



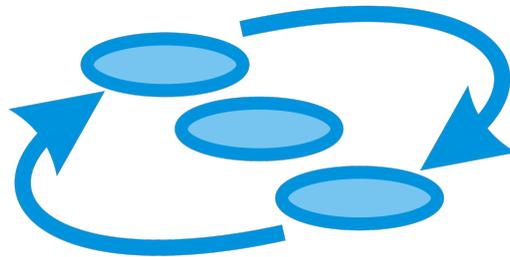
1

Grant Agreement: 644047

2

3

INtegrated TOol chain for model-based design of CPSs



**INTO-CPS**

4

## **INTO-CPS Tool Chain User Manual**

5

Deliverable Number: D4.3a

6

Version: 0.01

7

Date: December, 2017

8

Public Document

9

<http://into-cps.au.dk>

10

11 **Contributors:**

- 12 Victor Bandur, AU  
 13 Peter Gorm Larsen, AU  
 14 Kenneth Lausdahl, AU  
 15 Casper Thule, AU  
 16 Anders Franz Terkelsen, AU  
 17 Carl Gamble, UNEW  
 18 Adrian Pop, LIU  
 19 Etienne Brosse, ST  
 20 Jörg Brauer, VSI  
 21 Florian Lapschies, VSI  
 22 Marcel Groothuis, CLP  
 23 Christian Kleijn, CLP  
 24 Luis Diogo Couto, UTRC

25 **Editors:**

- 26 Victor Bandur, AU

27 **Reviewers:**

- 28 TBD

**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		

29

## 30 Document History

Ver	Date	Author	Description
0.01	11-01-2017	Victor Bandur	Initial version.

## 32 Abstract

33 This deliverable is the user manual for the INTO-CPS tool chain, an update  
34 of deliverable D4.2a [BLL<sup>+</sup>15]. It is targeted at those wishing to make use  
35 of the INTO-CPS technology to design and validate cyber-physical systems.  
36 As a user manual, this deliverable is concerned with those aspects of the tool  
37 chain relevant to end-users, so it is necessarily high-level. Other deliverables  
38 discuss finer details of individual components, including theoretical founda-  
39 tions and software design decisions. Readers interested in this perspective on  
40 the tool chain should consult deliverables D4.2b [PBLG16], D4.2c [BQ16],  
41 D4.2d [LNH<sup>+</sup>16], D5.2a [PLM16], D5.2b [BLM16], D5.2c [BHPG16], D5.2d  
42 [Gam16], D2.2a [ACM<sup>+</sup>16], D2.2b [FCC<sup>+</sup>16], D2.2c [CFTW16] and D2.2d  
43 [CW16].

44	<b>Contents</b>	
45	<b>1 Introduction</b>	<b>7</b>
46	<b>2 Overview of the INTO-CPS Tool Chain</b>	<b>8</b>
47	<b>3 Modelio and SysML for INTO-CPS</b>	<b>10</b>
48	3.1 Creating a New Project . . . . .	11
49	3.2 Exporting modelDescription.xml Files . . . . .	14
50	<b>4 The INTO-CPS Application</b>	<b>20</b>
51	4.1 Introduction . . . . .	20
52	4.2 Projects . . . . .	21
53	4.3 Multi-Models . . . . .	23
54	4.4 Co-simulations . . . . .	28
55	4.5 Additional Features . . . . .	33
56	4.6 The Co-Simulation Orchestration Engine . . . . .	33
57	<b>5 Using the Separate Modelling and Simulation Tools</b>	<b>36</b>
58	5.1 Overture . . . . .	36
59	5.2 20-sim . . . . .	44
60	5.3 OpenModelica . . . . .	53
61	<b>6 Design Space Exploration for INTO-CPS</b>	<b>59</b>
62	6.1 How to Launch a DSE . . . . .	59
63	6.2 Results of a DSE . . . . .	61
64	6.3 How to Edit a DSE Configuration . . . . .	61
65	<b>7 Test Automation and Model Checking</b>	<b>73</b>
66	7.1 Installation of RT-Tester RTT-MBT . . . . .	73
67	7.2 Test Automation . . . . .	74
68	7.3 Model Checking . . . . .	82
69	<b>8 Traceability support for INTO-CPS</b>	<b>90</b>
70	8.1 Overview . . . . .	90
71	8.2 INTO-CPS application . . . . .	90
72	8.3 Modelio . . . . .	91
73	8.4 OpenModelica . . . . .	91
74	8.5 20-sim . . . . .	97
75	<b>9 Code Generation for INTO-CPS</b>	<b>99</b>
76	9.1 Overture . . . . .	99

77	9.2	20-sim . . . . .	101
78	9.3	OpenModelica . . . . .	102
79	9.4	RT-Tester/RTT-MBT . . . . .	102
80	<b>10</b>	<b>Issue handling</b>	<b>102</b>
81	10.1	Are you using the newest INTO-CPS release? . . . . .	102
82	10.2	Has the issue already been reported? . . . . .	103
83	10.3	Reporting a new issue . . . . .	103
84	<b>11</b>	<b>Conclusions</b>	<b>103</b>
85	<b>A</b>	<b>List of Acronyms</b>	<b>109</b>
86	<b>B</b>	<b>Background on the Individual Tools</b>	<b>111</b>
87	B.1	Modelio . . . . .	111
88	B.2	Overture . . . . .	112
89	B.3	20-sim . . . . .	114
90	B.4	OpenModelica . . . . .	115
91	B.5	RT-Tester . . . . .	116
92	<b>C</b>	<b>Underlying Principles</b>	<b>119</b>
93	C.1	Co-simulation . . . . .	119
94	C.2	Design Space Exploration . . . . .	119
95	C.3	Model-Based Test Automation . . . . .	121
96	C.4	Code Generation . . . . .	121

