Grant Agreement: 644047

INtegrated TOol chain for model-based design of CPSs

Design of the INTO-CPS Platform

Deliverable Number: D4.1d

Version: 1.00

Date: 2015

Public Document

http://into-cps.au.dk

Contributors:

Kenneth Lausdahl, AU
Peter Gorm Larsen, AU
Sune Wolf, AU
Anders Terkelsen, AU
Victor Bandur, AU
Miran Hasanagić, AU
Casper Thule Hansen, AU
Ken Pierce, UNEW
Oliver Kotte, TWT
Adrian Pop, LIU
Etienne Brosse, ST
Jörg Brauer, VSI
Oliver Möller, VSI

Editors:

Kenneth Lausdahl, AU

Reviewers:

Ken Pierce, UNEW
Nuno Amálio, UY
Alie El-Din Mady, 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	14-01-2015	Kenneth Lausdahl	Initial document version
0.2	13-05-2015	Peter Gorm Larsen	Full structure of the deliverables and responsibilities
0.3	22-05-2015	Carl Gamble	Added section on provenance and traceability
0.4	04-06-2015	Anders Terkelsen	Added section on release management
0.5	04-06-2015	Miran Hasanagić	Added initial information about the required extension for the FMI standard
0.6	24-09-2015	Carl Gamble	Content added to DSE, and Provenance and Traceability sections
0.7	01-10-2015	Oliver Kotte	Added section and appendix on the variable stepsize calculator
0.8	29-10-2015	Kenneth Lausdahl	Updated Overview of the INTO-CPS Platform
0.9	30-10-2015	Kenneth Lausdahl	Added design content to the COE
0.91	30-10-2015	Casper Thule Hansen	Added Performance Optimization to the COE
0.92	03-11-2015	Victor Bandur	Updated model checking protocol description
0.93	04-11-2015	Kenneth Lausdahl	Added initial Related work section
0.94	06-11-2015	Victor Bandur	Completed the related work section
0.95	06-11-2015	Kenneth Lausdahl	Prepared document for internal review resolving missing cross referencing
0.96	27-11-2015	Kenneth Lausdahl	Corrected review comments
0.97	05-12-2015	Peter Gorm Larsen	Added conclusions
0.98	14-12-2015	Kenneth Lausdahl	Corrected layout
0.99	15-12-2015	Kenneth Lausdahl	Updated Appendix for intermediate model definition format
1.00	15-12-2015	Adrian Pop	Added an FMI section

Abstract

This deliverable contains the technical design documentation of the INTO-CPS platform at the end of the first year of the project. This is focussed around the Co-simulation Orchestration Engine (COE) and the integration of a number of existing simulation tools making use of the Functional Mockup Interface (FMI). However, the INTO-CPS tool chain contains a lot of other features as well. These range from requirements and graphical overviews of CPS elements in different views using SysML down to realisation of CPSs. Full traceability from the initial requirements as well as configuration management for all the different artefacts. Additional features enable Design Space Exploration (DSE), Test Automation (TA) and Model Checking (MC).

Contents

1 Introduction

This deliverable contains the technical design documentation of the INTO-CPS tool chain platform. Since the project has two Work Packages (WPs) dedicated to the development of tool features this deliverable will provide an overview of the entire tool chain, and provide references to the deliverables that explain the extension features for the INTO-CPS tool chain. This deliverable mainly focuses on the Co-simulation Orchestration Engine (COE) which implements a master algorithm for the Functional Mock-up Interface (FMI) standard version 2.0 [Blo14]. However, external input to the FMI standard such as [BBG$^+$13a] is also taken into account and the COE implements similar algorithms described in more detail in Section 6.3. The COE itself is a tool that performs co-simulations using a collection of FMUs including together with a co-simulation configuration, which describes how the FMUs relate to each other.

After this introduction Section 2 introduces the key parts of FMI. Section 3 provides an overview of the related work in particular with focus on co-simulation. Afterwards Section 4 provides a graphical overview of the different features in the INTO-CPS tool chain and indicates the interfaces between each of them. This is followed by Section 5 which describes the most likely use scenarios of the INTO-CPS tool chain.

The remaining sections introduce the designs of the different features in the INTO-CPS tool chain. The INTO-CPS Application described in Section 6.1 provides the main co-simulation user interface; Section 6.2 describe how SysML support is provided in the INTO-CPS tool chain; Section 6.3 describes the design of the COE and Section 6.4 how the baseline simulation tools are integrated with the COE using FMI. Afterwards the different extensions of the INTO-CPS platform are presented in a number of sections. Here Section 6.5 contains the top-level design of the Design Space Exploration (DSE) feature [GHJ$^+$15]. This is followed in Section 6.6 that provides a top level description of the test automation feature in an INTO-CPS context [MPB15]. Section 6.7 explains an overview of model-checking in the INTO-CPS context [BLL$^+$15]. Then Section 6.8 provides an indication how the users are intended to make use of code generation in an INTO-CPS context and refers to the deliverable where the principles for this are defined in more detail [HLG$^+$15]. Section 6.9 shows the plans for making use of provenance and traceability of artefacts in an INTO-CPS setting.

Section 7 describes the release mechanisms for the INTO-CPS tool including its independent baseline tools in their updated versions that fit together.

Section 8 concludes the work carried out during the first project year. Finally, the appendices: Appendix A described the COE protocol, Appendix B describes the variable stepsize calculation used in the COE, Table C lists the FMI functions uses by the COE, and Appendix D describes intermediate model description format exported from Modelio.

2 The Functional Mock-up Interface (FMI)

The Functional Mock-up Interface (FMI) [Blo14] is an open standard for exporting dynamic models on the system and component level either for model exchange or co-simulation. FMI gives model-based systems engineering the ability to share models between tools. More than 60 commercial or open-source simulation tools are now supporting the FMI standard.

In this section we give a brief overview of the FMI standard, for a more detailed description please consult [Blo14].

2.1 Overview

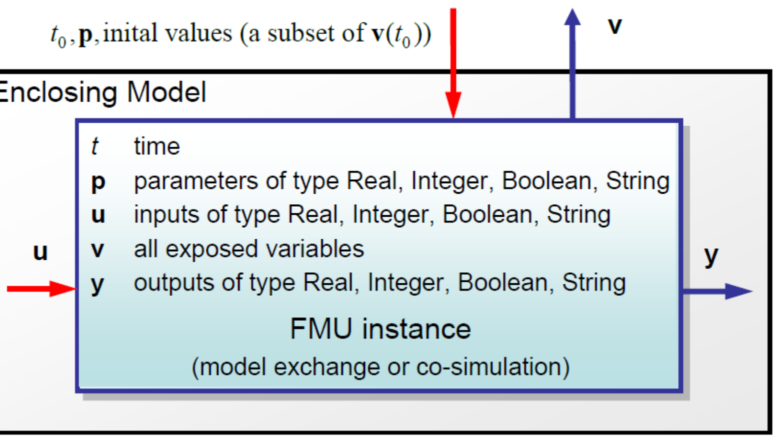
The FMI (Functional Mock-up Interface) defines an interface to be implemented by an executable called FMU (Functional Mock-up Unit). The FMI functions are called by a simulation environment (in the INTO-CPS context the COE) to create one or more instances of the FMU and to simulate them, typically together with other models. An FMU may either have its own solvers (FMI for Co-Simulation) or require the simulation environment to perform numerical integration (FMI for Model Exchange). See Figure 1.

In the INTO-CPS project we are only interested in the FMI standard version 2.0 and the FMI for Co-Simulation part of the standard.

The *FMI for Co-Simulation* interface is designed both for the coupling of simulation tools (simulator coupling, tool coupling), and coupling with subsystem models, which have been exported by their simulators together with its solvers as runnable code.

The FMI for co-simulation FMUs are of two types:

- standalone (not needing a tool to work) - Figure 2
- tool wrapper (need a tool to run) - Figure 3



t_0, $\mathbf{p}$, inital values (a subset of $\mathbf{v}(t_0)$)
v
Enclosing Model
t time
p parameters of type Real, Integer, Boolean, String
u inputs of type Real, Integer, Boolean, String
v all exposed variables
y outputs of type Real, Integer, Boolean, String
u
y
FMU instance
(model exchange or co-simulation)

Figure 1: FMU interaction with the environment; inputs are in red, outputs are in blue

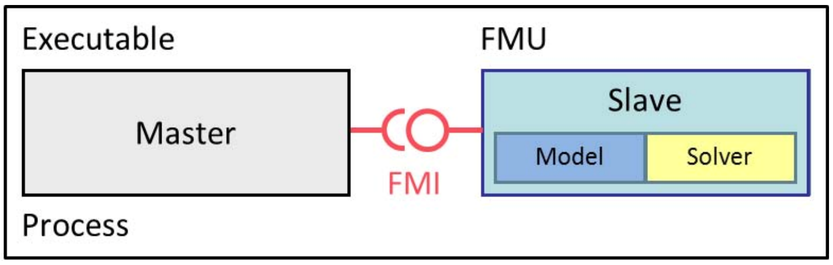


Figure 2: Co-simulation with an standalone FMU

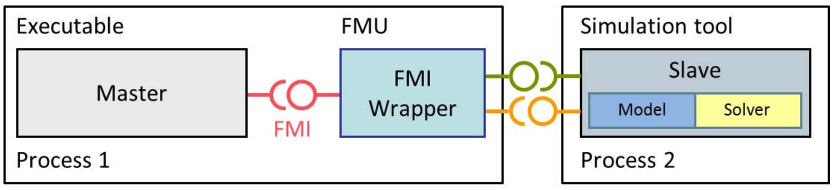


Figure 3: Co-simulation with an FMI tool wrapper FMU

In its most general form, a tool coupling based co-simulation is implemented on distributed hardware with subsystems being handled by different computers with maybe different OS (cluster computer, computer farm, computers at different locations). The data exchange and communication between the subsystems is typically done using one of the network communication technologies (i.e TCP/IP). The definition of this communication layer is not part of the FMI standard. Distributed co-simulation scenarios can be implemented using FMI and the master has to implement the communication layer.

2.2 FMI for Co-Simulation Computational Model

FMI for Co-Simulation provides an interface standard for the solution of time dependent coupled systems consisting of subsystems that are continuous in time (model components that are described by instationary differential equations) or time-discrete (model components that are described by difference equations like, for example discrete controllers). In a block representation of the coupled system, the subsystems are represented by blocks with (internal) state variables x(t) that are connected to other subsystems (blocks) of the coupled problem by subsystem inputs u(t) and subsystem outputs y(t). In this framework, the physical connections between subsystems are represented by mathematical coupling conditions between the inputs u(t) and the outputs y(t) of all subsystems.

During time integration, the simulation is performed independently for all subsystems (FMUs) restricting the data exchange between subsystems to discrete *communication points*. The term "communication point" in FMI for Co-Simulation refers to the communication between subsystems in a co-simulation environment and should not be mixed with the output points for saving simulation results to file. The data flow (input and output for an FMU) at communication points is given in Figure 4.

