits the time budget for calculating the FMU using the `fmi2DoStep()` function.

A simple guideline for the models and their FMUs (INTO-CPS requirement **0084**) is that they should be able to be calculated faster than real-time on the selected target (Raspberry Pi 3) for all FMU steps. This is easy to test for fixed step-size FMUs by running the FMU standalone with 20-sim 4C on the target. If the reported FMU simulation time in 20-sim 4C is progressing slower than the wall-clock time, the FMU is not suitable for real-time experiments on the selected target.

Note 1: Variable step-size FMUs are supported, but their internal calculation step-size varies based on the dynamic behaviour of the model. This means that the amount of steps per second is not constant and therefore it is not possible to guarantee that all FMU steps can be calculated within a fixed time slot. It is therefore better to avoid them in a real-time setting when possible. If a variable step-size FMU is required, you should carefully check the timing of the running FMU.

Note 2: First tests from Controllab with source code FMUs generated by various other tools like OpenModelica, Overture, Dymola, Catia, dSpace and Maplesim show that the compile phase is the most troublesome phase. The FMI 2.0 standard does not specify in detail how to compile a source code FMU and it turns out that this freedom given by the standard gives mixed results across tool vendors. Some test FMUs are incomplete (OpenModelica 1.12 release; not all required sources are embedded and sources are not mentioned in the modelDescription.xml) while others contain a manual that describes how to compile the FMU. This makes a fully automated process from FMU to a running task on a real-time target hard. At this point, the FMI standard should be improved in the future. Source code test FMUs generated from 20-sim, Overture, Dymola, Catia and MapleSim have been tested succesfully. For OpenModelica changes are needed for their FMU export proces to allow compilation without Makefile. The OpenModelica team is working on a fix to allow a succesful 20-sim 4C import and compile process.

4.4.2 FMI co-simulation engines

Most FMI co-simulation engines, amongst others the INTO-CPS COE are not written with real-time simulation in mind. As a consequence, strict timing guaranties for hard-real-time HIL-simulation cannot be given. The best that can be achieved at this moment is soft-real-time HIL-simulation (best effort real-time). Similar to the limited time budget for FMU step calculation on the real-time target, the PC-side of the HIL-simulation also should simulate in real-time. This means that each individual FMU should calculate faster than real-time. The 20-sim 4C tool wrapper FMU will then limit the calculation frequency of the entire co-simulation experiment to real-time.

4.4.3 Communication protocol

The 20-sim 4C target communication protocol (XML-RPC) is not optimized for HIL-simulation purposes. It is designed for rapid prototyping with signal monitoring and incidental parameter changes as goals. Monitored signals are buffered at the target for a short time and blockwise transferred to 20-sim 4C. HIL-simulation requires direct read/write access to the inputs and outputs of the FMU. This is currently implemented using the existing XML-RPC protocol. The protocol overhead currently limits the amount of signals

that can be exchanged between the co-simulation and the Raspberry Pi to approximately one dozen signals. This can be improved in the future by selecting a more efficient (binary) communication protocol.

5 Conclusions

The focus in Year 3 of the INTO-CPS project from the simulator point of view was mainly on bugfixes and enhancements of the existing FMI features developed in the first two years. New developments focused on optimizations of co-simulation speed, running distributed co-simulations and running HIL-simulations.

5.1 COE enhancements

The INTO-CPS COE has further improved during Year 3. Several optimizations are added to speed-up the co-simulation. Support for WCT synchronization was added to support HIL-simulations. Stability monitoring support was added to monitor the stability of co-simulation experiments, and support for running distributed co-simulations on multiple PCs is now available.

5.2 Overture enhancements

Overture FMI support has seen further improvement towards deployment as ANSI-C code for embedded targets. Next to the default tool-wrapper FMU export, it is now also possible to export VDM code as standalone FMU with C-code. This C-code can be imported in 20-sim 4C for further deployment to embedded targets and to run HIL-simulations.

5.3 20-sim FMI enhancements

Most FMI 2.0 developments were done in Year 1 and 2. Year 3 FMI developments in 20-sim focussed on bugfixing, fine-tuning and minor enhancements. A major new development is the replacement of the 3D animation FMU by a new cross-platform FMU based on Unity.

5.4 OpenModelica FMI enhancements

Year 3 OpenModelica FMI support was focused on improvements and bug fixing to existing FMI export and import built in the first two years.

5.5 HIL-simulation using FMI

Year 3 developments around 20-sim 4C have extended the usage of the FMI standard towards running FMUs in real-time on actual hardware without manual code writing. In addition a real-time tool-wrapper FMU has been developed that closes the loop between a PC-based co-simulation and the real-time FMU running on a target. This provides HIL-simulation possibilities based on the FMI standard. First tests from our industial partners show that this proof-of-concept is working. However it needs further optimizations to scale to larger systems with more I/O. It proved challenging to accept all source-code FMUs because the FMI 2.0 standard documentation around source code FMUs is limited which gives freedom for various solutions that will not automatically import and compile.

5.6 Requirements

The high-level INTO-CPS requirements (Section 1.1) as defined at the beginning of the INTO-CPS project are all being met.

5.7 Future work

The INTO-CPS research project is almost finished, but the Overture, 20-sim, 20-sim 4C and OpenModelica development will continue. Future FMI-related changes, bugfixes improvements and extensions will be done as work within of the new INTO-CPS association (e.g the Unity 3D animation FMU) and as part of the normal tool development work flow.

References

[BBG+13] D. Broman, C. Brooks, L. Greenberg, E.A. Lee, M. Masin, S. Tripakis, and M. Wetter. Determinate composition of FMUs for co-simulation. In *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*, pages 1–12, 2013.

[BHPG16] Victor Bandur, Miran Hasanagic, Adrian Pop, and Marcel Groothuis. FMI-Compliant Code Generation in the INTO-CPS Tool Chain. Technical report, INTO-CPS Deliverable, D5.2c, December 2016.

[BLL+17] Victor Bandur, Peter Gorm Larsen, Kenneth Lausdahl, Casper Thule, Anders Franz Terkelsen, Carl Gamble, Adrian Pop, Etienne Brosse, Jörg Brauer, Florian Lapschies, Marcel Groothuis, Christian Kleijn, and Luis Diogo Couto. INTO-CPS Tool Chain User Manual. Technical report, INTO-CPS Deliverable, D4.3a, December 2017.

[BLM16] Jörg Brauer, Florian Lapschies, and Oliver Möller. Implementation of a Model-Checking Component. Technical report, INTO-CPS Deliverable, D5.2b, December 2016.