## 97 1 Introduction

98 This deliverable is the user manual for the INTO-CPS tool chain. The  
99 tool chain supports a model-based development and verification approach  
100 for Cyber-Physical Systems (CPSs). Development of CPSs with the INTO-  
101 CPS technology proceeds with the development of constituent models us-  
102 ing established and mature modelling tools. Development also benefits from  
103 support for Design Space Exploration (DSE). The analysis phase is primarily  
104 based on co-simulation of heterogeneous models compliant with version 2.0 of  
105 the Functional-Mockup Interface (FMI) standard for co-simulation [Blo14].  
106 Other verification features supported by the tool chain include hardware-  
107 and software-in-the-loop (HiL and SiL) simulation and model-based test-  
108 ing. Presently there is limited support for Linear Temporal Logic model  
109 checking of discrete models, with further model checking support being de-  
110 veloped.

111 All INTO-CPS tools can be obtained from

112 `http://into-cps.github.io`

113 This is the primary source of information and help for users of the INTO-  
114 CPS tool chain. The structure of the website follows the natural flow of CPS  
115 development with INTO-CPS, and serves as a natural aid in getting started  
116 with the technology. In case access to the individual tools is required, pointers  
117 to each are also provided.

118 **Please note:** This user manual assumes that the reader has a good under-  
119 standing of the FMI standard. The reader is therefore strongly encouraged to  
120 become familiar with Section 2 of deliverable 4.1d [LLW<sup>+</sup>15] for background,  
121 concepts and terminology related to FMI.

122 The rest of this manual is structured as follows:

- 123 • Section 2 provides an overview of the different features and components  
124 of the INTO-CPS tool chain.
- 125 • Section 3 explains the relevant parts of the Modelio SysML modelling  
126 tool.
- 127 • Section 4 explains the different features of the main user interface of  
128 the INTO-CPS tool chain, called the INTO-CPS Application.
- 129 • Section 5 describes the separate modelling and simulation tools used in  
130 elaborating and verifying the different constituent models of a multi-  
131 model.

- 132 • Design Space Exploration (DSE) for INTO-CPS multi-models is pre-  
133 sented in Section 6.
- 134 • Section 7 describes model-based test automation and model checking  
135 in the INTO-CPS context.
- 136 • Section 9 provides a short overview of code generation in the INTO-  
137 CPS context.
- 138 • The appendices are structured as follows:
  - 139 – Appendix A lists the acronyms used throughout this deliverable.
  - 140 – Appendix B gives background information on the individual tools  
141 making up the INTO-CPS tool chain.
  - 142 – Appendix C describes how the individual tools can be obtained.
  - 143 – Appendix D gives background information on the various princi-  
144 ples underlying the INTO-CPS tool chain.

## 145 2 Overview of the INTO-CPS Tool Chain

146 The INTO-CPS tool chain consists of several special-purpose tools from a  
147 number of different providers. Note that it is an open tool chain so it is  
148 possible to incorporate other tools that also support the FMI standard for  
149 co-simulation and we have already tested this with numerous external tools  
150 (both commercial as well as open-source tools). The constituent tools are  
151 dedicated to the different phases of co-simulation activities. They are dis-  
152 cussed individually through the course of this manual. An overview of the  
153 tool chain is shown in Figure 1. The main interface to an INTO-CPS co-  
154 simulation activity is the INTO-CPS Application. This is where the user  
155 can design co-simulations from scratch, assemble them using existing FMUs  
156 and configure how simulations are executed. The result is a co-simulation  
157 *multi-model*.

158 The design of a multi-model is carried out visually using the Modelio SysML  
159 tool, in accordance with the SysML/INTO-CPS profile described in D2.2a  
160 [ACM<sup>+</sup>16]. Here one can either design a multi-model from scratch by specify-  
161 ing the characteristics and connection topology of Functional Mockup Units  
162 (FMUs) yet to be developed, or import existing FMUs so that the connections  
163 between them may be laid out visually. The result is a SysML multi-model of  
164 the entire co-simulation, expressed in the SysML/INTO-CPS profile. In the