For co-simulation two basic groups of functions have to be realized:

- functions for the data exchange between subsystems
- functions for algorithmic issues to synchronize the simulation of all subsystems and to proceed in communication steps from initial time to end time

In FMI for Co-Simulation both functions are implemented in one software component, the co-simulation master. The data exchange between the sub-

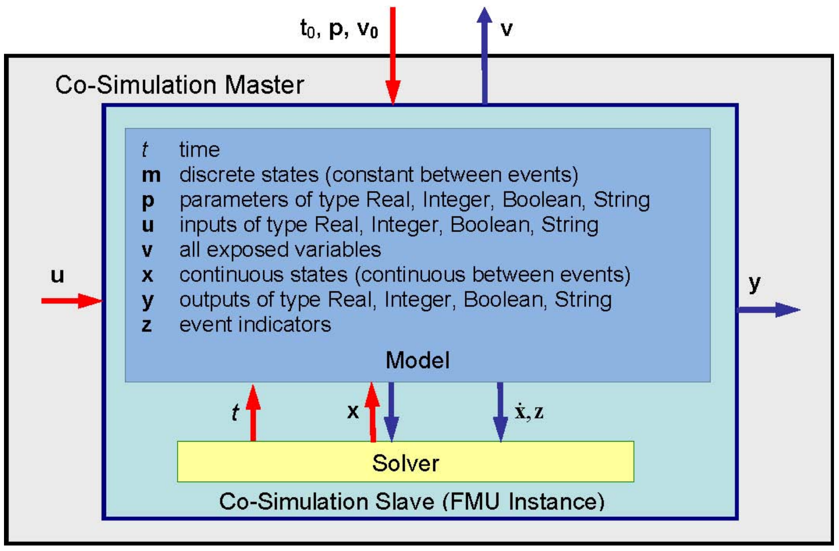

t_0, p, v_0
v
Co-Simulation Master
t time
m discrete states (constant between events)
p parameters of type Real, Integer, Boolean, String
u inputs of type Real, Integer, Boolean, String
v all exposed variables
x continuous states (continuous between events)
y outputs of type Real, Integer, Boolean, String
z event indicators
Model
u
y
t
x
$\dot{x}$, z
Solver
Co-Simulation Slave (FMU Instance)

Figure 4: FMI for co-simulation FMU input and output at communication points

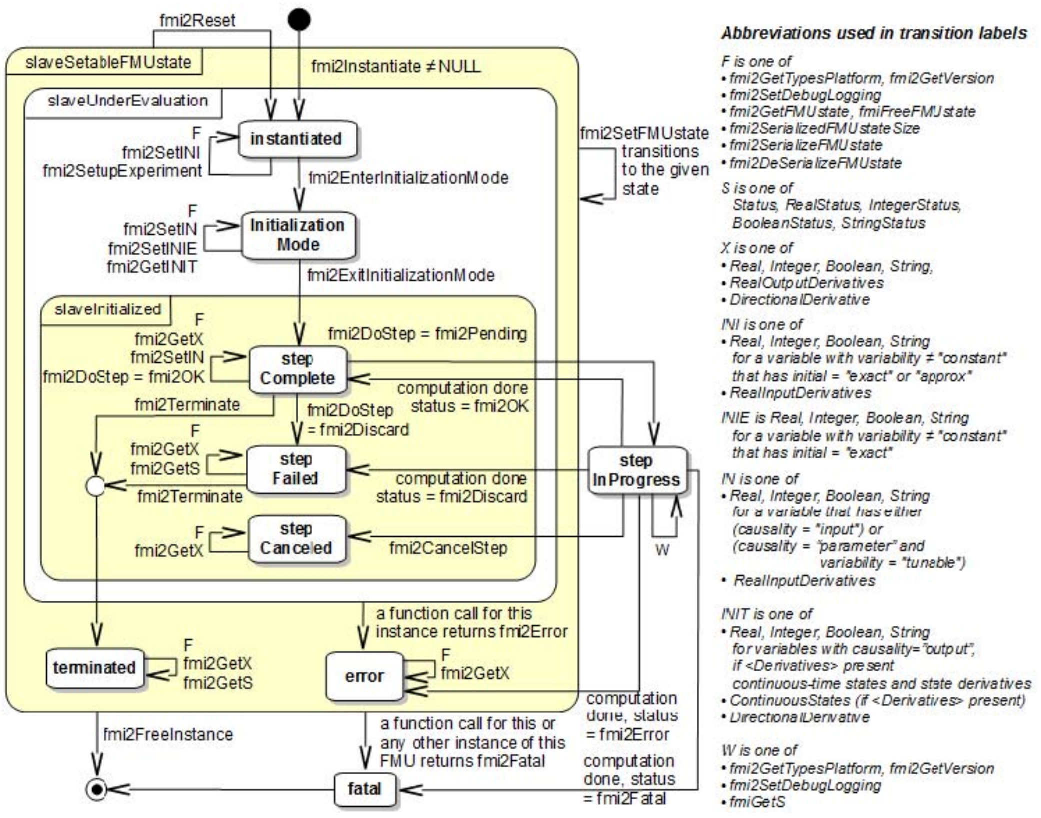
systems (slaves) is handled via the master only. There is no direct communication between the slaves. In the INTO-CPS project the master is the COE.

The state machine of calling an FMI for co-simulation FMU from the master is given in Figure 5

2.3 FMI Distribution

An Functional Mock-up Unit (FMU) model is distributed as a zip file with extension `.fmu` containing several files and directories. The zip file contains:

- The FMI Description File (in XML format).
- The C sources of the FMU, including the needed run-time libraries used in the model, and/or binaries for one or several target machines, such as Windows dynamic link libraries (.dll) or Linux shared object libraries (.so). The latter solution is especially used if the FMU provider wants to hide the source code to secure the contained know-how or to allow a fully automatic import of the FMU in another simulation environment. An FMU may contain physical parameters or geometrical dimensions, which should not be open.



Abbreviations used in transition labels

F is one of
- *fmi2GetTypesPlatform, fmi2GetVersion*
- *fmi2SetDebugLogging*
- *fmi2GetFMUstate, fmiFreeFMUstate*
- *fmi2SerializedFMUstateSize*
- *fmi2SerializeFMUstate*
- *fmi2DeSerializeFMUstate*

S is one of
Status, RealStatus, IntegerStatus, BooleanStatus, StringStatus

X is one of
- *Real, Integer, Boolean, String,*
- *RealOutputDerivatives*
- *DirectionalDerivative*

INI is one of
- *Real, Integer, Boolean, String*
  for a variable with variability ≠ "constant"
  that has initial = "exact" or "approx"
- *RealInputDerivatives*

INIE is Real, Integer, Boolean, String
for a variable with variability ≠ "constant"
that has initial = "exact"

IN is one of
- *Real, Integer, Boolean, String*
  for a variable that has either
  (causality = "input") or
  (causality = "parameter" and
  variability = "tunable")
- *RealInputDerivatives*

INIT is one of
- *Real, Integer, Boolean, String*
  for variables with causality="output",
  if <Derivatives> present
  continuous-time states and state derivatives
- *ContinuousStates (if <Derivatives> present)*
- *DirectionalDerivative*

W is one of
- *fmi2GetTypesPlatform, fmi2GetVersion*
- *fmi2SetDebugLogging*
- *fmiGetS*

Figure 5: FMI for co-simulation FMU calling sequence

- Additional FMU data (like tables, maps) in FMU specific file formats.

A more detailed structure of the `.fmu` file is given below:

```
// Structure of zip file of an FMU
modelDescription.xml // Description of FMU (required
   file)
model.png // Optional image file of FMU icon
documentation // Optional directory containing the FMU
    documentation
    index.html // Entry point of the documentation
    <other documentation files>
sources // Optional directory containing all C sources
    // all needed C sources and C header files to
       compile and link the FMU
    // with exception of: fmi2TypesPlatform.h ,
       fmi2FunctionTypes.h and fmi2Functions.h
    // The files to be compiled (but not the files
       included from these files)
    // have to be reported in the xml-file under the
       structure
    // <ModelExchange><SourceFiles> ... and <
       CoSimulation><SourceFiles>
binaries // Optional directory containing the binaries
    win32 // Optional binaries for 32-bit Windows
        <modelIdentifier>.dll  // DLL of the FMI
           implementation (build with option "MT" to
           include run-time environment)
        <other DLLs> // The DLL can include other DLLs
            or optional object Libraries for a
           particular compiler
        VisualStudio8 // Binaries for 32-bit Windows
           generated with Microsoft Visual Studio 8
           (2005)
            <modelIdentifier>.lib // Binary libraries
        gcc3.1 // Binaries for gcc 3.1.
            ...
        win64 // Optional binaries for 64-bit Windows
            ...
        linux32 // Optional binaries for 32-bit Linux
            <modelIdentifier>.so // Shared library of
               the FMI implementation
            ...
       linux64 // Optional binaries for 64-bit Linux
            ...
resources // Optional resources needed by the FMU
    < data in FMU specific files which will be read
       during initialization;
       also more folders can be added under resources
          (tool/model specific).
       In order for the FMU to access these resource
          files, the resource directory
       must be available in unzipped form and the
          absolute path to this directory
       must be reported via argument "
          fmuResourceLocation" via fmi2Instantiate.
   >
```

2.4 FMI Application Programming Interface

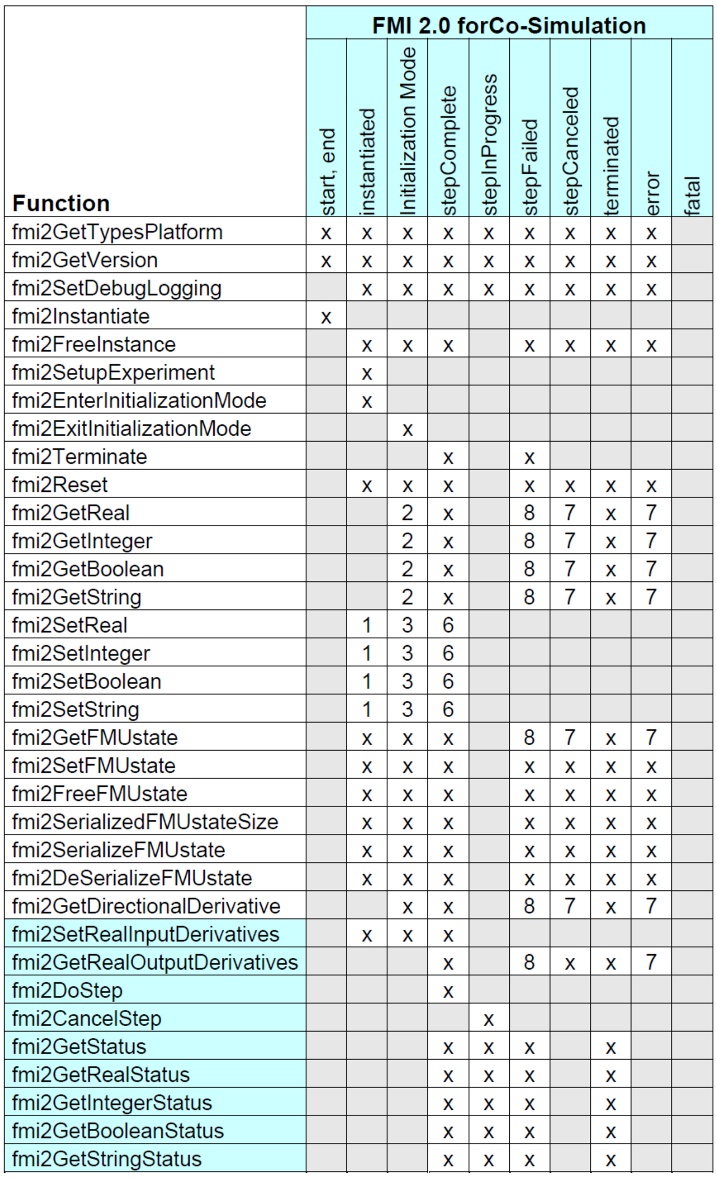
All needed equations or tool coupling computations are evaluated by calling standardized "C" functions. "C" is used, because it is the most portable programming language today and is the only programming language that can be utilized in all embedded control systems.

Three header files are provided that define the interface of an FMU:

- `fmi2TypesPlatform.h` - contains the type definitions of the input and output arguments of the functions. This header file must be used both by the FMU and by the target simulator
- `fmi2FunctionTypes.h` - contains typedef definitions of all function prototypes of an FMU. When dynamically loading an FMU, these definitions can be used to type-cast the function pointers to the respective function definition.
- `fmi2Functions.h` - contains the function prototypes of an FMU that can be accessed in simulation environments.

In all header files the convention is used that all C function and type definitions start with the prefix "fmi2"

The FMI for co-simulation API is given in Figure 6.



	FMI 2.0 forCo-Simulation									
Function	start, end	instantiated	Initialization Mode	stepComplete	stepInProgress	stepFailed	stepCanceled	terminated	error	fatal
fmi2GetTypesPlatform	x	x	x	x	x	x	x	x	x	
fmi2GetVersion	x	x	x	x	x	x	x	x	x	
fmi2SetDebugLogging		x	x	x	x	x	x	x	x	
fmi2Instantiate	x									
fmi2FreeInstance		x	x	x		x	x	x	x	
fmi2SetupExperiment		x								
fmi2EnterInitializationMode		x								
fmi2ExitInitializationMode			x							
fmi2Terminate				x		x				
fmi2Reset		x	x	x		x	x	x	x	
fmi2GetReal			2	x		8	7	x	7	
fmi2GetInteger			2	x		8	7	x	7	
fmi2GetBoolean			2	x		8	7	x	7	
fmi2GetString			2	x		8	7	x	7	
fmi2SetReal		1	3	6						
fmi2SetInteger		1	3	6						
fmi2SetBoolean		1	3	6						
fmi2SetString		1	3	6						
fmi2GetFMUstate		x	x	x		8	7	x	7	
fmi2SetFMUstate		x	x	x		x	x	x	x	
fmi2FreeFMUstate		x	x	x		x	x	x	x	
fmi2SerializedFMUstateSize		x	x	x		x	x	x	x	
fmi2SerializeFMUstate		x	x	x		x	x	x	x	
fmi2DeSerializeFMUstate		x	x	x		x	x	x	x	
fmi2GetDirectionalDerivative			x	x		8	7	x	7	
fmi2SetRealInputDerivatives		x	x	x						
fmi2GetRealOutputDerivatives				x		8	x	x	7	
fmi2DoStep				x						
fmi2CancelStep					x					
fmi2GetStatus				x	x	x		x		
fmi2GetRealStatus				x	x	x		x		
fmi2GetIntegerStatus				x	x	x		x		
fmi2GetBooleanStatus				x	x	x		x		
fmi2GetStringStatus				x	x	x		x		

x means: call is allowed in the corresponding state
number means: call is allowed if the indicated condition holds:
1 for a variable with `variability ≠ "constant"` that has `initial = "exact"` or `"approx"`
2 for a variable with `causality = "output"` or
continuous-time states or state derivatives (if element `<Derivatives>` is present)
3 for a variable with `variability≠"constant"` that has `initial="exact"`, or `causality="input"`
6 for a variable with `causality = "input"` or
`(causality = "parameter" and variability = "tunable")`
7 always, but retrieved values are usable for debugging only
8 always, but if status is other than fmi2Terminated, retrieved values are useable for debugging only

Figure 6: FMI for co-simulation C API

2.5 FMI Description Schema

All static information related to an FMU is stored in file `modelDescription.xml` in XML format.