[Blo14] Torsten Blochwitz. Functional Mock-up Interface for Model Exchange and Co-Simulation. `https://www.fmi-standard.org/downloads`, July 2014.

[CLB+16] Fabio Cremona, Marten Lohstroh, David Broman, Marco Di Natale, Edward A. Lee, and Stavros Tripakis. Step revision in hybrid co-simulation with FMI. In *MEMOCODE*, pages 173–183. IEEE, 2016.

[Con13] Controllab Products B.V. http://www.20sim.com/, January 2013. 20-sim official website.

[DES09] DESTECS (Design Support and Tooling for Embedded Control Software). European Research Project, June 2009. `http://www.destecs.org`.

[dSP17] dSPACE GmbH. `https://www.dspace.com/en/inc/home/support/supvers/supverscompm/fmicompatibility.cfm`, November 2017. dSPACE FMI support website.

[FE98] Peter Fritzson and Vadim Engelson. Modelica - A Unified Object-Oriented Language for System Modelling and Simulation. In *EC-*

COP '98: Proceedings of the 12th European Conference on Object-Oriented Programming, pages 67–90. Springer-Verlag, 1998.

[Fri04] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1.* Wiley-IEEE Press, January 2004.

[Gam17] Carl Gamble. Comprehensive DSE Support. Technical report, INTO-CPS Deliverable, D5.3e, December 2017.

[GMB17] Carl Gamble, Oliver Möller, and Victor Bandur. Test automation module in the INTO-CPS Platform. Technical report, INTO-CPS Deliverable, D5.3a, December 2017.

[GMF12] C.J. Gamble, M. Mansfield, and J.S. Fitzgerald. The Co-Simulation of a Cardiac Pacemaker using VDM and 20-sim. In J. S. Fitzgerald, T. Mak, A. Romanovsky, and A. Yakovlev, editors, *Procs. Workshop on Trustworthy Cyber-Physical Systems*, volume CS-TR-1347 of *Technical Report Series*. School of Computing Science, Newcastle University, UK, 2012.

[HLG+15] Miran Hasanagić, Peter Gorm Larsen, Marcel Groothuis, Despina Davoudani, Adrian Pop, Kenneth Lausdahl, and Victor Bandur. Design Principles for Code Generators. Technical report, INTO-CPS Deliverable, D5.1d, December 2015.

[KGD16] C. Kleijn, M.A. Groothuis, and H.G. Differ. *20-sim 4.6 Reference Manual.* Controllab Products B.V., 2016.

[Kle13] C. Kleijn. *20-sim 4C 2.1 Reference Manual.* Controllab Products B.V., 2013.

[LBF+10] Peter Gorm Larsen, Nick Battle, Miguel Ferreira, John Fitzgerald, Kenneth Lausdahl, and Marcel Verhoef. The Overture Initiative – Integrating Tools for VDM. *SIGSOFT Softw. Eng. Notes*, 35(1):1–6, January 2010.

[LLW+15] Kenneth Lausdahl, Peter Gorm Larsen, Sune Wolf, Victor Bandur, Anders Terkelsen, Miran Hasanagić, Casper Thule Hansen, Ken Pierce, Oliver Kotte, Adrian Pop, Etienne Brosse, Jörg Brauer, and Oliver Möller. Design of the INTO-CPS Platform. Technical report, INTO-CPS Deliverable, D4.1d, December 2015.

[LPO+17] Peter Gorm Larsen, Ken Pierce, Julien Ouy, Kenneth Lausdahl, Marcel Groothuis, Adrian Pop, Miran Hasanagic, Jörg Brauer, Etienne Brosse, Carl Gamble, Simon Foster, and Jim Woodcock. Re-

quirements Report Year 3. Technical report, INTO-CPS Deliverable, D7.7, December 2017.

[LRV+11] Kenneth G. Lausdahl, Augusto Ribeiro, Peter Visser, Frank Groen, Yunyun Ni, Jan F. Broenink, Angelica Mader, Joey W. Coleman, and Peter Gorm Larsen. D3.3b — Co-simulation Foundations. Technical report, The DESTECS Project (INFSO-ICT-248134), December 2011.

[MGP+17] Martin Mansfield, Carl Gamble, Ken Pierce, John Fitzgerald, Simon Foster, Casper Thule, and Rene Nilsson. Examples Compendium 3. Technical report, INTO-CPS Deliverable, D3.6, December 2017.

[NZL+17] Mihai Neghina, Constantin-Bala Zamrescu, Peter Gorm Larsen, Kenneth Lausdahl, and Ken Pierce. A Discrete Event-First Approach to Collaborative Modelling of Cyber-Physical Systems. In Fitzgerald, Tran-Jørgensen, Oda, editor, *The 15th Overture Workshop: New Capabilities and Applications for Model-based Systems Engineering*, pages 116–129, Newcastle, UK, September 2017. Newcastle University, Computing Science. Technical Report Series. CS-TR- 1513.

[OLF+17] Julien Ouy, Thierry Lecomte, Frederik Forchhammer Foldager, Andres Villa Henriksen, Ole Green, Stefan Hallerstede, Peter Gorm Larsen, Luis Diogo Couto, Pasquale Antonante, Stylianos Basagiannis, Sara Falleni, Hassan Ridouane, Hajer Saada, Erica Zavaglio, Christian König, and Natalie Balcu. Case Studies 3, Public Version. Technical report, INTO-CPS Public Deliverable, D1.3a, December 2017.

[PBLG15] Adrian Pop, Victor Bandur, Kenneth Lausdahl, and Frank Groen. Integration of Simulators using FMI. Technical report, INTO-CPS Deliverable, D4.1b, December 2015.

[PBLG16] Adrian Pop, Victor Bandur, Kenneth Lausdahl, and Frank Groen. Updated Integration of Simulators in the INTO-CPS Platform. Technical report, INTO-CPS Deliverable, D4.2b, December 2016.

[PLM16] Adrian Pop, Florian Lapschies, and Oliver Möller. Test automation module in the INTO-CPS Platform. Technical report, INTO-CPS Deliverable, D5.2a, December 2016.

[PLS+17] Nicolai Pedersen, Kenneth Lausdahl, Enrique Vidal Sanchez, Peter Gorm Larsen, and Jan 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)*, pages 73–82, Madrid, Spain, July 2017. ISBN: 978-989-758-265-3.

[PSA+14] Adrian Pop, Martin Sjölund, Adeel Ashgar, Peter Fritzson, and Francesco Casella. Integrated Debugging of Modelica Models. *Modeling, Identification and Control*, 35(2):93–107, 2014.