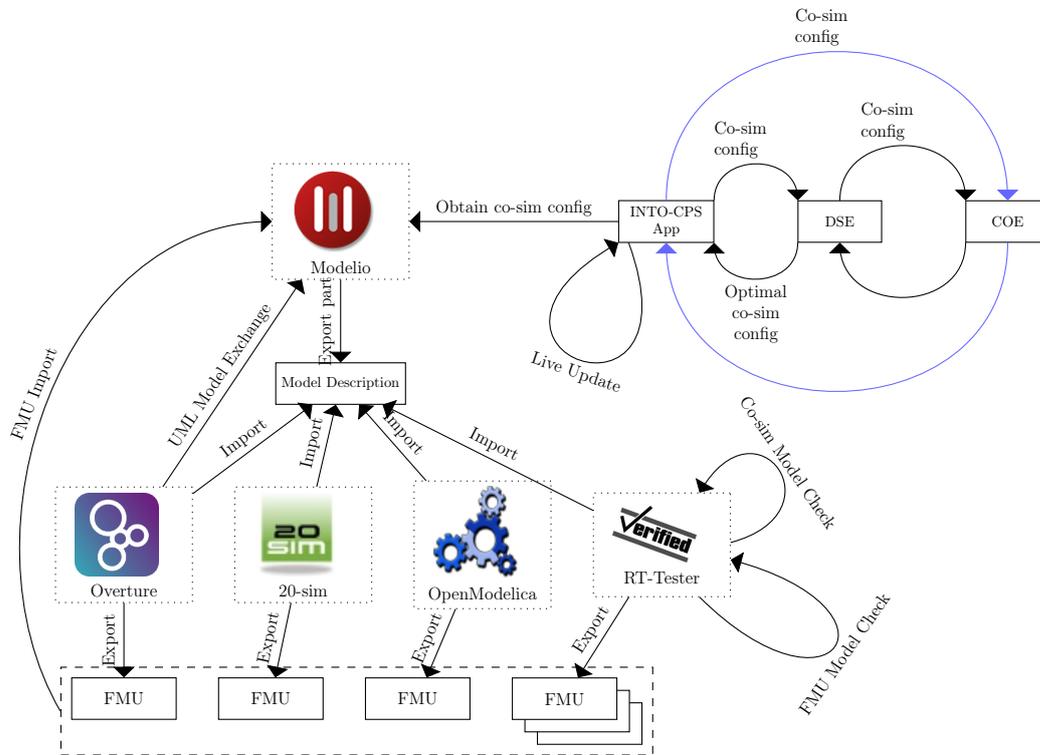


Figure 1: Overview of the structure of the INTO-CPS tool chain.

165 former case, where no FMUs exist yet, a number of `modelDescription`  
 166 `.xml` files are generated from this multi-model which serve as the starting  
 167 point for constituent model construction inside each of the individual simu-  
 168 lation tools, leading to the eventual FMUs.

169 Once a multi-model has been designed and populated with concrete FMUs,  
 170 the Co-simulation Orchestration Engine (COE) can be invoked to execute  
 171 the co-simulation. The COE controls all the individual FMUs in order to  
 172 carry out the co-simulation. In the case of tool-wrapper FMUs, the model  
 173 inside each FMU is simulated by its corresponding simulation tool. The tools  
 174 involved are Overture [LBF<sup>+</sup>10], 20-sim [Con13] and OpenModelica [Lin15].  
 175 RT-Tester is not under the direct control of the COE at co-simulation time, as  
 176 its purpose is to carry out testing and model checking rather than simulation.  
 177 The user can control a co-simulation, for instance by running it with different  
 178 simulation parameter values and observing the effect of the different values  
 179 on the co-simulation outcome.

180 Alternatively, the user has the option of exploring optimal simulation pa-  
 181 rameter values by entering a Design Space Exploration phase. In this mode,

182 ranges are defined for various parameters which are explored, in an intel-  
 183 ligent way, by a design space exploration engine that searches for optimal  
 184 parameter values based on defined optimization conditions. This engine in-  
 185 teracts directly with the COE and itself controls the conditions under which  
 186 the co-simulation is executed.

### 187 3 Modelio and SysML for INTO-CPS

188 The INTO-CPS tool chain supports a model-based approach to the develop-  
 189 ment and validation of CPS. The Modelio tool and its SysML/INTO-CPS  
 190 profile extension provide the diagramming starting point. This section de-  
 191 scribes the Modelio extension that provides INTO-CPS-specific modelling  
 192 functionality to the SysML modelling approach.

193 The INTO-CPS extension module is based on the Modelio SysML extension  
 194 module, and extends it in order to fulfill INTO-CPS modelling requirements  
 195 and needs. Figure 2 shows an example of a simple INTO-CPS Architecture  
 Structure Diagram under Modelio. This diagram shows a *System*, named

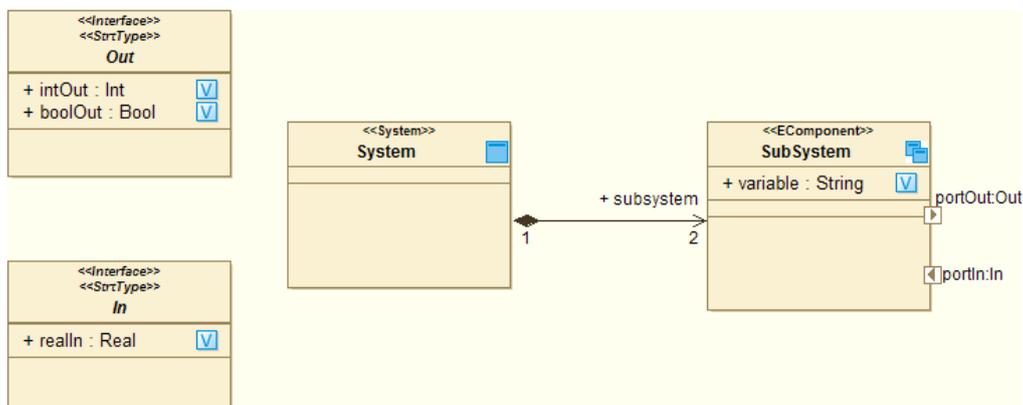


Figure 2: Example INTO-CPS multi-model.

196 “System”<sup>1</sup>, composed of two *EComponents* of kind *Subsystem*, named “Sub-  
 197 System”<sup>2</sup>. These *Subsystems* have an internal *Variable* called “variable” of  
 198 type *String* and expose two *FlowPorts* named “portIn” and “portOut”. The  
 199 type of data going through these ports is respectively defined by types *In*  
 200

<sup>1</sup>An abstract description of an INTO-CPS multi-model.

<sup>2</sup>Abstract descriptions of INTO-CPS constituent models.

201 and *Out* of kind *StrtType*. More details on the SysML/INTO-CPS profile  
 202 can be found in deliverable D2.2a [ACM<sup>+</sup>16].