The schema defines the structure and content of the `modelDescription.xml` XML file generated by a modeling environment. This XML file contains the definition of all variables of the FMU in a standardized way. It is then possible to run the C code in an embedded system without the overhead of the variable definition (the alternative would be to store this information in the C code and access it via function calls, but this is neither practical for embedded systems nor for large models). Furthermore, the variable definition is a complex data structure and tools should be free how to represent this data structure in their programs. The selected approach allows a tool to store and access the variable definitions (without any memory or efficiency overhead of standardized access functions) in the programming language of the simulation environment, such as C++, C#, Java, or Python.

2.5.1 Definition of an FMU (fmiModelDescription)

The root-level definition given in Figure 7 contains all the elements that can appear in the XML file. Attributes (see for example Figure 8) of the elements are storing extra data.

3 Related Work

This section discusses existing work on co-simulation of heterogeneous systems. It is divided into two parts, the first describing projects focused on co-simulation, the second discussing literature on co-simulation which is not directly associated with any of the projects identified. The projects discussed in the first part are taken from the project survey of WP3. These projects all relate to INTO-CPS in terms of co-simulation capabilities. The technologies developed either use FMI or include their own simulation protocol. Projects are identified as relevant in this context based on the following criteria:

- The technology uses the FMI standard. Projects which use a bespoke co-simulation protocol are mentioned, but not discussed in detail.
- The technology developed allows some level of heterogeneity in the pool of models forming a co-simulation. That is, the technology can cope

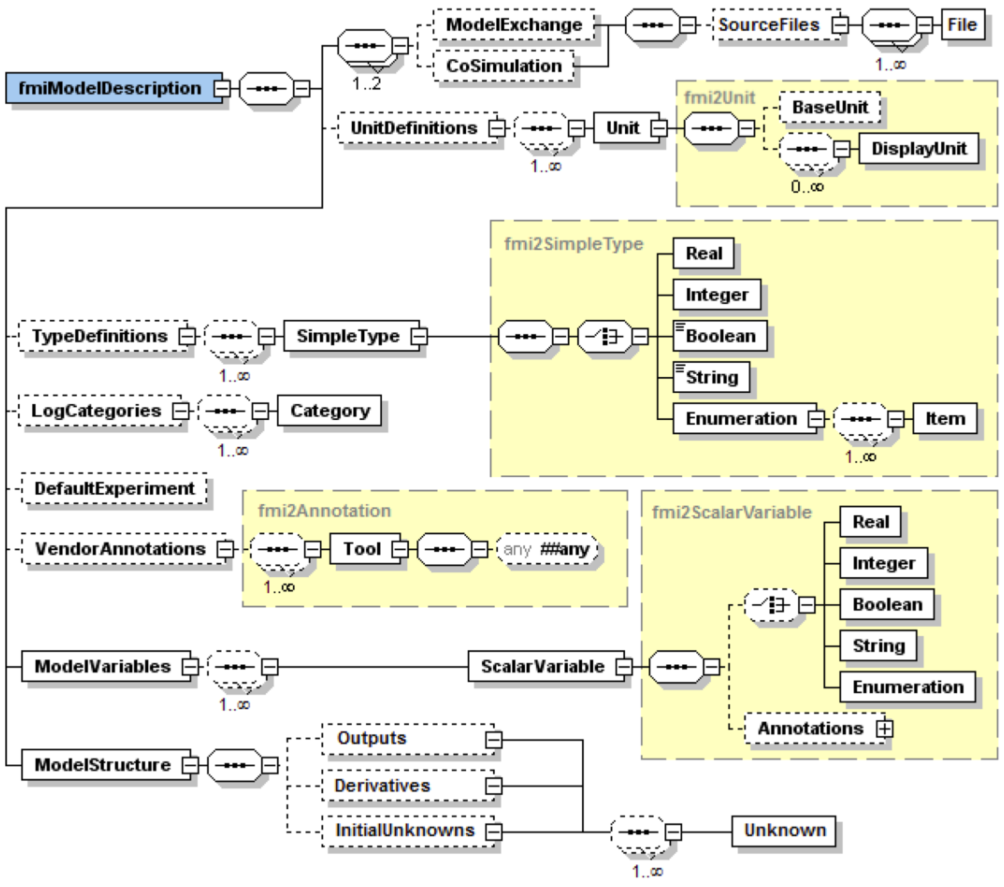


Figure 7: FMI model description element

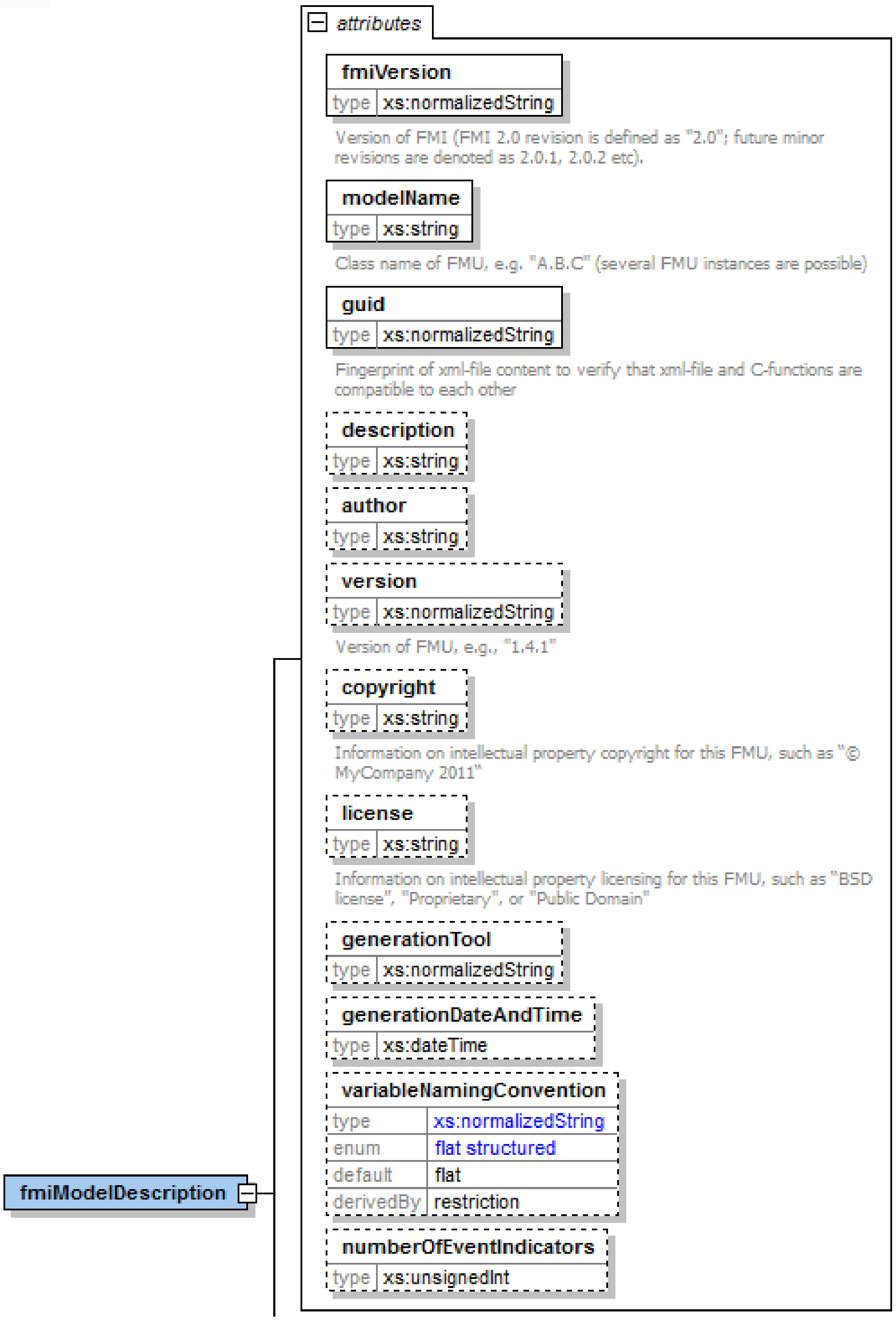

attributes
fmiVersion
type xs:normalizedString
Version of FMI (FMI 2.0 revision is defined as "2.0"; future minor revisions are denoted as 2.0.1, 2.0.2 etc).
modelName
type xs:string
Class name of FMU, e.g. "A.B.C" (several FMU instances are possible)
guid
type xs:normalizedString
Fingerprint of xml-file content to verify that xml-file and C-functions are compatible to each other
description
type xs:string
author
type xs:string
version
type xs:normalizedString
Version of FMU, e.g., "1.4.1"
copyright
type xs:string
Information on intellectual property copyright for this FMU, such as "© MyCompany 2011"
license
type xs:string
Information on intellectual property licensing for this FMU, such as "BSD license", "Proprietary", or "Public Domain"
generationTool
type xs:normalizedString
generationDateAndTime
type xs:dateTime
variableNamingConvention
type xs:normalizedString
enum flat structured
default flat
derivedBy restriction
numberOfEventIndicators
type xs:unsignedInt
fmiModelDescription

Figure 8: FMI model description attributes

with co-simulating discrete- and continuous-time systems together, as well orchestrating different simulation tools within each category. Projects which only allow a limited amount of model heterogeneity are not considered.

The remaining discussion of literature which is independent of these projects follows no particular scheme.

The INTO-CPS project is similar to most of the related projects, but also aims to support well-founded co-simulation taking all the tools in the INTO-CPS tool suite into account. The project focus on developing an efficient COE while focusing on stability. The COE described in the document, at its early state, is similar to what has been carried out in the related projects but it will be developed in line with the semantics described in Deliverable D2.1d [ACG+15]. The COE itself will be based on FMI for co-simulation making it independent on the FMU export tool oppose to some of the related projects.

3.1 Projects which Do Not Use FMI

The following projects use approaches to co-simulation that are not compliant with the FMI standard.

- The DESTECS project[1] [DES09] developed methods and tools which combine continuous time system models with discrete event controller models through co-simulation to allow multidisciplinary modelling. Co-simulation is achieved through a custom exchange protocol similar to FMI. The simulation engine can co-simulate one discrete event model using Overture and one continuous time model using 20-sim. Co-simulation uses only fixed time steps. Both tools implement extensions which allow the co-simulation orchestrator to utilize all features available in the individual tools during a co-simulation, *e.g.* debugging and 3D visualization. Automatic design-space exploration is supported via automatic permutation of parameter values in user specified ranges.
- The HYCON 2 project[2] focuses on co-simulation of heterogeneous models developed specifically in toolboxes for either Matlab or Simulink. The modelling language is the Compositional Interchange Format (CIF)[3],

[1]`http://destecs.org/`
[2]`http://www.hycon2.eu`
[3]`http://cif.se.wtb.tue.nl`

an automata-based general-purpose language for hybrid systems. The project targets specifically version 2.0 of CIF. Choice of CIF facilitates model interchange through model transformations. Interaction with external modelling tools, specifically toolboxes in Matlab and Simulink, is facilitated through the use of Simulink S-Functions. The project lists over 800 publications, but no project deliverables are available in the public domain, making it impossible to gain abstract insights into the simulation technologies developed.