[Ras17] Raspberry Pi Foundation. https://www.raspberrypi.org/, August 2017. Raspberry Pi Foundation website.

[Sie10] Alexander Siemers. *Contributions to Modelling and Visualisation of Multibody Systems Simulations with Detailed Contact Analysis*. Doctoral thesis No 1337, Linköping University, Department of Computer and Information Science, 2010.

[SNF05] Alexander Siemers, Iakov Nakhimovski, and Dag Fritzson. Meta-modelling of mechanical systems with transmission line joints in modelica. In *Proceedings of the 4th International Modelica Conference*, 2005.

[Tec16] Unity Technologies. Unity. `https://unity3d.com/`, December 2016.

[TL16] Casper Thule and Peter Gorm Larsen. Investigating concurrency in the co-simulation orchestration engine for into-cps. *Proceedings of the Institute for System Programming of the RAS*, 28(2):139–156, 2016.

[TLL18] Casper Thule, Kenneth Lausdahl, and Peter Gorm Larsen. Maestro: The into-cps co-simulation orchestration engine. 2018. To be submitted to Simulation Modelling Practice and Theory.

[Xen17] Xenomai. http://xenomai.org, August 2017. Xenomai website.

A List of Acronyms

20-sim	Software package for modelling and simulation of dynamic systems
ACA	Automatic Co-model Analysis
AST	Abstract Syntax Tree
AU	Aarhus University
CLI	Command-line Interface
CLP	Controllab Products B.V.
COE	Co-simulation Orchestration Engine
CPS	Cyber-Physical Systems
CT	Continuous-Time
DE	Discrete Event
DESTECS	Design Support and Tooling for Embedded Control Software
DSE	Design Space Exploration
FMI	Functional Mockup Interface
FMI-Co	Functional Mockup Interface – for Co-simulation
FMI-ME	Functional Mockup Interface – Model Exchange
FMU	Functional Mockup Unit
HIL	Hardware-in-the-Loop
HMI	Human Machine Interface
HW	Hardware
ICT	Information Communication Technology
IDE	Integrated Design Environment
M&S	Modelling and Simulation
MBD	Model Based Design
MIL	Model-in-the-Loop
OMG	Object Management Group
OS	Operating System
PROV-N	The Provenance Notation
RPC	Remote Procedure Call
SIL	Software-in-the Loop
SysML	Systems Modelling Language
TA	Test Automation
TRL	Technology Readiness Level
UML	Unified Modelling Language
VDM	Vienna Development Method
VSI	Verified Systems International
WCT	Wall-clock Time
WP	Work Package
XML	Extensible Markup Language

B COE Protocol

The COE protocol is based on a JSON over HTTP protocol using a similar approach as REST. However, unlike REST, the COE keeps state. The format used to describe each command is: URL and any arguments, prefixed with a colon, *e.g.* "*:section*".

B.1 COE Information

Information about the COE is available at:

`http://localhost:8082/`

B.2 The API Command

The command is available at:

`http://localhost:8082/api`

or the following for a PDF version:

`http://localhost:8082/api/pdf`

If successful, the command returns one of two types of content:

Content-Type: application/pdf A PDF version of this document.

Content-Type: text/plain The LaTeX source file of this document.

B.3 The Status Command

The command is available at:

`http://localhost:8082/status/:session`

The command takes the following arguments:

:session Optional session id filtering the returned data array to the single instance with the given session id.

If no session is provided, then the command returns an array with the status of all sessions similar to the example below.

```
[
{
   "status":"idle",
   "sessionid": "-1"
},
{
   "status":"idle",
   "sessionid": "0"
}
]
```

If a session ID is given, then a single session is returned

```
{
   "status":"idle",
   "sessionid": "-1"
}
```

B.4 The Create Session Command

The command is available at:

`http://localhost:8082/createSession`

The command takes no argument and returns a JSON string containing the session id. An example is presented below, where the sessionId is 12345:

```
{"sessionId":"12345"}
```

B.5 The Attach Session Command

The command is available at:

`ws://localhost:8082/attachSession/:session`

The command takes no arguments and opens a WebSocket (`https://tools.ietf.org/html/rfc6455`). Output data from connected outputs will be live streamed according to the following format:

```
{
   "{fmuName}":{
      "instanceName":{
```

```
            "variableName": variableValue
        }
    }
}
```

B.6 The Initialize Command

The command is available at:

`http://localhost:8082/initialize/:session`

The command takes the following argument and requires a JSON payload, Content-Type: application/json:

:session The session ID.

The Data payload:

```
{
    "fmus":{
        "{controllerFmu}":"file://controller.fmu",
        "{tankFmu}":"file://tank.fmu"
    },
    "connections":{
        "{controllerFmu}.crtlIns.valve":["{tankFmu}.tankIns.
            valve"],
        "{tankFmu}.tankIns.level":["{controllerFmu}.crtlIns.
            level"]
    },
    "parameters":{
        "{controllerFmu}.crtlIns.maxLevel":8,
        "{controllerFmu}.crtlIns.minLevel":2
    },
    "algorithm":{
        FIXED-STEP-SIZE-CONFIG or VARIABLE-STEP-SIZE-CONFIG
    },
    "livestream":{
        "{controllerFmu}.crtlIns":["local","local2"],
        "{tankFmu}.tankIns":["level"]
    },
    "logVariables":{
        "{tankFmu}.tankIns":["local"]
    },
    "parallelSimulation": false,
    "stabalizationEnabled":false,
    "global_absolute_tolerance": 0.0,
```

```
27        "global_relative_tolerance":0.01
28
29    }
```

FIXED-STEP-SIZE-CONFIG The fixed step size configuration contains the following:

```
1         "type":"fixed-step",
2         "size":0.1
```

VARIABLE-STEP-SIZE-CONFIG The variable step size configuration contains the following:

```
1         "type":"var-step",
2         "size":[1E-6, 1.0],
3         "initsize":1E-4,
4         "constraints":{
5            STEPSIZE-CONSTRAINT*
6         }
```

Where the properties are defined as:

type:"var-step" selects the variable stepsize calculator (each algorithm has a `type`).

size:[<minimal stepsize> , <maximal step size>] defines the stepsize interval as an array of double values.

initsize:<initial stepsize> defines the initial stepsize as double value.

constraints: defines the stepsize constraints as follows:

```
1            "id":{
2               type:"[zerocrossing|boundeddifference|
                    samplingrate|fmumaxstepsize]",
3               ...
4            }
```

The `id` is a string that is used to identify the constraint e.g. in the log. All constraints have a single common name-value pair with name `type`. The value of `type` specifies the type of the constraint; the other name-value pairs of the constraint depend on the value of `type`. The `zerocrossing` is described in Section B.6.1, `boundeddifference` in Section B.6.2, and `samplingrate` in Section B.6.3 `fmumaxstepsize` in Section B.6.4.