203 Figure 3 illustrates the main graphical interface after Modelio and the INTO-  
 CPS extension have been installed. Of all the panes, the following three are

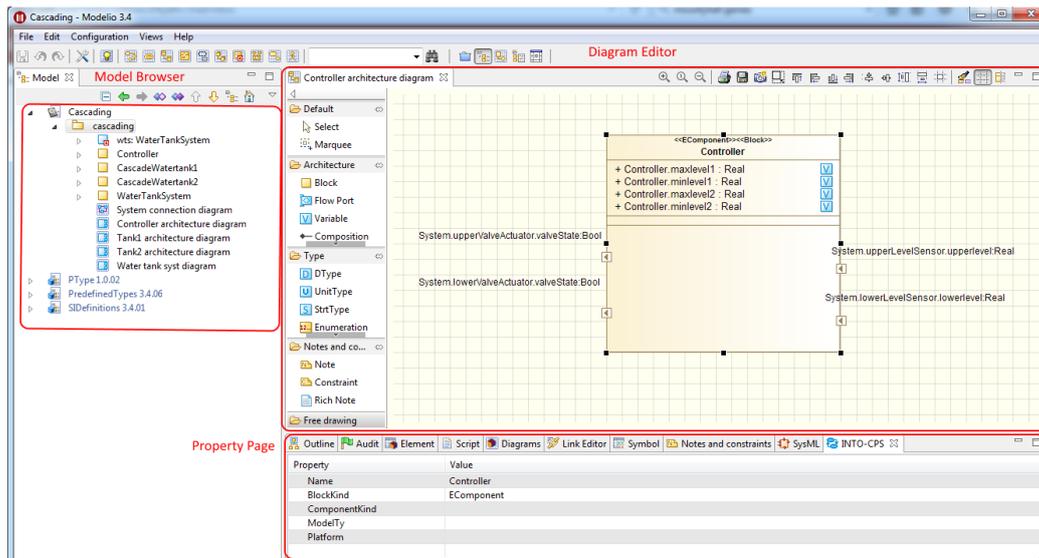


Figure 3: Modelio for INTO-CPS.

204  
 205 most useful in the INTO-CPS context.

- 206 1. The Modelio model browser, which lists all the elements of your model  
 207 in tree form.
- 208 2. The diagram editor, which allows you to create INTO-CPS design archi-  
 209 tectures and connection diagrams.
- 210 3. The INTO-CPS property page, in which values for properties of INTO-  
 211 CPS subsystems are specified.

### 212 3.1 Creating a New Project

213 In the INTO-CPS Modelling workflow described in Deliverable D3.2a [FGPP16],  
 214 the first step will be to create, as depicted in Figure 4, a Modelio project:

- 215 1. Launch Modelio.
- 216 2. Click on *File* → *Create a project...*

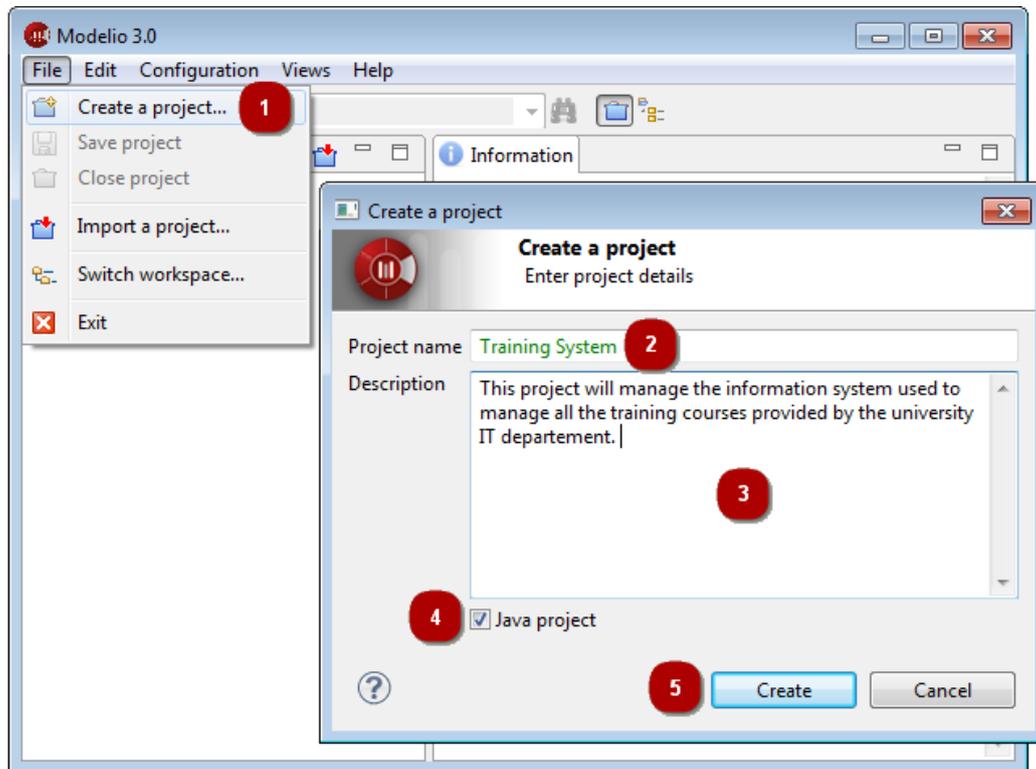


Figure 4: Creating a new Modelio project.

- 217 3. Enter the name of the project.
- 218 4. Enter the description of the project.
- 219 5. If it is envisaged that the project will be connected to a Java develop-  
220 ment workflow in the future (unrelated to INTO-CPS), you can choose  
221 to include the Java Designer module by selecting *Java Project*, other-  
222 wise de-select this option.
- 223 6. Click on *Create* to create and open the project.