- The MODESEC project[4] takes a model-driven approach to the development of CPSs, with particular emphasis on security. The project uses Ptolemy II as the implementation platform (described below). Therefore, simulation in this project benefits from the facilities of the Ptolemy II platform, but it is not the central focus.
- The SPEEDS project[5] focused on the development of heterogeneous component-based systems through functional problem decomposition based on multiple viewpoints. Heterogeneity within a system thus developed is achieved through the use of the Heterogeneous Rich Components (HRC) meta-model, which captures model interfaces independently of the individual COTS modelling tools. A co-simulation requires tool-independent exports from these tools, in the same sense as standalone FMUs, but conforming to HRC meta-model. Co-simulation of HRCs benefits from variable time steps.
- Like HYCON 2, the MULTIFORM project[6,7] also addresses the problem of model-driven component-based system design and implementation through the use of CIF. Heterogeneity is achieved in the resulting system models by allowing different modelling tools, which are concerned with different aspects of design, to work together, the same principle as in INTO-CPS.

3.2 Projects which Use FMI

The following projects make use of the FMI standard at the level of co-simulation.

[4] `http://modesec-project.eu`

[5] `http://www.speeds.eu.com`

[6] `http://www-verimag.imag.fr/MULTIFORM.html?lang=fr`

[7] The project-specific website at `http://www.multiform.bci.tu-dortmund.de/` is inaccessible, making it impossible to delve further into the project literature.

- Ptolemy II [Pto14][8] is a single-tool simulation and modelling platform which can perform simulation of heterogeneous models constructed from so-called *domains* (realized as *models of computation*), for instance, continuous time ordinary differential equation descriptions, discrete-event systems, state machines *etc.*, under the control of *directors*, which implement the control laws for each domain. The tool has the ability to import standalone FMUs, leading to a high degree of model heterogeneity through a combination of native domains and external FMUs. However, it is unclear at this time whether tool-wrapper FMUs can be co-simulated.
- The iCyPhy project[9] [FJLM14] focuses on the semantics of component interoperation, but a simulation tool based on Ptolemy II, FIDE [CLT$^+$15], achieves co-simulation of FMUs. CyPhyML provides a graphical editor for models and an API for programmatic construction of models. At this time it is impossible to gain further insight into the co-simulation technology produced under this project, as much of the relevant literature is not yet publicly available.
- The DANSE project[10] models System of System (SoS) using block diagrams and is able to export this as FMUs which can be simulated in their DESYRE environment. The project develops its own specification language, the DANSE language. In addition to simulation, the project also supports statistical model checking and optimised simulation based on metrics of interest. Both are done by reading information directly from DANSE specifications, since the FMI standard does not include the required structural information. Of note is the fact that the technology allows multiple levels of abstraction of any given model component in a simulation. The connection between the two levels is made stochastically. It is believed that allowing such multi-level abstraction makes simulations requiring high numbers of components more tractable while still yielding accurate results. In terms of simulation support, the project supports both local simulation (termed "hosted simulation"), as well as distributed co-simulation, in the sense of INTO-CPS. However, the project makes use of FMI only for hosted simulation, where essentially only standalone FMUs are co-simulated on a single host, whereas distributed co-simulation is achieved through the use of the United States Department o Deference's High-Level Ar-

[8] `http://ptolemy.eecs.berkeley.edu`
[9] `http://www.icyphy.org/index.html`
[10] `http://www.danse-ip.eu/home/index.php`

chitecture (HLA). Further information is available from the project website, as all project deliverables are publicly available.

- The CosiMate project[11] develops a co-simulation approach for heterogeneous systems which is very similar to INTO-CPS. However, it allows the connection of external simulation tools not only through FMI, but also through their native control interfaces. Addition of a new simulation tool to a co-simulation scenario is facilitated by an Eclipse-based interface construction environment. The co-simulation platform supports variable time steps.
- The ADVANCE project[12] [SBC14] allows co-simulation of Event-B machines with external continuous FMUs through FMI version 1. The resulting technologies support model-based testing and model-checking of CPS using ProB. The co-simulation capabilities of the ADVANCE MultiSim simulation framework are similar to those projected for the INTO-CPS tool chain, and are implemented as a plugin for the Rodin platform for Event-B. However, owing to the capabilities of Rodin, proof in that domain is better integrated with the relevant tool than current proof support for VDM-RT, but INTO-CPS has the main advantage that it seeks to make a co-simulation platform. The aim in INTO-CPS is to co-simulate both discrete and continuous FMUs together without knowing the details about the implementation of the FMUs, as long as they are compatible with the FMI version 2 standard. Further information is available from the project website, as all project deliverables are publicly available.

3.3 Work not Associated with the Other Related Projects

Determinate composition of FMUs for co-simulation is discussed in [BBG+13b] along with a criticism of the FMI standard concerning the level of information that a given FMU must provide. They provide two algorithms that support co-simulation, and show that the FMI version 2.0 algorithm has a high simulation cost because it is based on roll-back where state must be saved for all FMIs. The second algorithm uses a small FMI extension for obtaining the minimum next step size per FMU, and is more efficient but not FMI compliant. Both of the presented algorithms are methods to deterministically orchestrate a co-simulation with both discrete and continuous

[11] `http://site.cosimate.com`
[12] `http://www.advance-ict.eu/`

FMUs. The INTO-CPS COE presented in section Section 6.3, supports both algorithms from [BBG+13b] automatically detecting if the extension method exists. The COE is providing the necessary support for the FMUs exported by Overture [LBF+10] as described in Section 6.4.

In [NGL+14] it is demonstrated that a co-simulation with multiple FMUs potentially running with different step-sizes can be simulated on their Command and Control Wind Tunnels platform using the High-Level Architecture (HLA) framework.

A language-based approach is used in [vADVM15] for the generation of a static optimized master algorithm. The paper classifies the FMU interconnections and based on this selects the type of master algorithm. The main focus is on feedback types and requires that all input-output relations be present in the model description of all FMUs. Their solution also relies on the ability to restore state, only takes fixed time steps into account and uses cycle detection as described in [BCWS11]. However, in their solution, under certain constraints, the FMUs in a co-simulation can be simulated with different step-sizes, using interpolation if needed, to provide all inputs at the required communication points. This approach is something we do not yet support but are interested in pursuing due to the potential improvement in simulation speed for certain multi-models.

4 Overview of the INTO-CPS Platform

This section gives an overview of the structure of the INTO-CPS tool chain. First an introduction to the top-level structure is provided, centered on the main interface to the tool chain, the INTO-CPS Application. Subsequent sections describe in more detail how the design of the tool chain facilitates the different phases of the co-simulation workflow as described in Section 5 [FGPP15a].

4.1 The INTO-CPS Application

This section describes how the INTO-CPS Application interfaces with the relevant tools of the INTO-CPS tool chain as shown in Figure 9. It can be seen that the INTO-CPS Application is responsible for coordinating the simulation with the Co-Simulation Orchestration Engine, Design Space Exploration driver, and to generate test FMUs using the RT-Tester. The Modelio tool can export test models to the RT-Tester tool and co-simulation configurations to the INTO-CPS Application. The connections in the figure represent,

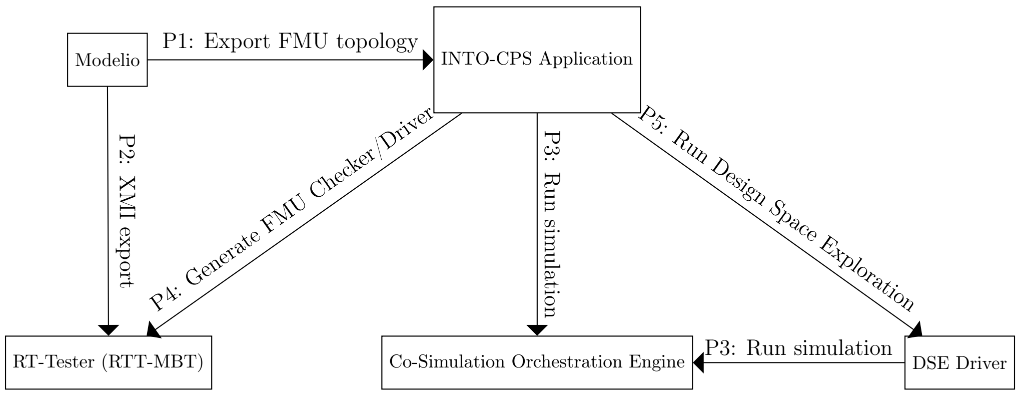


Figure 9: Overview of the INTO-CPS Application

at an abstract level, communications in the direction of the arrow. These are as follows:

P1: A protocol to extract FMU topologies from architectural models expressed in the INTO-CPS profile of SysML [APCB15]. The export format is a partial co-simulation configuration containing the FMU topology that the COE needs to perform a co-simulation.

P2: A protocol for model checking multi-models against user-defined properties.

P3: The simulation protocol documented in Appendix A. The protocol controls a simulation through actions such initialization, executing simulation steps, obtaining results and observing the status of the simulation. The protocol requires a list of FMUs, their connections, design parameters, the type of simulation algorithm and configuration that must be applied to it.

P4: This is the protocol to generate test FMUs. The protocol is command-line based and essentially asks RT-Tester to generate a number of FMU's, all with the same interface, which can then be used in a simulation invoked by the INTO-CPS Application. The RT-Tester tool must be pre-configured with a state diagram that may be modelled in Modelio and then exported via XMI for RT-Tester.

P5: A protocol used to enter the designspace exploration phase. The DSE script requires the same information regarding the co-simulation configuration as a single co-simulation, as it takes over the co-simulation in order to explore optimal configurations.

4.2 System Modelling

This section describes the interactions and interfaces necessary between the different tools such that Modelio may be used to layout a co-simulation configuration graphically using the SysML profile described in Section 6.2. Once the elements and connection topology of such a multi-model are laid out, the information contained in the resulting SysML model can be exported to the INTO-CPS Application for use in a co-simulation. For each SysML block, Modelio exports the interface definition as an FMI `ModelDescription.xml` file. This information can be used by the other applications as a starting point, such that the models created there comply with the interfaces as defined in the initial SysML multi-model. On the other hand, Modelio can import existing FMUs and extract their interface definitions to create a starting point for multi-model definition. These two workflows are facilitated by interfacing Modelio with the other tools, as illustrated in Figure 10. The

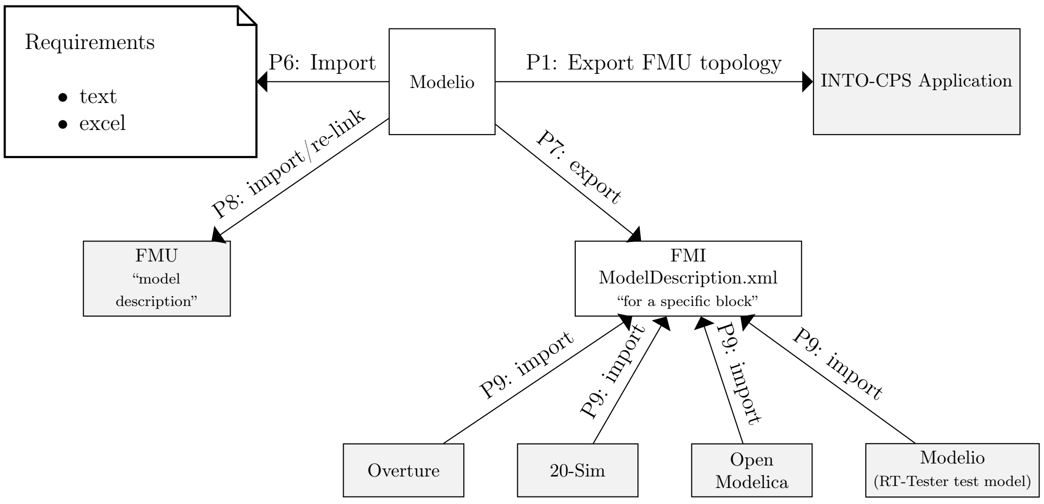


Figure 10: Overview of the Modelio tool

connections in the figure represent, at an abstract level, protocols controlled from the source of the arrows. These are as follows:

P1: A protocol that can be used to open a SysML model and obtain the FMU connection topology.

P6: Import of requirements from Excel, Word or OpenOffice documents.

P7: Export of FMI `ModelDescription.xml` file that contains the elements described in Appendix D.

P8: Import of existing FMU (`ModelDescription.xml`) in order to extract interface information for use inside the SysML multi-model.

P9: Imports the model description and creates an internal tool-specific structure that enables the tool to re-export the same model description, but with the `GUID` updated and internal model structure added. See Section 4.3.