The JSON payload contains the following entries:

fmus A list of the location of the FMUs.

connections A map of connections, output to input.

parameters A map from parameter to value.

algorithm Step size algorithm configuration:

fixed-step A fixed step size algorithm is available requiring a step size to be specified.

livestream These scalar variables will be livestreamed via websockets. They must have a causality of either local or output.

logVariables These scalar variables will be logged. They must have a causality of either local or output.

parallelSimulation Optional boolean condition specifying if the COE should parallelize certain parts of the simulation. This condition is or'ed together with all `simulation.parallelise.*` properties.

stabalizationEnabled This enabled stabilization mode in the COE. Currently it makes use of *successive substitution* retrying a maximum of 5 steps. It uses the `global_absolute_tolerance` and `global_relative_tolerance` to decide if the signals are stable or another stabilization step is needed. The implementation uses the *Numpy.isclose* function. When this option is enabled the cyclic check is disabled and a warning is for any FMU that is in a cycle that does not support get and set state from FMI.

global_absolute_tolerance The global absolute tolerance used for stabilization.

global_relative_tolerance The global relative tolerance used for stabilization.

The command returns the following response on success:

```
{
   "status":"initialized",
   "sessionid": "1234",
   "avaliableLogLevels":{
      "{8c4e810f-3df3-4a00-8276-176fa3c9f001}.tank":[
      {
         "name":"logAll",
         "description":"Description of this loggin level"
      },
```

```
            {
                "name":"logError",
                "description":null
            }],
            "{8c4e810f-3df3-4a00-8276-176fa3c9f000}.controller":[
            {
                "name":"logAll",
            }]
        }
    }
```

The `avaliableLogLevels` value will be specific to the FMUs given in the initial payload. The `sessionid` is the ID that must be supplied in any subsequent calls.

B.6.1 Definition of a Zero Crossing Constraint

A constraint of `"type":"zerocrossing"` is defined by

```
    "id":{
        "type":"zerocrossing",
        "ports":[
        "<guid>.<instance>.<outport>",
        "<guid>.<instance>.<outport>"
        ],
        "order":[1|2],
        "abstol":<double>,
        "safety":<double>
    }
```

where the second entry in the `ports` list and the attributes `order`, `abstol` and `safety` are optional. The name-value pairs have the following meaning.

- `ports:` Defines the zero crossing function f as an array of strings of size 1 or 2. If one output port is provided, then f is the value of that output port. If two output ports are provided, then f is the difference between the values of the first and second output ports. Any other size of the string array is not supported.
- `order:` This name-value pair is optional; it specifies the extrapolation order that is used to predict a zero crossing (see Section C.3.1). First and second order extrapolation are supported. The default is second order extrapolation.

- `abstol:` This name-value pair is optional; it specifies the absolute tolerance. The stepsize calculator attempts to adjust the stepsize such that at a time instant t_{ZC} the absolute value of the zero crossing function f is smaller or equal to the absolute tolerance, $|f(t_{ZC})| \leq abstol$. The default value for the absolute tolerance is 10^{-3}.
- `safety:` This name-value pair is optional; it adjusts the conservatism of the zero crossing prediction. The neutral default value is 0.0. If the variable stepsize calculator fails to resolve a zero crossing of a particular co-simulation within the absolute tolerance (and the minimal stepsize is not the limiting factor), then the value for `safety` can be increased for more conservatism in the zero crossing prediction. Negative values for less conservatism are mathematically possible, but should probably not be used.

B.6.2 Definition of a Bounded Difference Constraint

A constraint of `"type":"boundeddifference"` is defined by

```
"id":{
   "type":"boundeddifference",
   "ports":[
   "<guid>.<instance>.<outport>"
   ,"<guid>.<instance>.<outport>"
   ,"<guid>.<instance>.<outport>"
   ...
   ]
   ,"abstol":<double>
   ,"reltol":<double>
   ,"safety":<double>
   ,"skipDiscrete":<boolean>
}
```

where entries after the first in the `ports` list and the attributes `abstol`, `reltol`, `safety` and `skipDiscrete` are optional. The name-value pairs have the following meaning.

- `ports:` Defines a set of values whose minimal and maximal value shall have a bounded difference. The set of values is defined by a non-empty array of strings. If one output port is provided, then the set of values comprises that output port's current value and its previous value. If at least two output ports are provided, then the set of values comprises the output ports' current values.

- `abstol`: This name-value pair is optional; it specifies the absolute tolerance. The stepsize calculator attempts to adjust the stepsize such that the absolute difference between the minimal and maximal value is smaller than the value of `abstol`. The default value for the absolute tolerance is 10^{-3}.
- `reltol`: This name-value pair is optional; it specifies the relative tolerance. The stepsize calculator attempts to adjust the stepsize such that the relative difference between the minimal and maximal value is smaller than the value of `reltol`. The default value for the relative tolerance is 10^{-2}.
- `safety`: This name-value pair is optional; it adjusts the conservatism of the algorithm that selects the next stepsize. The neutral default value is 0.0. If the variable stepsize calculator fails to keep the difference bounded (and the minimal stepsize is not the limiting factor), then the value of `safety` can be increased for more conservatism in the stepsize selection algorithm. Small negative values above $\alpha_{RISKY} - 1$, i.e. per default above -0.4 (see Table 7), are possible for less conservatism. Negative values below or equal to $\alpha_{RISKY} - 1$ lead to undefined behavior of the difference bin assignment algorithm (see Section C.4).
- `skipDiscrete`: This optional name-value pair is by default set to `true`, i.e. the skipping over previous stepsizes that were limited by discrete constraints (see Section C.7.3) is by default enabled. It may be disabled by setting this value to `false`.

B.6.3 Definition of a Sampling Rate Constraint

A constraint of `"type":"samplingrate"` is defined by

```
"id":{
   "type":"samplingrate",
   "base":<integer>,
   "rate":<integer>,
   "startTime":<integer>
}
```

with the following name-value pairs.

- `base`: Defines the exponent of 10 of the time base in seconds.
- `rate`: Defines the sample rate in multiples of 10^{base}.

- `startTime`: Defines the occurrence of the first sample hit in multiples of 10^{base}.

B.6.4 Definition of a FMU Max Step Size Constraint

A constraint of `"type":"fmumaxstepsize"` is defined by

```
"id":{
   "type":"fmumaxstepsize"
}
```

The constraint limits the step size to the minimum of the step size returned by `getMaxStepSize` from all instances that support the function.