224 Once you have successfully created a Modelio project, you have to install  
225 the Modelio extensions required for INTO-CPS modelling, *i.e.* both Modelio  
226 SysML and INTO-CPS extensions, as described at

227 <http://into-cps.github.io>

228 If both modules have been correctly installed, you should be able to create,  
229 under any package, an INTO-CPS Architecture Structure Diagram in order  
230 to model the first subsystem of your multi-model. For that, in the Mode-

231 lio model browser, right click on a *Package* element then in the *INTO-CPS*  
 232 entry, choose *Architecture Structure Diagram* as shown in Figure 5. Figure 6 represents an example of an Architecture Structure Diagram. Besides

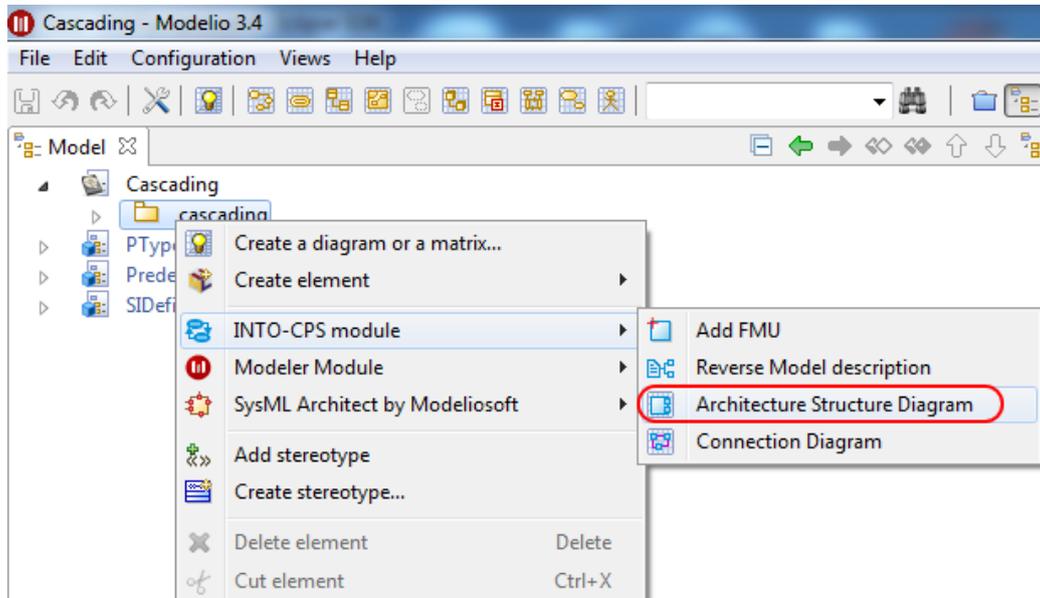


Figure 5: Creating an Architecture Structure diagram.

233  
 234 creating an Architecture Structure Diagram from scratch, the INTO-CPS  
 235 extension allows the user to create it from an existing modelDescription  
 236 .xml file. A modelDescription.xml file is an artifact defined in the  
 237 FMI standard which specifies, in XML format, the public interface of an  
 238 FMU. To import a modelDescription.xml file,

- 239 1. Right click in the Modelio model browser on a *Package* element, then  
 240 in the *INTO-CPS* entry choose *Import Model description*, as shown in  
 241 Figure 7.
- 242 2. Select the desired modelDescription.xml file in your installation  
 243 and click on *Import* (Figure 8).

244 This import command creates an Architecture Structure Diagram describing  
 245 the interface of an INTO-CPS *block* corresponding to the modelDescrip-  
 246 tion.xml file imported, cf. Figure 9. Once you have created several such  
 247 blocks, either from scratch or by importing modelDescription.xml files,  
 248 you must eventually connect instances of them in an INTO-CPS Connection  
 249 Diagram. To create an INTO-CPS Connection diagram, as for an INTO-  
 250 CPS Architecture Structure Diagram, right click on a *Package* element, then

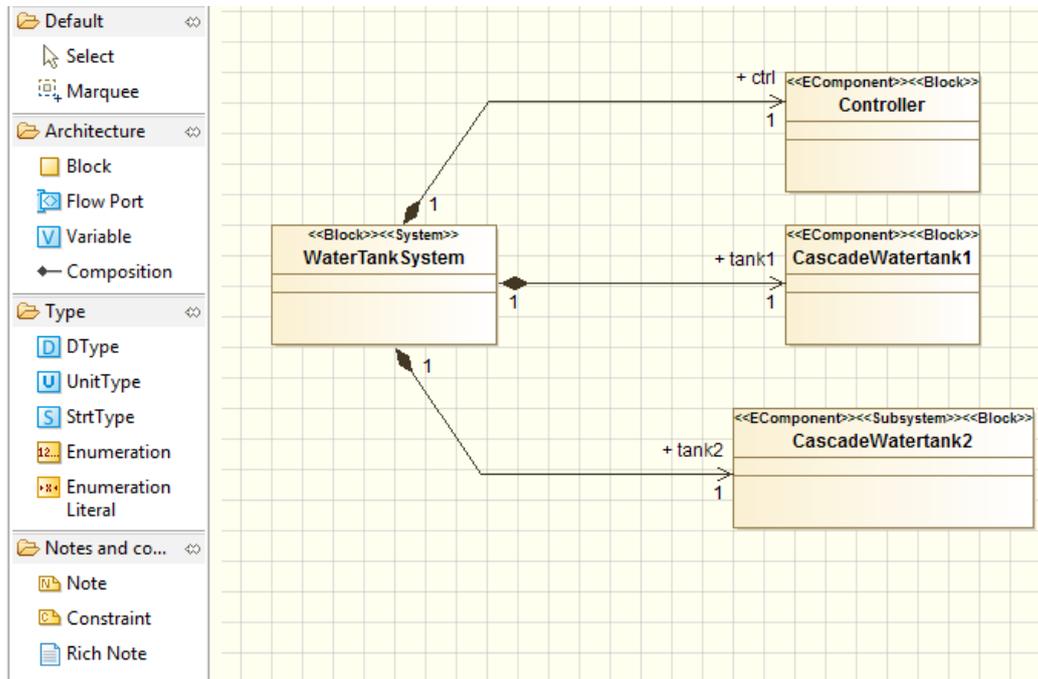


Figure 6: Example Architecture Structure diagram.