4.3 Model Import / Code Generation

This section describes how the tools Overture, 20-Sim, OpenModelica, and RT-Tester interface with model descriptions exported from Modelio. These tools use these model descriptions as the base interface for the respective models being developed. Once a model is developed, the different kinds of FMUs that can be exported from these tools will respect the model description previously imported. The simulators Overture, 20-Sim and OpenModelica can export two kinds of FMUs: standalone FMUs having no dependencies, or tool wrapper FMUs which require the simulator to be present for simulation. The RT-Tester tool is also capable of exporting FMUs, but will export multiple versions of the same FMU which have different roles in a testing context of the same system. In Figure 11 an overview is given of the interfaces between the tools. The various tools mentioned are represented here as "Generic" and the models developed inside these tools are illustrated as the internal model. When exported, this model will respect the model description imported. The connections in the figure represent, at an abstract level, communications in the direction of the arrow. These are as follows:

P7: Modelio extracts FMU interface information from SysML multi-model and exports it as `ModelDescription.xml`.

P10: Exports a tool specific model as a standalone FMU, *i.e.*, one that is self-contained and which does not require the tool to simulate.

P11: Exports a tool-specific model with a tool wrapper FMU, *i.e.*, one that requires the tool to be present for simulation.

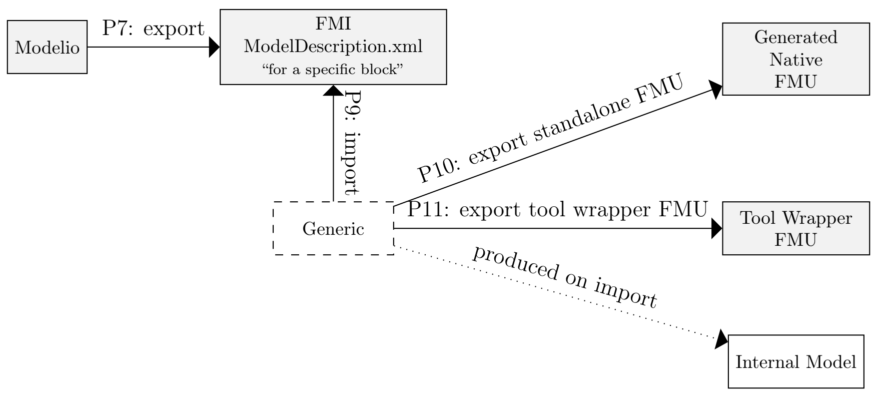


Figure 11: Overview of the import of `ModelDescription.xml` and code generation.

4.4 Simulation

This section describes the interfaces involved in a co-simulation with primary focus on the Common Orchestration Engine (COE). The interactions for the COE is illustrated in Figure 12. It interfaces all FMUs according to the INTO-CPS variant of the FMI standard. It is controlled and configured through the INTO-CPS Application or the design space exploration driver. The connections in the figure represent, at an abstract level, communications in the direction of the arrow. These are as follows:

P12: This is the INTO-CPS variant of the FMI Standard. Further details about its functions usage of the COE can be found in Appendix C.

P13: Live feedback for the INTO-CPS Application to enable monitoring of changes to FMU output variables during simulation. See Appendix A.

The interaction between the COE (described in Section 6.3), the INTO-CPS Application (described in Section 6.1, and the design space exploration driver (described in Section 6.5) illustrated in Figure 12 shows how the COE can simulate a multimodel based on a co-simulation configuration which describes the connection between the models and other relevant configuration parameters. The protocol P13 illustrates live feedback from the COE to the INTO-CPS Application allowing users to observe the progress of a co-simulation. The design space exploration extension is comprised of a set of predefined search algorithm scripts along with user definable evaluation and ranking scripts that use an API to access the COE. During DSE, the driver uses feedback from the user defined evaluation and ranking scripts to direct the exploration until a set of optimal configurations have been obtained. The

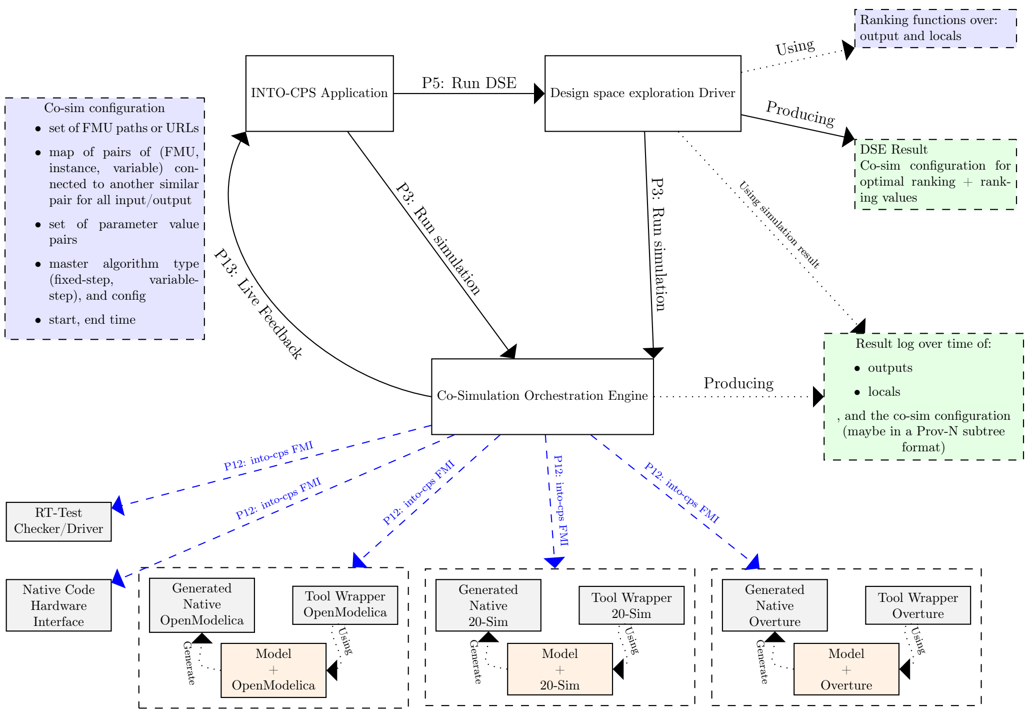

Co-sim configuration
set of FMU paths or URLs
map of pairs of (FMU, instance, variable) connected to another similar pair for all input/output
set of parameter value pairs
master algorithm type (fixed-step, variable-step), and config
start, end time
INTO-CPS Application
P5: Run DSE
Design space exploration Driver
Using
Ranking functions over: output and locals
Producing
DSE Result Co-sim configuration for optimal ranking + ranking values
P13: Live Feedback
P3: Run simulation
P3: Run simulation
Using simulation result
Co-Simulation Orchestration Engine
Producing
Result log over time of: outputs, locals, and the co-sim configuration (maybe in a Prov-N subtree format)
P12: into-cps FMI
RT-Test Checker/Driver
Native Code Hardware Interface
Generated Native OpenModelica
Tool Wrapper OpenModelica
Generate
Using
Model + OpenModelica
Generated Native 20-Sim
Tool Wrapper 20-Sim
Model + 20-Sim
Generated Native Overture
Tool Wrapper Overture
Model + Overture

Figure 12: Overview of the COE

evaluation and ranking scripts obtain the data they need from the COE log for each simulation run.

4.5 Model Checking

Model-checking takes place at two different levels of an INTO-CPS multi-model. First, the RT-Tester Model-Based Test Case Generator (RTT-MBT), which adds model-based testing functionality to the RT-Tester test system, can model-check individual FMUs against desired properties, which are specified in LTL. Second, RTT-MBT will be able to model-check the complete multi-model against desired properties. The model-checking interaction is illustrated in Figure 13. RTT-MBT uses XMI-exported test models to carry out both automated model-based testing, as well as model-checking at the two levels, that is, single components as well as multi-models. Further details are given in Section 6.7.

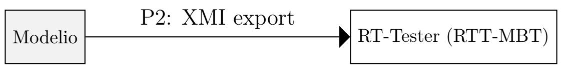


Figure 13: Overview of model-checking and automated model-based testing.

5 Example Use Case for the INTO-CPS Technologies

The INTO-CPS technologies will enable new workflows for model-based design of CPSs from conception to realisation. Seven baseline technologies are combined to form a tool chain that supports this. In order to give an overall picture of how these technologies come together, we give an illustrative example of the technologies. We consider a putative development of a CPS, from scratch, that utilises all features of the INTO-CPS technologies. We simplify the workflow by omitting iteration or feedback between levels.

This is only one such way in which the technologies might be used. For example, not every development will begin with an entirely blank state, and therefore not all steps are necessary. The choice of approach will depend on the experience of the team, their existing practices and the needs of their customers. Deliverable D3.1a [FGPP15a] considers in more detail how the INTO-CPS technologies might be used and what workflows they enable.

In the following description, terms in **bold** are baseline tools or INTO-CPS tools. Terms in *italics* correspond to activities in the ontology that produce artifacts of traceability and provenance (Deliverable D3.1b [FGPP15b]).

- At each step in the development, engineers can store and retrieve artefacts using the **INTO-CPS Application**. Design rational and *Design Notes* are attached to artefacts, as well as information about which engineer created or modified them. Data can be retrieved to reconstruct the design rationale at any time, enable traceability throughout the entire design process.
- Using **Modelio**, engineers can construct an architectural model of a system expressed in the SysML/INTO-CPS profile [APCB15]. Some blocks (components or subsystems) have an associated model (either discrete or continuous), to be described using some modelling language (e.g., VDM-RT, OpenModelica, 20sim); these block with separate models result in FMUs. A SysML/INTO-CPS connections diagram provides the topology of the FMUs, and describes flow of information

between the different system components.

- From the architectural model, **RT-Tester** can build an internal representation of the system that is used to *Create Tests* and can also be used for *Model Checking*.
- **Modelio** can *Export FMI version 2.0 Model Descriptions* for modelling tools: **Overture**, **OpenModelica**, and **20-sim**. Each modelling tool can *Import a Model Description* to be used as a basis for *Simulation Modelling*. Once models have been completed, each tool can *Export an FMU* for co-simulation.
- Using the architectural model, the **INTO-CPS Application** can *Create a co-simulation configuration* that forms the basis for a co-simulation which can then be co-simulated by the **Co-simulation Orchestration Engine (COE)**.
- The **Design Space Exploration (DSE) tool** can be used to create multiple configurations to co-simulate multiple variations of a design. The **DSE tool** can be used to rank designs using objective functions and feedback can cull unnecessary configurations and optimise designs.
- The **COE** can run co-simulations with one or more FMUs being replaced by signals from real hardware or from *Code Generated* by the modelling tools. **RT-Tester** can create a *Test Oracle FMU* to run tests against these elements of the prototype.

6 The INTO-CPS Tool chain

This section gives an introduction to the tools that are part of the INTO-CPS tool chain, and provides information about where more information can be found about the individual tools that are not described in detail in this document.

6.1 The INTO-CPS Application

In the Spring semester of 2015 two MSc thesis students elicites requirements of the user interface of the INTO-CPS tool [TC15]. This effort is taken forward in the development of an INTO-CPS Application connected to the different baseline technologies and the new INTO-CPS features from a common platform. This INTO-CPS Application is the main graphical interface of INTO-CPS tool chain. Its goal is to allows user to manage Co-Simulation configuration, DSE runs, and FMU checker generation which will respectively utilized by COE, DSE, and RTTester *cf.* Figure 9.

Developed as a collection of Eclipse plug-ins `https://eclipse.org`, it allows the INTO-CPS Application to be independent of all INTO-CPS baseline tool but also compatible with Modelio as so to have one unique interface for INTO-CPS SysML Modelling Section 6.2 and Co-Simulation configuration. The INTO-CPS Application is, for now, composed of two components:

File Browser: As shows in Figure 14 this plug-in provides an overview of the INTO-CPS files - i.e. FMUs, Co-Simulation configurations, Run Results.

Co-Simulation configuration editor: This plug-in provides an editor for Co-Simulation configuration files. It allows the user to configure all parameter of a Co-Simulation e.g. which FMUs is used, the connections between them, initialize the inputs, and configure the algorithm and the timing used for a given simulation as shown in Figure 15. The configuration data follow the XSD Schema provided in Appendix E.