B.7 The Simulate Command

The command is available at:

`http://localhost:8082/simulate/:session`

The command takes the following arguments and requires a JSON payload, Content-Type: application/json:

:session The session ID.

The Data payload:

```
{
   "startTime":0.0,
   "endTime":10.1,
   "logLevels": {
      "{8c4e810f-3df3-4a00-8276-176fa3c9f001}.tank":
      ["logAll", "logError"],
      "{8c4e810f-3df3-4a00-8276-176fa3c9f000}.tank":
      ["logError"]
   }
}
```

The payload contains the start and end time interval plus the log levels.

The command returns the following response on success:

```
[
{
   "status":"Finished",
```

```
4           "sessionid": "1234"
5       }
6       ]
```

B.8 The Stop Simulation Command

This command sets a flag such that the simulation related to a given sessionID is stopped on completion of its current step.

The command is available at:

`http://localhost:8082/stopsimulation/:session`

The command takes the following arguments:

:session The session ID.

B.9 The Result Command

The command is available at:

`http://localhost:8082/result/:session/:type`

The command takes the following arguments:

:session The session ID.

:type Optional parameter. Possible parameters are: plain/zip and default is plain.

The command returns the following response on success. A response supports two return formats selected by the `:type` argument and indicated using the content type:

Content-Type: application/zip Returns a zip file containing the initialization data + start data + the result obtained during the simulation

Content-Type: text/plain Returns the result obtained during the simulation as text. The result is a CSV formatted string with: time, stepsize, and all outputs at that time

B.10 The Destroy Command

The command is available at:

`http://localhost:8082/destroy/:session`

The command takes the following arguments:

:session The session ID.

The command destroys a session and releases all resources bound to the session on success full termination.

B.11 The Reset Command

The command is available at:

`http://localhost:8082/reset`

The command resets the COE to its initial state on success full termination.

C COE Variable Stepsize Calculation

Three of the four constraint types (Zero Crossing, Bounded Difference, and Sampling Rate) are defined in a JSON file that is posted to the COE with the initialize command (see Section B.6), and one (FMU-requested) is requested by the simulated FMUs.

After initialization, the variable stepsize calculator holds a set of constraint handlers. Each handler is responsible for one constraint. When asked for the next stepsize by the COE, the variable stepsize calculator asks each handler for the next stepsize and returns the minimum of these values.

C.1 Interface with the Master Algorithm

The variable stepsize calculator is called by the master algorithm before each `doStep`. It is given by the master algorithm the current time, the previous stepsize, the current output values, and the (estimated) output derivatives of

the FMUs. The variable stepsize calculator returns to the master algorithm the next stepsize.

After a `doStep`, the master algorithm asks the variable stepsize calculator to validate the taken step, i.e. to check whether any constraints have been violated. If that is the case, a warning is issued. If all FMUs support rollback, a rollback is initiated and the master algorithm asks the variable stepsize calculator for a new, reduced stepsize.

The algorithm for derivative estimation, see Section C.3.2, has been moved from the variable stepsize calculator to the COE. This is done so that the master algorithm may estimate derivatives and supply these to FMUs that have the capability canInterpolateInputs. To be clear, if the FMU that supplies these signals also provides derivatives, these are used, but if that FMU has `maxOutputDerivativeOrder=0` (or `<=1` in the case of second order input derivatives]), the estimated values are used.

C.2 Constraint Types

There are four constraint types:

- Zero Crossing
- Bounded Difference
- Sampling Rate
- FMU Max Step Size

The constraints are defined in the JSON file (see Section B.6). The fourth constraint, FMU Max Step Size, was enabled by default until COE version 0.2.14. See Section C.6 for more info on the FMU Max Step Size Constraint.

C.3 Zero Crossing Constraints

A zero crossing constraint is a continuous constraint. A zero crossing occurs at the point where a function changes its sign. In simulation, it can be important to adjust the stepsize such that a zero crossing is hit (more or less) exactly. 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 that.

A solver in a tool such as Simulink can adjust the stepsize using iterative approaches, but in a co-simulation a rollback of the participating models' internal states is in general not possible or efficient. Hence, the variable stepsize calculator bases its stepsize adjustments on the *prediction* of a future zero crossing.

C.3.1 Extrapolation

To predict a future zero crossing, the zero crossing function f must be extrapolated.

For first order extrapolation, the following calculation is used:

$f(t + \Delta t) = f(t) + \dot{f}(t)\Delta t$

For second order extrapolation, the following calculation is used:

$f(t + \Delta t) = f(t) + \dot{f}(t)\Delta t + 0.5\ddot{f}(t)\,(\Delta t)^2$

C.3.2 Derivative Estimation

The derivatives $\dot{f}(t)$ and $\ddot{f}(t)$ are either provided by the FMUs (if the capability `maxOutputDerivativeOrder` is high enough), or estimated. For first order extrapolation, the last two data points are used to estimate the first derivative. For second order extrapolation, either the last three data points are used to estimate the first and second derivate, or, if the FMU provides the first but not the second derivative, the last two data points and their first derivatives are used to estimate the second derivative.

C.3.3 Extrapolation Error Estimation

Extrapolation will generally incur an extrapolation error; the variable stepsize calculator estimates that error based on past extrapolation errors. After completion of a time step, the variable stepsize calculator compares the actual value x of the zero crossing function f with the value $\hat{x}$ that was predicted one time step earlier. The estimated extrapolation error $\hat{\epsilon}$ follows:

$$\epsilon \leftarrow \left\{ \begin{array}{ll} \alpha\hat{\varepsilon} + (1-\alpha)\,|x - \hat{x}| & \text{if } \hat{\varepsilon} > |x - \hat{x}| \\ |x - \hat{x}| & \text{otherwise} \end{array} \right\} \tag{1}$$

For example, it decreases slowly ($\alpha = 0.7$) with a first order IIR-filter rule when the extrapolation error becomes smaller, and rises abruptly to the actual value when the extrapolation error becomes larger.

C.3.4 Estimation of the number of timesteps to a zero crossing

The variable stepsize calculator (conservatively) estimates the number of timesteps n to hit the predicted zero crossing $f(t_{ZC}) = 0$ at time t_{ZC}, when starting from the current time t (with $t \leq t_{ZC}$) and when keeping the current stepsize Δt constant, to:

$$n = \frac{t_{ZC} - t}{\Delta t} \cdot \frac{1}{1 + \hat{\varepsilon} + \sigma} \tag{2}$$

where $\hat{\varepsilon}$ is the estimated extrapolation error and σ the (additional) level of conservatism optionally specified by the attribute `safety` in the JSON config file.