251 in the *INTO-CPS* entry choose *Connection Diagram*, as shown in Figure 10.  
 252 Figure 11 shows the result of creating such a diagram. Once you have created  
 253 all desired block instances and their ports by using the dedicated command in  
 254 the Connection Diagram palette, you will be able to model their connections  
 255 by using the connector creation command (Figure 12). At this point your  
 256 blocks have been defined and the connections have been set. The next step  
 257 is to simulate your multi-model using the app. For that you must first gener-  
 258 erate a configuration file from your Connection diagram. Select the desired  
 259 Connection diagram, right click on it and in the *INTO-CPS* entry choose  
 260 *Generate configuration*, as shown in Figure 13. In the final step, choose a  
 261 relevant name and click on *Generate*.

## 262 3.2 Exporting `modelDescription.xml` Files

263 The SysML Connection diagram defines the components of the system and  
 264 their connections. The internals of these block instances are created in  
 265 the various modeling tools and exported as FMUs. The modeling tools  
 266 Overture, 20-sim and OpenModelica support importing the interface def-  
 267 inition (ports) of the blocks in the Connection diagram by importing a

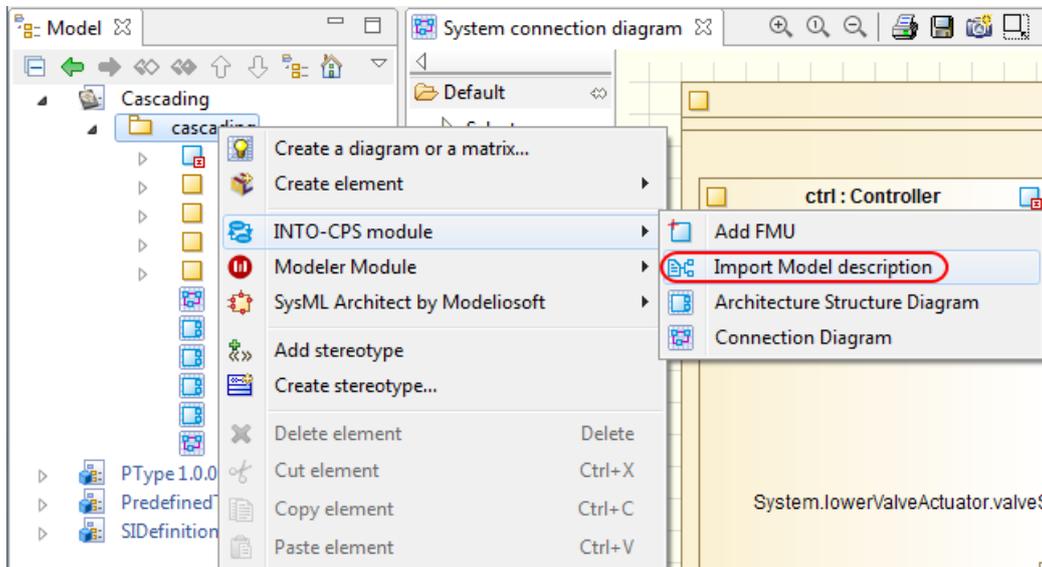


Figure 7: Importing an existing model description.

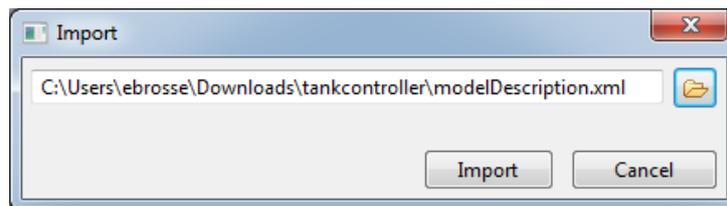


Figure 8: Model description selection.

268 modelDescription.xml file containing the block name and its interface  
 269 definition.

270 Follow these steps to export a modelDescription.xml file from Mode-  
 271 lio:

- 272 1. In Modelio, right-click on the model block in the tree.
- 273 2. Select *INTO-CPS* → *Generate Model Description* (see Figure 14).
- 274 3. Choose a file name containing the text “modelDescription.xml” and  
 275 click *Export* (see Figure 15).

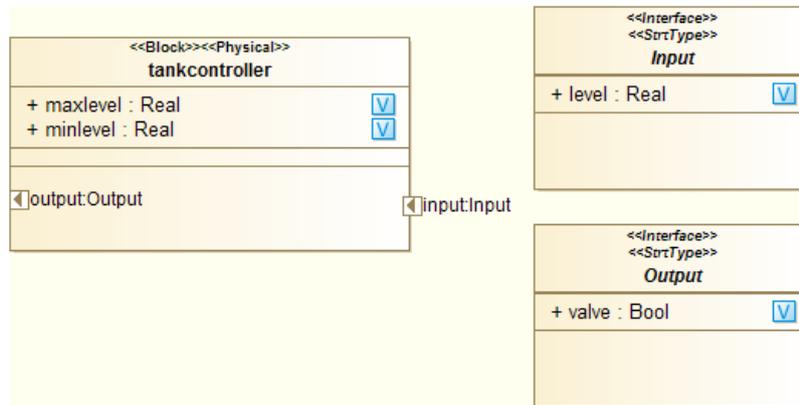


Figure 9: Result of model description import.