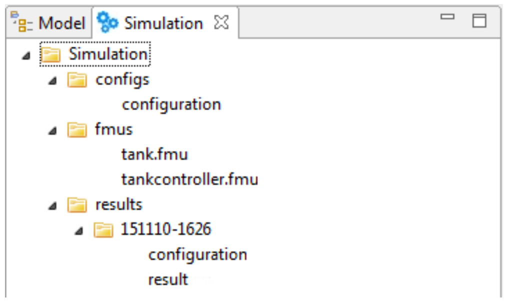
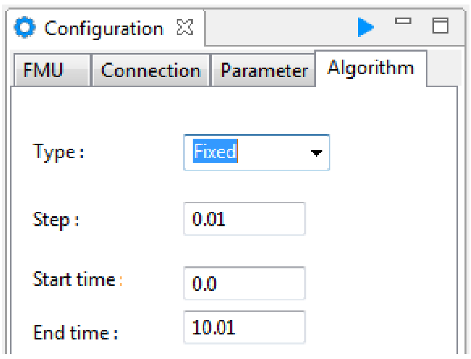


Figure 14: INTO-CPS Application browser



Type : Fixed
Step : 0.01
Start time : 0.0
End time : 10.01

Figure 15: INTO-CPS Application co-simulation configuration editor

Detailed information about how to use the INTO-CPS Application are available in [BLL+15]

6.2 The INTO-CPS SysML Modelling

To facilitate co-simulation the INTO-CPS project has extended SysML with an INTO-CPS SysML profile [APCB15] that contains several extensions and a formal semantics of the INTO-CPS SysML. The SysML modelling in INTO-CPS is implemented as a Modelio module, and is able to interface with the INTO-CPS Application. The SysML profile allows the user to define the FMU topology[BQS15] that can be exported to the INTO-CPS Application, which then allows the user to specify additional information required for a co-simulation such as: start- and end-time.

6.3 Co-Simulation Orchestration Engine

The Co-Simulation Orchestration Engine (COE) is the core engine that is used for connecting different FMUs together in co-simulations in the INTO-CPS project. The sources for this development are stored in a private Git repository at `https://github.com/twt-gmbh`. The executable derived from these sources is made freely available to all partners of the consortium (as well as external parties) and released at regular intervals.

The COE is built as a service and serves its clients using JSON over the HTTP protocol (see Appendix A), allowing clients to perform co-simulations based on a co-simulation configuration that is composed of a number of FMUs, their connections, parameters and the desired step-size algorithm. The COE supports two types of step-size algorithms:

Fixed step size algorithm: This algorithm performs fixed size steps with a predefined step size. However, the COE uses this algorithm in combination with an error recovery feature that, under some constraints, can progress a discarded fixed step by a smaller step. This enables an otherwise fatal error to be recovered so the fixed step size algorithm can continue to drive the simulation after the discarded step is recovered. The recovery depends on the ability of all FMUs to roll-back and provide status information about the last successfully simulation time. If this is provided then COE will roll-back all FMUs and calculate a smaller step size based on the current time and the minimum of the last successful time the FMUs reported and do a step with that size before resuming the use of the fixed step size algorithm.

Variable step size algorithm: This algorithm performs variable steps. The algorithm is similar to *Algorithm 2*, and *Algorithm 3* from [BBG$^+$13a]. However our implementation of *Algorithm 3* does not yet support stepping the FMUs that supports roll-back but not `getMaxStepsize` first. The COE implementation automatically detects if the INTO-CPS specific FMI extension to obtain the maximum step size for each

FMU is available, and based on this selects the appropriate algorithm. When using the variable variable step size algorithm then COE uses constraints on the connections to make sure that a given step does not result in instability [?] which is detected by a violation of one of the constraints. See Appendix B for in-depth details on how the variable step size calculation is performed.

6.3.1 FMI Integration

The COE supports version 2 of the FMI Standard, using dynamic loading of FMUs. It utilizes the functions listed in Appendix C to orchestrate co-simulations. The COE also supports an INTO-CPS specific function to obtain the maximum step size that a given FMU can accept at a given synchronization point during a co-simulation. The need for this function arises from the assumption that a co-simulation should complete as fast as possible, in combination with some looseness in the standard, whereby the roll-back capability has to be provided by an FMU. The standard provides a way to obtain the last successful simulation time once a simulation has failed, but if this is used with FMUs which cannot roll-back, then each discarded `doStep` will require a complete restart with a new step size which is the last smallest step size known to have executed successfully. While it is possible to perform a simulation where all FMUs support roll-back it is not be possible to incorporate an FMU with variable step size that cannot roll-back. However, using the custom `getMaxStepsize` function presented here, together with the variable step size algorithm [BBG+13a], avoids this kind of restart, the need for roll-back, and thus enables faster co-simulations.

6.3.2 Co-Simulation Execution

A co-simulation can be performed by the COE as shown in Figure 16, where transitions to the two states *User configuration configured* and *User simulation start-end time configured* are triggered by the user through the INTO-CPS Application by calling one of the `initialize` or `simulate` commands from the protocol described in Appendix A.

The state machine *Initialize* is shown in detail in Figure 17. It shows the phases through which the COE goes as it initializes for a given simulation. The *Simulate* state represents the simulation proper, and is shown in detail in Figure 18.

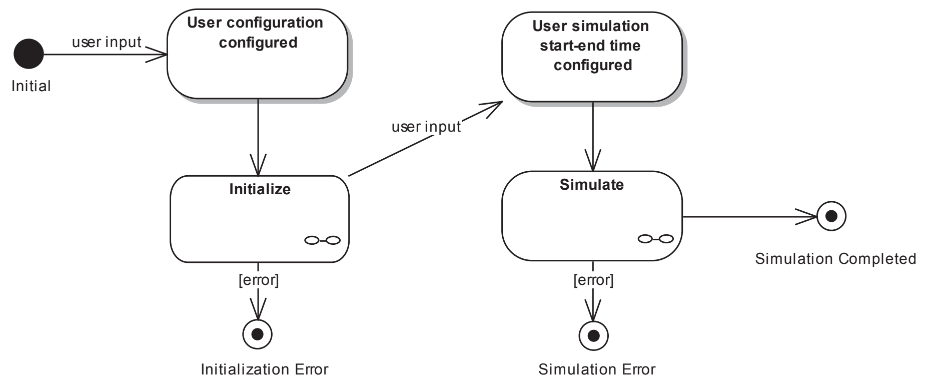


Figure 16: Top level state machine of the COE.

The Initialization State The state machine starts by reading all model description files embedded in the FMUs and validating their structure using the schema from the FMI standard. In addition to this, it also validates them using the semantic rules obtained from the textual description of the FMI standard (*i.e.* rules which cannot be enforced by schema-based validation). These include how *scalar variables* may be configured with their attributes, and how the *model structure* must be laid out.

After the model descriptions are validated, the COE uses the co-simulation configuration supplied by the user, through the INTO-CPS Application, to configure inputs and outputs of all FMUs, checking that all connections are valid, and caches these in lookup tables for execution. Then the libraries implementing the FMU behaviour are extracted from the FMU zip archives and dynamically loaded, followed by creation of all required FMU instances. The number and name of the instances are obtained from the cached connection tables. Finally, the COE extracts the log categories from the model description of each FMU and returns these, together with a session ID, to the INTO-CPS Application.

The Simulation State The simulation state machine shown in Figure 18 can only be entered if the COE has initialized the current session and if the user has supplied the required start and the optional end time, and the optional set of activated logging categories. It starts by setting the logging categories, if supplied, followed by experiment setup, setting start and optionally end times, allowing the FMUs to allocate internal memory for the simulation. Initialization in this context refers to the FMI `setX` functions,

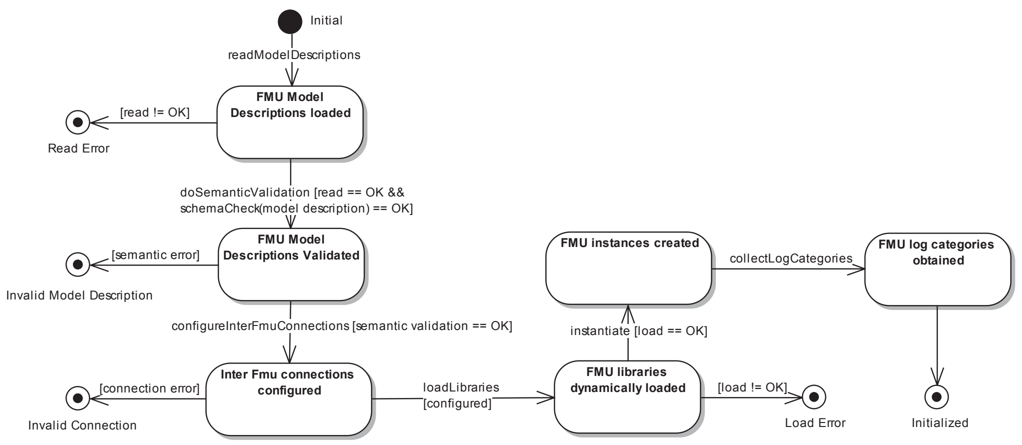

Initial
readModelDescriptions
FMU Model Descriptions loaded
[read != OK]
Read Error
doSemanticValidation [read == OK && schemaCheck(model description) == OK]
FMU Model Descriptions Validated
[semantic error]
Invalid Model Description
configureInterFmuConnections [semantic validation == OK]
Inter Fmu connections configured
[connection error]
Invalid Connection
loadLibraries [configured]
FMU libraries dynamically loaded
[load != OK]
Load Error
instantiate [load == OK]
FMU instances created
collectLogCategories
FMU log categories obtained
Initialized

Figure 17: View of the *Initialize* state machine from Figure 16.

where parameters and initial values are provided, and obtained either from the model description or from the co-simulation configuration. After the FMUs are initialized, the global state is obtained across all FMUs and co-simulation stepping can begin starting by calculating the step size. This is done either by using a fixed step size given be the user of the COE, or by using a variable step size algorithm that uses a INTO-CPS specific extension `getMaxStepsize` on each FMU and taking the minimum of the obtained values. While the current time is less than the end time, stepping progresses the FMUs by a single time step, as shown in Figure 19. When the end time is reached, the COE terminates all FMUs and frees the allocated FMU instances. If an error occurs during stepping, the simulation is aborted.

Stepping The Stepping state machine shown in Figure 19 either performs the requested step in all FMUs and obtains the new global state, or if any FMU discards the step size, and they all are able to roll-back and provide information about the step size they can accept, then the COE will retry with this new step size. Rolling back involves resetting to a previous state in all FMUs and discarding the estimated derivatives. In case any of the FMUs reports an error, or if they cannot all roll-back, then the COE exits with an error.

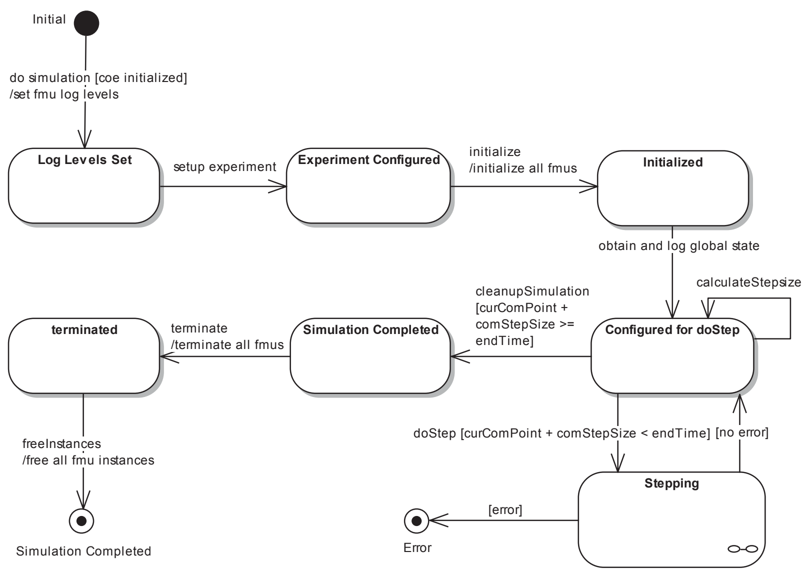


Figure 18: View of the *Simulate* state machine from Figure 16.

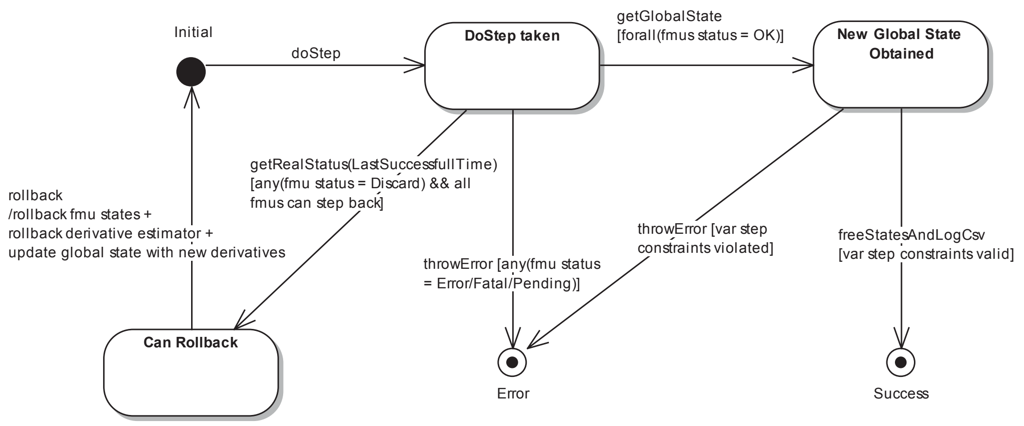

Initial
doStep
DoStep taken
getGlobalState
[forall(fmus status = OK)]
New Global State Obtained
getRealStatus(LastSuccessfullTime)
[any(fmu status = Discard) && all
fmus can step back]
rollback
/rollback fmu states +
rollback derivative estimator +
update global state with new derivatives
throwError [any(fmu status
= Error/Fatal/Pending)]
throwError [var step
constraints violated]
freeStatesAndLogCsv
[var step constraints valid]
Can Rollback
Error
Success

Figure 19: View of the *Stepping* state machine from Figure 18.