The rationale of this equation is that the left term predicts the zero crossing exactly when the zero crossing function f is, in the case of first order extrapolation, a straight line, or, in the case of second order extrapolation, a straight line or second order parabola. An extrapolation error generally occurs for all other functions f, with the danger of overestimating n and thus potentially choosing a too large stepsize (that steps over the zero crossing with the consequence that the tolerance of the zero crossing may be violated). Therefore, n is conservatively underestimated. The degree of this conservatism is defined by the second term and depends on both the (time-varying) estimated extrapolation error $\hat{\varepsilon}$ and the (constant) value of the safety attribute σ.

C.3.5 Detection of unstable oscillations

Unstable oscillations around the zero crossing are detected by monitoring the last three data points and checking whether these lie on alternating sides of the zero crossing and increase in absolute value.

C.3.6 Stepsize adjustment strategy

The chosen stepsize Δt is in most cases determined by a factor ρ that is multiplied with the previous stepsize Δt_{prev} (and saturated to lie within the specified stepsize interval). The stepsize is said to be *adjusted to hit* the zero crossing when $\rho = n$ (for $n \leq 1$). The stepsize is said to be *tightened* when $\rho = TIGHTENING\_FACTOR$. The stepsize is *held constant*, when $\rho = 1$. The stepsize is said to be *relaxed* when $\rho = RELAXATION\_FACTOR$. The stepsize is said to be *strongly relaxed* when $\rho = STRONG\_RELAXATION\_FACTOR$. The default values for these factors are listed in Table 5.

Table 5: Default values for the stepsize adjustment factors.

$TIGHTENING\_FACTOR$	0.5
$RELAXATION\_FACTOR$	1.2
$STRONG\_RELAXATION\_FACTOR$	3.0

By inspecting the last two data points, the direction of the simulated trajectory with respect to the zero crossing can be either:

- *distancing zero crossing*,
- *approaching zero crossing* or
- *crossed zero*.

When *distancing a zero crossing*, the stepsize is *strongly relaxed*.

When *approaching a zero crossing*, the current value of the zero crossing function, $f(t)$, is compared to the value of the absolute tolerance, *abstol*.

If:

$$|f(t)| \leq abstol \cdot TOLERANCE\_SAFETY\_FACTOR \tag{3}$$

where $TOLERANCE\_SAFETY\_FACTOR \leq 1.0$ and a default value of 0.5, then $f(t)$ is said to be *well within tolerance*, and the stepsize is *relaxed* (the zero crossing has not yet occurred but is already precisely resolved).

If

$$|f(t)| \leq abstol \tag{4}$$

then $f(t)$ is said to be *within tolerance*, and the stepsize is *held constant* (the zero crossing has not yet occurred but is already resolved).

If

$$|f(t)| > abstol \tag{5}$$

then $f(t)$ is said to be *outside tolerance*, and the (conservatively) estimated value for the number of timesteps to hit the predicted zero crossing, n, is considered.

- If $n \leq 1$, then the stepsize is *adjusted to hit* the zero crossing.
- If $1 < n \leq \delta_{tighten}$, then the stepsize is *tightened*.
- If $\delta_{tighten} < n \leq \delta_{relax}$, then the stepsize is *held constant*.
- If $\delta_{relax} < n \leq \delta_{stronglyrelax}$, then the stepsize is *relaxed*.
- If $\delta_{stronglyrelax} < n$, then the stepsize is *strongly relaxed*.

The default values of the parameters δ_i are listed in Table 6.

Table 6: Default values of the distance bin separators for the number of timesteps to hit a predicted zero crossing.

$\delta_{tighten}$	1.8 $(= 1.5 \cdot RELAXATION\_FACTOR)$
δ_{relax}	3.0 $(= STRONG\_RELAXATION\_FACTOR)$
$\delta_{stronglyrelax}$	30.0 $(= 10.0 \cdot \delta_{relax})$

When the simulated trajectory *crossed zero* in the previous time step, it is checked whether or not unstable oscillations around the zero crossing are building up.

- If unstable oscillations occur, and $f(t)$ is *well within tolerance*, then the stepsize is *held constant*.
- If unstable oscillations occur, and $f(t)$ is *within tolerance*, then the stepsize is *tightened*.
- If unstable oscillations occur, and $f(t)$ is *outside tolerance*, then the stepsize is set to its minimal value.
- If unstable oscillations do not occur, and $f(t)$ is *well within tolerance*, then the stepsize is *relaxed*.

- If unstable oscillations do not occur, and $f(t)$ is *within tolerance*, then the stepsize is *held constant*.
- If unstable oscillations do not occur, and $f(t)$ is *outside tolerance*, then the stepsize is *tightened*.

The final three reactions (when unstable oscillations do not occur) are somewhat conservative, with the intention of discouraging possible oscillations around the zero crossing from developing. Therefore, the stepsize immediately after the zero crossing is kept small. Altogether, these are the 14 possible reactions of the variable step size calculator to exhaustively handle a zero crossing constraint.

C.4 Bounded Difference Constraints

A bounded difference constraint is 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 stepsize is reduced). For the definition of a bounded difference constraint in the JSON file, see Section B.6.2.

The capability to impose a bounded difference can be useful in co-simulation, for instance, in the calculation of the heat exchange between model A of temperature T_A and model B of temperature T_B. Here, at least one of the models must calculate the heat flow, which is a function of both T_A and T_B. The model that calculates the heat flow, say model A, knows its own temperature T_A but only has a view, $T_{B,view}$, on model B's true temperature T_B. To bound the error of the calculated heat flow, a bounded difference between T_B and $T_{B,view}$ is imposed.

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 stepsize that keeps the difference bounded).

To choose the next stepsize, the current absolute and relative differences between the minimal and maximal values, δ_A and δ_R, are calculated and compared to the absolute and relative tolerances, ε_A and ε_R, respectively. Based on this comparison, the absolute and relative differences are each assigned to one of five distance bins. The bins are determined with the safety

factor:

$$\sigma = \frac{1}{1 + safety}, \tag{6}$$

where $i = A, R$, and with the default values of the parameters

$$\alpha_{SAFE} \leq \alpha_{TARGET} \leq \alpha_{RISKY} \leq 1 \tag{7}$$

listed in Table 7.

Table 7: Default values of the parameters used in the distance bin assignment of the bounded difference algorithm.