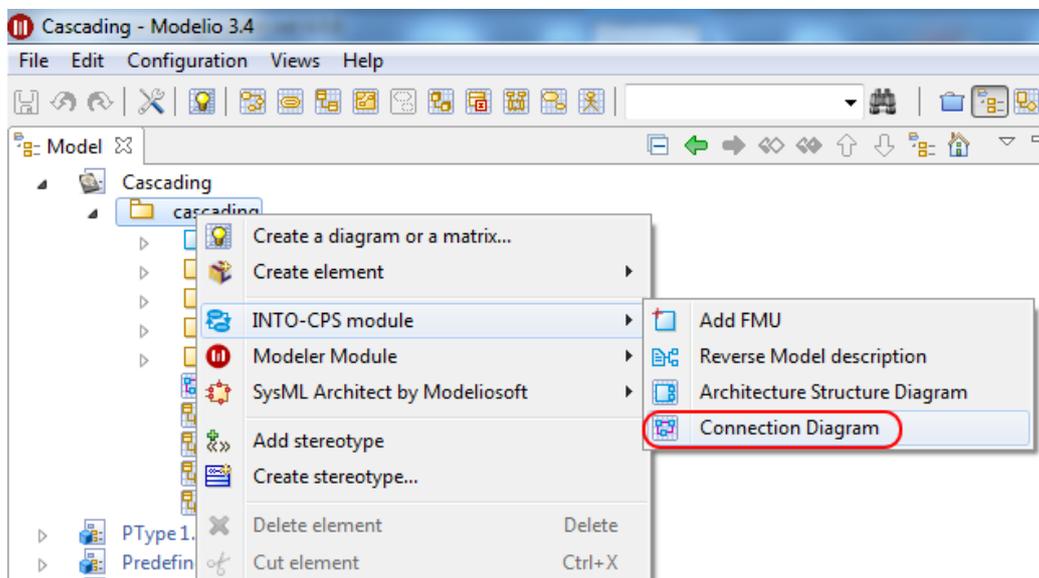


Figure 10: Creating a Connection diagram.

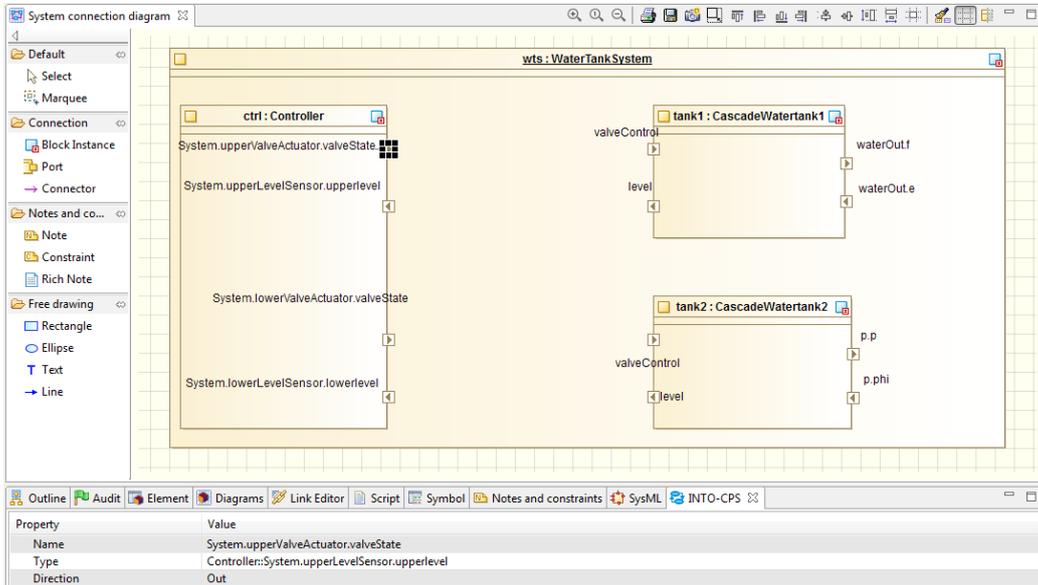


Figure 11: Unpopulated Connection diagram.

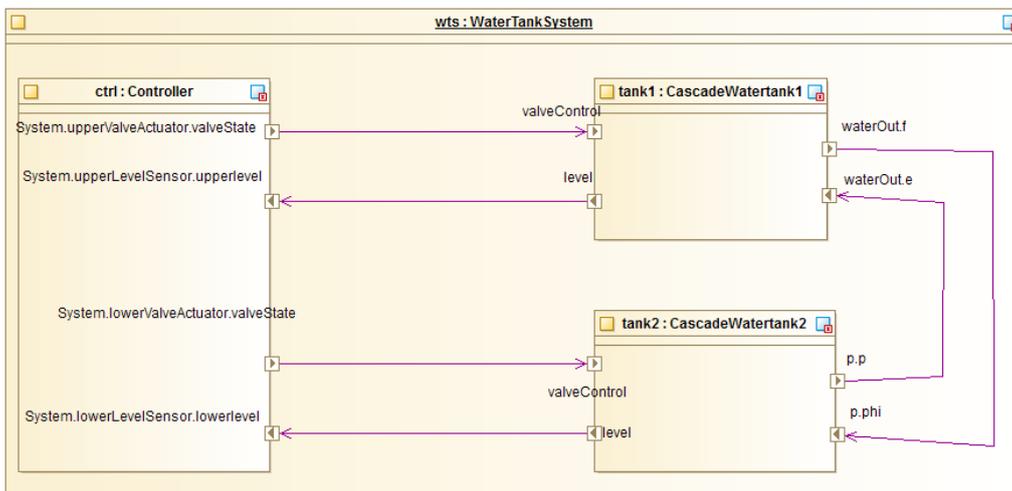


Figure 12: Populated Connection diagram.