6.3.3 Performance Optimization

The execution time of a simulation is a crucial factor in increasing the usefulness to the user, and the uptake potential of the COE. As some FMUs perform complex work in the stepping process, the time taken for different FMUs to perform a step can vary greatly. Implementing concurrency in the COE can decrease the execution time of a simulation and take full advantage of available hardware [Han16]. To optimize the performance of different simulations, it is necessary to implement different execution strategies, including one where concurrency is not used at all. The different options to optimize performance include, but are not limited to, letting the COE change strategy between steps, providing feedback to be used in future simulations and providing additional configuration options. Currently, different concurrency strategies are implemented, along with a test framework to measure the performance of these and future strategies.

6.4 Integration of Simulation Tools

Fundamentally, integration of the individual tools which make up the INTO-CPS tool chain is facilitated by compliance with the FMI standard version 2.0 for co-simulation. Deliverable D4.1b [PBLG15] describes how the tools Overture [LBF$^+$10], 20-sim [Bro97], and OpenModelica [Fri04] achieve compliance with the FMI standard.

6.5 Design Space Exploration

The Design Space Exploration (DSE) extension of the INTO-CPS tool chain allows a user to automatically explore a range of parameters and candidate implementation options automatically while avoiding the problem of state space explosion. The extension is launched from the main INTO-CPS application but is actually a separate entity. The interested reader can find more details about the DSE extension in [GHJ$^+$15]. There is three main parts to the extension:

DSE Driver This is the main component of the DSE extension, it has the role of gathering the user's desired parameter ranges and model descriptions and to determine exactly which combinations of parameters should be simulated next. The choice to parameters will be determined by which of the DSE search algorithms the user has elected to use given the nature of the model and problem domain;

DSE Analysis has the role of processing the results of simulation to obtain the key objective values by which it should be evaluated. Such analysis could be as simple as finding the maximum power consumed by a device during the simulation, or it could be more complex such as computing the deviation from a path or some temporal constraint over the states visited during the simulation (the latter functionality will be implemented in part using the RT-Tester tool, see Section 6.7); and

DSE Ranking Once the key objective values have been determined then this data is processed to determine which parameters produced the best results. The ranking can be via a ranking function, if the user understands specifically what they are looking for; or by the pareto optimal method, if the user wishes to find a range of best tradeoffs between two or more of the objectives. If a closed loop DSE search algorithm, such as a genetic method, has been chosen in the driver then this ranking information is fed back to the driver for it to use when selecting the next simulations to perform.

6.6 Test Automation

The test automation extension of the INTO-CPS tool chain allows users to automatically generate test stimulations and test oracles from test models. In principle, test models are collections of state charts which specify the desired behaviour of the system-under test. The test case generate evaluates these models, extracts a sequence of suitable stimulations for covering the test model, and generates checks to detect deviations between the behaviour of the System-Under-Test (SUT) and the test model. This functionality will be implemented within the RT-Tester Model-Based Test Case Generator (RTT-MBT) as a tool that is entirely independent of the INTO-CPS Application. However, the configuration of the co-simulation environment itself and the test automation functionality provided by RTT-MBT takes place entirely in the main INTO-CPS Application. Further details are given in Deliverable D5.1b [MPB15].

6.7 Model Checking

Model checking in INTO-CPS shall be applied to discrete event (DE) test models, but also support continuous time (CT) models via suitable abstraction mechanisms, see Deliverable D5.1c [BF15]. The RT-Tester Model-Based Test Case Generator (RTT-MBT) is an upgrade for RT-Tester that adds model-based testing to the RT-Tester test system. One particular feature of RTT-MBT is bounded model checking (BMC) of LTL specifications for DE systems, where the entire model to be checked consists of a SUT on the one hand, and an environment model on the other. Arbitrary LTL specifications can be verified, where the atomic propositions typically range over outputs of the SUT, model variables that are internal to the SUT, and timers. For example, let us assume that a test model shall express the following behaviour: If some input voltage is below 10 units, the SUT shall set an output error flag within 10 time units. This is a typical property to be checked using model checking techniques in RTT-MBT.

The core feature of model checking in INTO-CPS is the integration and configuration of different test models into a single SUT configuration, to which established model checking techniques can then be applied. This approach allows the combined behaviour of several components to be checked, and takes into account the interaction between these components. The configuration has to be performed via the main INTO-CPS Application, which then invokes RTT-MBT so as to perform the actual model checking. Details about modelling the interfaces and connections between the different system components are given in [BLL+15].

6.8 Code Generation

The INTO-CPS platform needs to be able to support the ability of code generating sub models that are included in a multi-model. This will take place using the individual baseline tools augmented with new code generation features as needed by the case study owners from WP1. The principles for the code generators is further described in Deliverable D5.1d [HLG+15].

6.9 Provenance and Traceability

The ability to relate the many artefacts stored within the different tools is an essential feature of the INTO-CPS tool chain. This feature has three different abilities:

- the ability to logically relate artefacts contained in the many tools, such as relating a requirement to a model that implements it, this we refer to as traceability;
- the ability to record the temporal links between entities produced during the development, such as connecting simulation results and the tools and models that created them, this we term the provenance; and
- finally the ability to query this stored data to produce useful information for the various stakeholders of the project.

The success of the provenance and traceability activities will depend on the tool support, which begins in year 2 of the project, but more so on a solid ontology of the INTO-CPS artefacts and relations that will be tracked. The initial work describing these elements is included in Appendix A of [FGPP15a] and the interested reader is directed there to find more details. Initial exploration of how this can be incorporated in the tool suite has been carried out [Han16].

7 Release Management

The goal for release management in INTO-CPS is to establish a process for officially and hopefully quality assured releases of the entire tool chain. INTO-CPS involves a lot of tools, with completely independent development and release schedules, and therefore a major INTO-CPS release consists of a release bundle containing the released tools which includes a listing of these tools with their respective versions that have been tested together and thus guaranteed to work.

7.1 In practice

The release management team will make a online repository for partners to deliver their build releases in binary form. These can be black box tested to some extent using the FMU export support or other INTO-CPS specific tool protocol. Uploading a new release will trigger a set of automated test suites to be run, and if errors are found, then an error report will be sent to developers responsible for that tool.

If the initial automated tests pass then further tests may be made by the release management team and once the quality has been assured the specific release candidate of the newly uploaded INTO-CPS tool will be marked as approved. Then the release manager can assemble a new INTO-CPS release updating the previous release bundle with the new INTO-CPS tools that has been approved since last release.

7.2 Releases

At certain scheduled dates (set by the project coordinator) stable releases will be released to the INTO-CPS project, covering all the tools.

However, beta releases should be made available as soon as possible. Meaning that, as soon as a new tool release has been marked approved by the release management team it must become available online for the the INTO-CPS project.

7.3 Dependencies

The INTO-CPS tools have dependencies to either the FMI standard version or other INTO-CPS tools. Therefore we except to create a simple way for specifying tool dependencies. The initial idea is as follows: Each release is uploaded as a single zip-file. Inside this file there must be a `into-cps-dependencies.txt` specifying dependencies to other INTO-CPS tools. The format of the file contents will be similar to that used in Pip packages, or Bower packages (Python, and JavaScript package management system, respectively[pip, npm]). For example, if a tool depends on version 2.0.4 of Overture or newer than the file would have a line like so: `overture>=2.0.4`.

8 Conclusion

During the first year of the INTO-CPS project a lot of effort has been devoted to establishing the best approaches for connecting the different baseline tools and their extensions to from an INTO-CPS tool chain. This includes the different XML formats as well as the protocols necessary for connecting the different features in a seamless fashion that will become natural for users of the tool chain. There is not yet full connectivity but the design for the different ways of interacting has been completed in the first year. At the core of the tool chain a first stable FMI-based COE able to act both with fixed as well as variable time steps has been produced and the different baseline modelling and simulation tools have been extended to export FMUs. In addition to this center of the INTO-CPS tool chain coverage along the development process has been started up with extensions of Modelio with export of model descriptions of the interfaces for the different FMUs to be included in a CPS as well as export of the overall composition of the constituent models included and combined in the model of the CPS. This means that connections between many individual tools has been established and in addition plans for the connections that will be established in the second year of the project has been established as well. All in all we feel that good progress on establishing the overall platform for the INTO-CPS tool chain has been made during the first year of the project.

References

[ACG$^+$15] Nuno Amalio, Ana Cavalcanti, Stefan Gulan, Christian König, and Jim Woodcock. Foundations for FMI Co-Modelling. Technical report, INTO-CPS Deliverable, D2.1d, December 2015.

[APCB15] Nuno Amalio, Richard Payne, Ana Cavalcanti, and Etienne Brosse. Foundations of the SysML profile for CPS modelling. Technical report, INTO-CPS Deliverable, D2.1a, December 2015.

[BBG$^+$13a] 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, Sept 2013.

[BBG$^+$13b] David Broman, Christopher Brooks, Lev Greenberg, Edward A. Lee, Michael Masin, Stavros Tripakis, and Michael Wetter. Determinate composition of fmus for co-simulation. Technical Report UCB/EECS-2013-153, EECS Department, University of California, Berkeley, Aug 2013.

[BCWS11] Jens Bastian, Christoph Clauss, Susann Wolf, and Peter Schneider. P.: Master for co-simulation using fmi. In *8th International Modelica Conference*, 2011.

[BF15] Jörg Brauer and Simon Foster. Abstraction Techniques from CT to DE. Technical report, INTO-CPS Deliverable, D5.1c, December 2015.

[BLL$^+$15] Victor Bandur, Peter Gorm Larsen, Kenneth Lausdahl, Sune Wolff, Carl Gamble, Adrian Pop, Etienne Brosse, Jörg Brauer, Marcel Groothuis, and Christian Kleijn. User Manual for the INTO-CPS Tool Chain. Technical report, INTO-CPS Deliverable, D4.1a, December 2015.

[Blo14] Torsten Blochwitz. Functional mock-up interface for model exchange and co-simulation. https://www.fmi-standard.org/downloads, July 2014. Torsten Blochwitz Editor.

[BQS15] Etienne Brosse, Imran Quadri, and Andrey Sadovykh. COE Contracts from SysML. Technical report, INTO-CPS Deliverable, D4.1c, December 2015.

[Bro97] Jan F. Broenink. Modelling, Simulation and Analysis with 20-Sim. *Journal A Special Issue CACSD*, 38(3):22–25, 1997.

[CLT$^+$15] Fabio Cremona, Marten Lohstroh, Stavros Tripakis, Christopher Brooks, and Edward A. Lee. Fide – an fmi integrated development environment. In *Symposium on Applied Computing 2016*, 2015.

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

[FGPP15a] John Fitzgerald, Carl Gamble, Richard Payne, and Ken Pierce. Method Guidelines 1. Technical report, INTO-CPS Deliverable, D3.1a, December 2015.

[FGPP15b] John Fitzgerald, Carl Gamble, Richard Payne, and Ken Pierce. Methods Progress Report 1. Technical report, INTO-CPS Deliverable, D3.1b, December 2015.

[FJLM14] Amit Fizher, Clas A. Jacobson, Edward A. Lee, and Richard M. Murray. Industrial Cyber-Physical Systems – iCyPhy. In M. Aiguier et al., editor, *Complex Systems Design and Management*, pages 21–37. Springer, January 2014.

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

[GHJ$^+$15] Carl Gamble, Francois Hantry, Claes Dühring Jæger, Christian König, Alie El din Madie, and Richard Payne. Design Space Exploration in the INTO-CPS Platform. Technical report, INTO-CPS Deliverable, D5.1a, December 2015.

[Han16] Casper Thule Hansen. Investigating Concurrency in the Co-Simulation Orchestration Engine for INTO-CPS. Master's thesis, Department of Engineering, Aarhus University, Denmark, January 2016.

[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.

[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.