α_{RISKY}	0.6
α_{TARGET}	0.4
α_{SAFE}	0.2

If:

- $\delta_i > \varepsilon_i$, then the difference i is assigned to the VIOLATION bin.
- $\varepsilon_i \geq \delta_i > \varepsilon_i \sigma \alpha_{RISKY}$, then the difference i is assigned to the RISKY bin.
- $\varepsilon_i \sigma \alpha_{RISKY} \geq \delta_i > \varepsilon_i \sigma \alpha_{TARGET}$, then the difference i is assigned to the TARGET bin.
- $\varepsilon_i \sigma \alpha_{TARGET} \geq \delta_i > \varepsilon_i \sigma \alpha_{SAFE}$, then the difference i is assigned to the SAFE bin.
- $\varepsilon_i \sigma \alpha_{SAFE} \geq \delta_i$, then the difference i is assigned to the SAFEST bin.

Of the two assigned distance bins, the less safe one (the one ranking higher in the bullet list above) is chosen. If this distance bin is the

- VIOLATION bin, then the stepsize is *strongly tightened*.
- RISKY bin, then the stepsize is *tightened*.
- TARGET bin, then the stepsize is *held constant*.
- SAFE bin, then the stepsize is *relaxed*.
- SAFEST bin, then the stepsize is *strongly relaxed*.

A *strongly tightened* stepsize means that $\delta = STRONG\_TIGHTENING\_FACTOR$ with default value 0.01 is multiplied with the previous stepsize $(\Delta t)_{prev}$ to obtain the next stepsize Δt. The meaning of the other stepsize adjustments is analogous to the implementation of the zero crossing algorithm (see Section C.3.6). The chosen stepsize is saturated to the stepsize interval.

This algorithm for the bounded difference handler tries to adjust the stepsize such that it is kept within the TARGET bin throughout the simulation. Because a variable stepsize calculator in a co-simulation cannot (efficiently) obtain the stepsize through an iterative approach, it needs to make fairly sure that the stepsize it selects does not lead to a tolerance violation. The stepsize calculation must therefore be somewhat conservative, which is essentially manifested in the RISKY bin as a buffer between the TARGET and VIOLATION bins.

On the safe side of the TARGET bin, two bins must exist. The SAFE bin has an associated relaxation factor that is small enough so that a stepsize relaxation should not lead to an overshoot of the bound difference beyond the TARGET bin in the next time step. The SAFEST bin has an associated strong relaxation factor that is equal to the strong relaxation factor used by all other continuous constraints to prevent interference between continuous constraints (see Section C.7.1).

Note that the above described algorithm of the Bound Difference handler is extended below to prevent interference by discrete events (see Section C.7.3).

C.5 Sampling Rate Constraints

A sampling rate constraint is a discrete constraint. It constrains the stepsize such that repetitive, predefined time instants are exactly hit. This can be useful in co-simulation, for instance, when a modeled control unit reads a sensor value every x milliseconds. For the definition of a sampling rate constraint in the JSON file, see Section B.6.3.

The chosen stepsize is either the time difference between the current time and the time instant of the next sampling, or the maximal stepsize, whichever is smaller. Note that the minimal stepsize may be violated to hit a sampling event.

C.6 FMU Max Step Size Constraints

The FMU Max Step Size constraint limits the step size to the value returned from an FMU if the function is supported by the FMU. A proposal is underway to extend the FMI standard with the procedure:

```
fmi2Status fmi2GetMaxStepSize(fmi2Component c, fmi2Real *
    maxStepSize);
```

This means that an FMU can report in advance the maximal stepsize that it will accept in the next time step. The variable stepsize calculator queries all FMUs for these stepsizes and uses the minimum of the reported values as upper bound for the next stepsize. The implementation in the COE is based on the principle presented in [BBG+13, CLB+16] for *Master-Step With Predictable Step Sizes*. To the authors knowledge, this feature is implemented in FMUs exported from the tools: 20-sim, OpenModelica and Overture.

C.7 Interference between constraint handlers

When multiple constraints are present, their handlers may interfere with each other in the sense that one constraint may become active only because another one has been active in the previous step. Measures are taken to counter such interference.

C.7.1 Interference between continuous constraints handlers

Interference between continuous constraint handlers occurs when:

1. In one time step, Constraint A is active (i.e. constrains the stepsize);
2. In the next time step, the handler for Constraint A relaxes the stepsize by a factor $\rho_A > 1$, and
3. Constraint B becomes active – not because its handler protects against a potential violation, but only because it cannot relax the stepsize by more than a factor $\rho_B < \rho_A$.

To prevent such interference, all continous contraints must have the same value for their respective maximal relaxation factors. Therefore, in the implementation of the variable stepsize calculator, $STRONG\_RELAXATION\_FACTOR$ is the maximal relaxation factor for both Zero Crossing and Bounded Difference constraints and defined in the scope of the whole

calculator – not in the scope of individual constraints (as other factors are). When constraints *relax strongly*, $STRONG\_RELAXATION\_FACTOR$ is used[5].

C.7.2 Interference between discrete constraints handlers

Discrete constraints handlers base their stepsize requirements on independent time instants and therefore do not interfere with each other.

C.7.3 Interference between discrete and continuous constraint handlers

When a discrete constraint handler has limited the stepsize in the previous step, the question arises how a continuous constraint handlers shall proceed with its calculation of the next stepsize. The situation that shall be avoided is this: all continuous constraint handlers would allow a large stepsize, but a discrete constraint handler enforces a sudden, strong reduction of the stepsize. In the steps that follow, there are no discrete events, but the continuous constraint handlers require potentially many steps to repeatedly *strongly relax* the stepsize until it becomes large again.

The solution to this problem is different for Zero Crossing and Bounded Difference constraint handlers.

Extension of the Zero Crossing handler To prevent the above described undesired situation, Zero Crossing handlers calculate the next stepsize based on the last stepsize *that was not limited by a discrete constraint.*

To be precise, a Zero Crossing handler uses the previous data points irrespective of the previously active constraints to calculate the extrapolation. But when it calculates the next stepsize, it discards all previous stepsizes that were limited by a discrete constraint and chooses the last stepsize that was limited by a continuous constraint. With the thus chosen previous stepsize (and the result of the extrapolation), the handler calculates the factor ρ that is multiplied to the chosen previous stepsize in order to obtain the next

[5]Strictly speaking, when all continuous constraints *relax strongly* with the same relaxation factor, they all become active. The important point is that none of them slows down the relaxation process unnecessarily by relaxing less than the others.

stepsize. With this approach, introduced discrete events do not markedly affect the tightening and relaxation of the stepsize selected by a Zero Crossing handler.

This approach is safe, in the sense that a zero crossing should not be crossed prematurely, for two reasons. First, introduced discrete events always shorten the stepsize when approaching the zero crossing, which is conservative. Second, the assumed previous stepsize may be larger than the true previous stepsize (that was limited by a discrete constraint handler), but this does no harm: The calculation of the next stepsize is based on the number of timesteps to the predicted zero crossing, n, with the assumption that the (assumed) previous stepsize is held constant. When the previous stepsize is larger, n becomes smaller, favoring a stronger tightening of the next stepsize in particular close to the zero crossing, where the stepsize is *adjusted to hit*.

Essentially, the Zero Crossing handler can safely ignore previous stepsizes that were limited by discrete constraints because it needs to hit a time *instant* (i.e. the zero crossing) and that time instant does not depend on the previous stepsizes (time *differences*). The situation is different for the Bounded Difference handler.

Extension of the Bounded Difference handler Whereas the Zero Crossing handler needs to hit a time *instant* (i.e. the zero crossing) that does not depend on the previous stepsizes (time *differences*), the Bounded Difference handler needs to limit a value difference that does depend on the stepsize. When the Bounded Difference handler notices that the previous stepsize was limited by a discrete constraint, it may proceed in either of two ways.

First, the Bounded Difference handler could simply go forward as usual (i.e. it calculates the next stepsize by scaling the previous stepsize by the factor that is associated with the determined diffe-rence bin). Because the previous stepsize was limited by a discrete event and was therefore shorter than the stepsize that the Bounded Difference handler would have chosen, this strategy will frequently lead to the stepsize being *relaxed* or *strongly relaxed*.

Second, the Bounded Difference handler could take the last stepsize that was limited by a continuous constraint and repeat the decision it made then *on that stepsize*. To prevent that a repeated decision overly relaxes the stepsize, the repeated decision will *hold* the stepsize *constant* whenever the past decision was to *relax* or *strongly relax* it. To prevent that a repeated decision overly tightens the step-size, the chosen next stepsize may never be

smaller than the one obtained with the above (usual) strategy.

By default, the second strategy is enabled. However, in rare cases that strategy may lead to a tolerance violation (a chain of discrete events could carry a past decision to *hold* the stepsize *constant* through time; when the chain of discrete events stops, the stepsize will be *held constant* in the next step but it might have needed to be *tightened* instead). Therefore, it is possible to disable the second strategy by setting the optional attribute `"skipDiscrete"` to `false` in the definition of the Bounded Difference constraint in the JSON configuration file (see Section B.6.2).

When the second strategy is disabled, an active discrete constraint will likely reduce the next stepsize(s) proposed by the Bounded Difference constraint handler, potentially reducing efficiency.

C.8 Logging

The variable stepsize calculator writes to the same log as the COE.

When a step is taken with maximal stepsize, the variable stepsize calculator produces no log output.

When a step is taken with a less than maximal stepsize, the variable stepsize calculator logs the identifiers of the active constraints and the action of their handlers. For instance, a log entry would read

```
Time 0.9499999999999998, stepsize 0.09, limited by constraint
"bd" with decision to hold the stepsize constant
(absolute difference within target range)
```

When all continuous constraints relax strongly, the log entry does not list all constraints but is shortened to:

```
Time 5.000458745644138, stepsize 9.536808544011453E-4,
all continuous constraint handlers allow strong relaxation}
```

When a Zero Crossing constraint handler detects a zero crossing, it produces a log entry which would read:

```
A zerocrossing of constraint "zc" occurred in the time interval
[ 14.999971188014648 ; 15.000117672389647 ] and was hit
with a distance of 0.18103104302103257
```

When the variable stepsize calculator detects that a constraint has been violated in the previous step, it logs a warning. For instance, such a warning would read:

```
Absolute tolerance violated!
      | A zerocrossing of constraint "zc"
      | occurred in the time interval [ 4.998123597131701 ;
        5.008123597131701 ]
      | and could only be resolved with a distance of
        11.789784201164633
      | which is greather than the absolute tolerance of 1.0
      | The stepsize equals the minimal stepsize of 0.01 !
      | Decrease the minimal stepsize
      or increase this constraint's tolerance}
```

D COE Program properties

The COE program flow can be changed using the following Java properties which can be set on program launch using `-D` followed by the property and a value can be specified after an equal sign:

`simulation.program.delay.enable` If this property is set to `true` then the COE will interpret the time step size in seconds and make sure that the steps are at least separated by a program delay of that time step size. It only introduces a delay if the execution of `doStep` on all FMU instances are faster than the requested time step size.

`fmi.instantiate.with.empty.authority` This inserts an empty authority into the URI sent to the FMU in `instantiate`. E.g. `file:/` will become `file:///`. This can be necessary for faulty FMU implementations.

`coe.fmu.custom.factory` This must be given a fully qualified name to a class which implements `org.intocps.orchestration.coe.IFmuFactory` if set this class will be instantiated and ask to handle FMU creation before the internal factory. This can thus be used to override the default behaviour.

`coe.livestream.filter` Limits the time resolution on the values send using live logging. The values will only be send at the given interval or greater.

`simulation.parallelise.resolveinputs` Run all per instance input actions in parallel.

`simulation.parallelise.setinputs` Run all per instance set inputs in parallel.

simulation.parallelise.dostep Run all per instance `doSte` calls in parallel.

simulation.parallelise.obtainstate Run all per instance calls to obtain a global state in parallel.

simulation.profile.executiontime Log execution time for the main calls involved in performing a global step.

Note that enabling parallel execution may not give faster simulations it is highly dependent on the number of instances, the amount of inputs/outputs and the actual execution to of the `doStep` call per instance.

E Performance Test FMU

```
within ;
model Snail "slow and useless, heats your PC"
  import Modelica.Constants.eps;

  Modelica.Blocks.Interfaces.RealInput u annotation (Placement(
      transformation(extent={{-140,-20},{-100,20}})));
  Modelica.Blocks.Interfaces.RealOutput y annotation (Placement
      (transformation(extent={{100,-10},{120,10}})));


  parameter Integer nLoop = 10 "array size" annotation(Evaluate
      =false);

protected
  Real uPos;


algorithm

  uPos := abs(u);
  y := 0;

  for i in 1:nLoop loop
    y := y + (-1).^i * exp( atan2(uPos, log(uPos + eps)) / (i *
        sqrt(uPos + eps)));
  end for;

  annotation (
    Icon(coordinateSystem(preserveAspectRatio=false)),
    Diagram(coordinateSystem(preserveAspectRatio=false)),
    uses(Modelica(version="3.2.2")));
```

```
end Snail;
```