[MPB15] Oliver Möller, Adrian Pop, and Jörg Brauer. Distributed Testing and Simulation Network. Technical report, INTO-CPS Deliverable, D5.1b, December 2015.

[NGL$^+$14] Himanshu Neema, Jesse Gohl, Zsolt Lattmann, Janos Sztipanovits, Gabor Karsai, Sandeep Neema, Ted Bapty, John Batteh, Hubertus Tummescheit, and Chandrasekar Sureshkumar. Model-based integration platform for fmi co-simulation and heterogeneous simulations of cyber-physical systems. In *The 10th International Modelica Conference 2014*, Lund, Sweden, March 2014. Modelica Association.

[npm] npm is the package manager for javascript. https://www.npmjs.com/. Accessed: 2015-1-25.

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

[pip] pip 7.1.2. the pypa recommended tool for installing python packages. https://pypi.python.org/pypi/pip. Accessed: 2015-1-25.

[Pto14] Claudius Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014.

[SBC14] Vitaly Savicks, Michael Butler, and John Colley. Co-simulating event-B and Continuous Models via FMI. In *Proceedings of the 2014 Summer Simulation Multiconference*, SummerSim '14, pages 37:1–37:8, San Diego, CA, USA, 2014. Society for Computer Simulation International.

[TC15] Rasmus Thimsen and Sam Chieng. Eliciting User Interface Requirements. Master's thesis, Department of Engineering, Aarhus University, Denmark, June 2015.

[vADVM15] Bert van Acker, Joachim Denil, Hans Vangheluwe, and Paul De Meulenaere. Generation of an Optimised Master Algorithm for FMI Co-simulation. In *DEVS '15 Proceedings of the Symposium on Theory of Modeling & Simulation*, January 2015.

A The COE Protocol

The COE protocol is based the HTTP protocol where payloads are send as JSON data, using a similar approach as Representational State Transfer (REST). However, the COE keeps state opposed to RESTful. The format used to describe each command is that its URL is listed and any arguments are prefixed with a colon, *e.g.* "*:section*".

The COE operates as a server where co-simulations can be carried out. Once a co-simulation is requested a session is created and the `sessionId` is returned and must be used in any further communication with the COE for the particular co-simulation. The following sections will list the API for the COE where each command has it own section. The minimum required command sequence for a co-simulation is: *initialize*, *simulate*, *result*, and the *destroy*.

A.1 COE Information

Information about the COE is available at:

`http://localhost:8082/`

This returns an html page viable in a standard browser listing the COE version and with links to the API command.

A.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 the COE protocol part of this document.

Content-Type: text/plain The LaTeX source file for COE protocol part of this document.

The returned content contains the COE protocol part of this document in the format specified.

A.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

The command returns the following response on success:

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

Listing 1: JSON status payload

If no session ID is given, an array of all session statuses are returned.

A.4 The Initialize Command

The command is available at:

`http://localhost:8082/initialize`

The command takes no arguments but requires a JSON payload, Content-Type: application/json containing the following entries:

fmus A list of the location of the FMUs.

connections A map of connections, output to inputs. The format is `<GUID>.<instance name>.<scalar variable name>`.

`<GUID>` the guid from the model description in the FMU.

`<instance name>` any name not containing dot. Each name is used to identify a unique instance of the FMU.

`<scalar variable name>` may contain any FMI version 2.0 allowed characters including dot.

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.

```
{
    "fmus":[
        "file://controller.fmu",
        "file://tank.fmu"
    ],
    "connections":{
        "{guid-controller}.instance1.valve":"[{guid-tank}.
           instance1.valve"],
        "{guid-tank}.instance1.level":["{guid-controller}.
           instance1.level"]
    },
    "parameters":{
        "maxLevel":8,
        "minLevel":2
    },
    "algorithm":{
        FIXED-STEP-SIZE-CONFIG or VARIABLE-STEP-SIZE-CONFIG
    }
}
```

Listing 2: JSON co-simulation payload

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

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

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

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

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:

```
"id":{
        type:"[zerocrossing|boundeddifference|
            samplingrate]",
        ...
}
```

The `id` is a string that is used to identify a 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 A.4.1, `boundeddifference` in section A.4.2, and `samplingrate` in section A.4.3.

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",
            }]
        }
}
```

Listing 3: JSON initialization payload

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

A.4.1 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>
}
```

Listing 4: JSON zerocrossing constraint payload definition

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 B.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.

A.4.2 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>
}
```

Listing 5: JSON boundeddifference constraint payload definition

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 3), 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 B.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 B.7.3) is by default enabled. It may be disabled by setting this value to `false`.

A.4.3 Definition of a Sampling Rate Constraint

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

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

Listing 6: JSON samplingrate constraint payload definition

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}.

A.5 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, and executes a co-simulation that is already initialized with the supplied session id:

: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"]
    }
}
```

Listing 7: JSON simulate payload

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

The command returns the following response on success:

```
{
    "status":"simulating",
    "sessionid": 1234
}
```

Listing 8: JSON status payload

A.6 The Result Command

The command is available at:

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

The command takes the following arguments, and based on these returns the result:

: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

A.7 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.

A.8 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.

B 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 A.4), 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.

B.1 Interface with the Master Algorithm

The variable stepsize calculator is called by the master algorithm (COE) before each doStep. The step size calculator is provided with 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 B.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.

B.2 Constraint Types

There are four constraint types,

- Zero Crossing
- Bounded Difference

- Sampling Rate
- FMU-requested,

of which the first three are defined in the JSON file (see Section A.4). The fourth constraint, FMU-requested, is not defined in the JSON file but is coded in an FMU and reported by the FMU when the COE queries for it (see Section B.6).

B.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 20-sim, Open Modelica, or 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.

B.3.1 Extrapolation

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

For first order extrapolation,

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

is used.

For second order extrapolation,

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

is used.

B.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.

B.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

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

i.e. 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.

B.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

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

with $\hat{\varepsilon}$ 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.

B.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.

B.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 1.

Table 1: 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*,

- *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

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

with $TOLERANCE\_SAFETY\_FACTOR \leq 1.0$ 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 2.

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.

Table 2: 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 $(= 10.0 \cdot \delta_{relax})$

- 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.

B.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 A.4.2.

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 zerocrossing 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. With the safety factor

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

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

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

listed in Table 3,

Table 3: 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

the bins are determined. 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 B.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 B.7.1).

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

B.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 A.4.3.

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.

B.6 FMU-requested Constraints

A constraint requested by an FMU is a discrete constraint. A proposal is underway to extend the FMI standard with the procedure

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

so 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.

B.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.

B.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 .

(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.)

B.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.

B.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 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 JSON configuration file (see Section A.4.2).

to the definition of the Bounded Difference constraint in the JSON config file. When the second stra-tegy is disabled, an active discrete constraint will likely reduce the next stepsize(s) proposed by the Bounded Difference constraint handler, potentially reducing efficiency.

B.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, for instance,

```
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
```

C FMI integration of the COE

The table presented in Table 4 lists all FMI version 2.0 functions plus the custom INTO-CPS functions. It also indicates where the COE uses the functions.

Common Functions	Usage
getTypesPlatform	Not used
getVersion	Main algorithm
setDebugLogging	Main algorithm
instantiate	Main algorithm
freeInstance	Main algorithm
setupExperiment	Main algorithm
enterInitializationMode	Main algorithm
exitInitializationMode	Main algorithm
terminate	Main algorithm
reset	Not used
getReal	Main algorithm
getInteger	Main algorithm
getBoolean	Main algorithm
getString	Main algorithm
setReal	Main algorithm
setInteger	Main algorithm
setBoolean	Main algorithm
setString	Main algorithm
getFMUstate	Main algorithm
setFMUstate	Main algorithm
freeFMUstat	Main algorithm
serializedFMUstateSize	Ignored
serializeFMUstate	Ignored
deSerializeFMUstate	Ignored
getDirectionalDerivative	not used
Functions for FMI2 for Co-Simulation	
setRealInputDerivatives	Not used
getRealOutputDerivatives	Main algorithm
doStep	Main algorithm
cancelStep	Not used
getStatus	Not used
getRealStatus	Main algorithm
getIntegerStatus	Not used
getBooleanStatus	Not used
getStringStatus	Not used
INTO CPS specific	
getMaxStepsize	Main algorithm (var step)

Table 4: Table describing the COE integration with the FMI standard functions.

D The Intermediate Model Description Format

The intermediate model description follows the FMI version 2 model description format but only contains the following information:

`fmiModelDescription/guid` On export a new unique GUID will be generated

`ModelVariables/ScalarVariable*` Scalar variables are both handled on import and export. The following attributes are supported:

`name` The name of the scalar variable

`description` An optional description of the variable

`causality` With the following values:

- `input`
- `output`
- `parameter`

The scalar variable may only have on of the following types:

- `real`
- `integer`
- `boolean`

E Co-simulation Configuration File Format

The co-simulation configuration file format is defined as follows:

- node Configuration [1]
  - node Fmus [1]
    - node Fmu [0..*]
      - attribut path:String (mandatory)
  - node Connections [1]
    - node Connection [0..*]
      - attribut sourceFmu:String (mandatory)

        · attribut sourceVariable:String (mandatory)
        · attribut targetFmu:String (mandatory)
        · attribut targetVariable:String (mandatory)
  - node Parameters [1]
    * node Parameter[0..*]
        · attribut fmu:String (mandatory)
        · attribut name:String (mandatory)
        · attribut init:String (mandatory)
  - node Algorithm [1]
    * attribut type:String "fixed" or "variable" (mandatory)
    * attribut stepSize:Float (optional)
  - node Time
    * attribut startTime:Float (mandatory)
    * attribut endTime:Float (mandatory)

In Listing 9 an XSD Schema is also given for the co-simulation configuration file format that enabled automated validation.

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault
   ="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="Configuration">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Fmus">
          <xs:complexType>
            <xs:choice maxOccurs="unbounded" minOccurs="0">
              <xs:element name="Fmu">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute type="xs:string" name="path"
                          use="mandatory"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:choice>
          </xs:complexType>
        </xs:element>
        <xs:element name="Connections">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Connection" maxOccurs="unbounded
" minOccurs="0">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute type="xs:string" name="
                          sourceFmu" use="mandatory"/>
                      <xs:attribute type="xs:string" name="
                          sourceVariable" use="mandatory"/>
                      <xs:attribute type="xs:string" name="
                          targetFmu" use="mandatory"/>
                      <xs:attribute type="xs:string" name="
                          targetVariable" use="mandatory"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Parameters">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Parameter" maxOccurs="unbounded"
 minOccurs="0">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute type="xs:string" name="fmu"
                          use="mandatory"/>
                      <xs:attribute type="xs:string" name="name"
                           use="mandatory"/>
                      <xs:attribute type="xs:float" name="init"
                          use="mandatory"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Algorithm">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute type="xs:string" name="type">
```

```
                  <xs:simpleType>
                    <xs:restriction base="xs:string">
                      <xs:enumeration value="fixed"/>
                      <xs:enumeration value="variable"/>
                    </xs:restriction>
                  </xs:simpleType>
                </xs:attribute>
                <xs:attribute type="xs:string" name="stepSize"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="Time">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute type="xs:float" name="startTime"/>
                <xs:attribute type="xs:float" name="endTime"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Listing 9: Co-simulation configuration file schema Model Description XSD Schema

F List of Acronyms

20-sim	Software package for modelling and simulation of dynamic systems
API	Application Programmers Interface
AST	Abstract Syntax Tree
AU	Aarhus University
CIF	Compositional Interchange Format
CLE	ClearSy
CLP	Controllab Products B.V.
COE	Co-simulation Orchestration Engine
COTS	Commercial Off-The-Shelf
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
FMU	Functional Mockup Unit
HiL	Hardware-in-the-Loop
HLA	High Level Architecture
HMI	Human Machine Interface
HRC	Heterogeneous Rich Components
HW	Hardware
ICT	Information Communication Technology
IDE	Integrated Design Environment
JSON	JavaScript Object Notation
M&S	Modelling and Simulation
MBD	Model Based Design
MC	Model Checking
MiL	Model-in-the-Loop
OMG	Object Management Group
OS	Operating System
PROV-N	The Provenance Notation
REST	Representational State Transfer
RPC	Remote Procedure Call
SiL	Software-in-the Loop
SoS	System of Systems
ST	Softeam
SVN	Subversion
SysML	Systems Modelling Language

TA	Test Automation
TRL	Technology Readiness Level
TWT	TWT GmbH Science & Innovation
UML	Unified Modelling Language
UNEW	University of Newcastle upon Tyne
UTP	Unifying Theories of Programming
UTRC	United Technology Research Center
UY	University of York
VDM	Vienna Development Method
VSI	Verified Systems International
WP	Work Package
XML	Extensible Markup Language