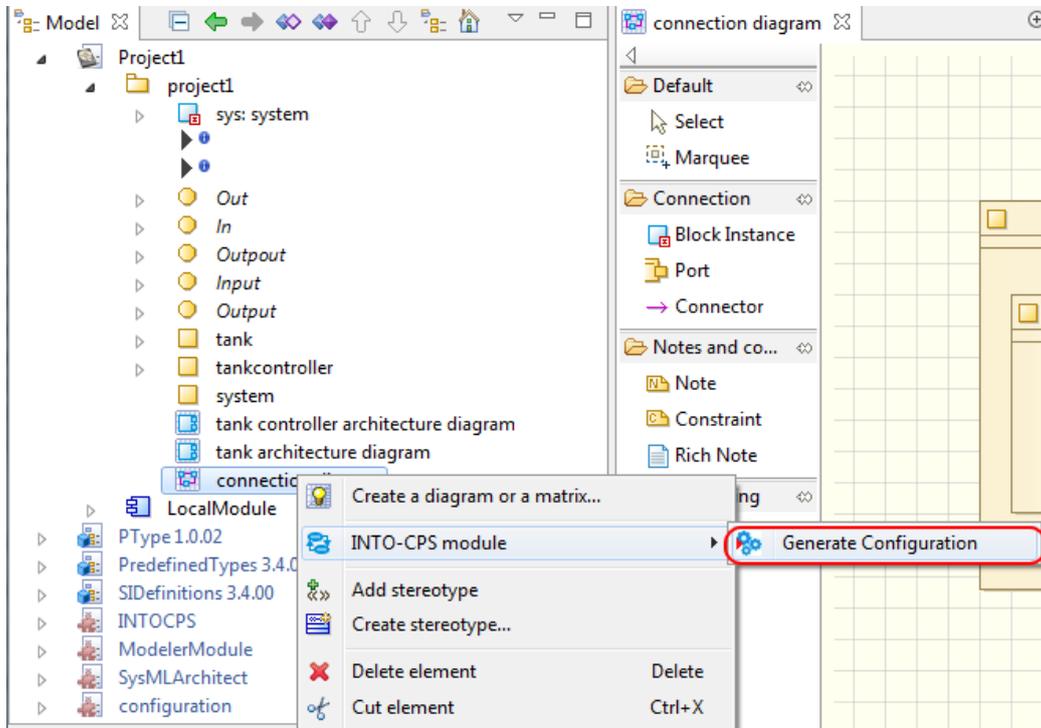


Figure 13: Generating a configuration file.

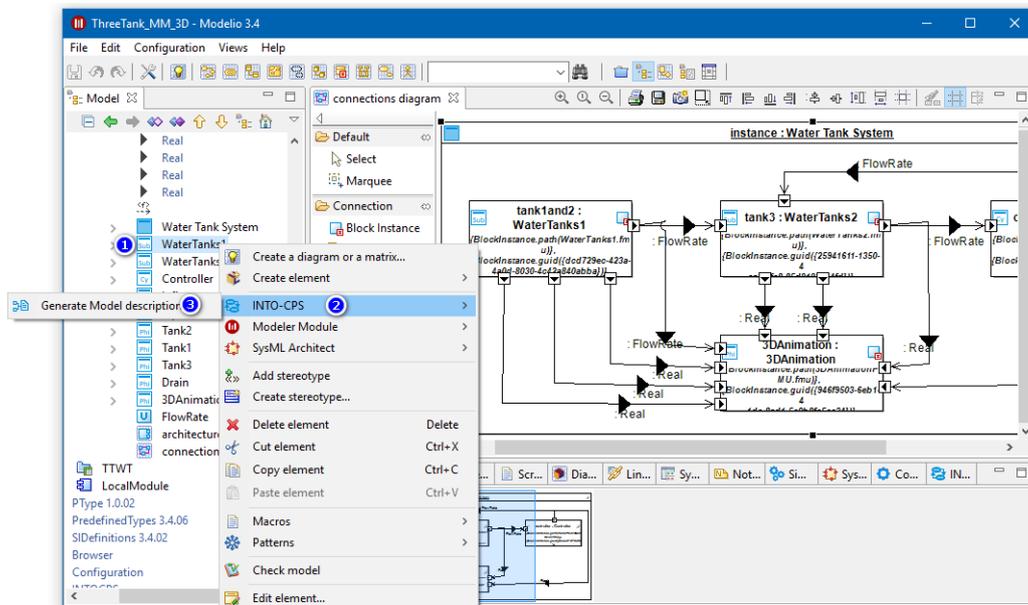


Figure 14: Exporting a modelDescription.xml file.

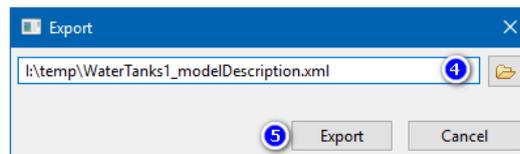


Figure 15: Naming the model description file.

## 276 4 The INTO-CPS Application

277 This section describes the INTO-CPS Application (here referred to as *the*  
278 *app*), the primary gateway to the INTO-CPS tool chain. Section 4.1 gives  
279 an introductory overview of the app. Section 4.2 describes how the app  
280 can be used to create new INTO-CPS co-simulation projects. Section 4.3  
281 describes how multi-models can be assembled. Section 4.4 describes how co-  
282 simulations are configured, executed and visualized. Section 4.5 lists some  
283 additional useful features of the app, while Section 4.6 describes how the  
284 co-simulation engine itself can be started manually, for specialist use.

### 285 4.1 Introduction

286 The app is the front-end of the entire INTO-CPS tool chain. The app defines  
287 a common INTO-CPS project and it is the easiest way to configure and  
288 execute co-simulations. Certain features in the tool chain are only accessible  
289 through the app. Those features will be explained in their own sections  
290 of the user manual. This section introduces the app and its basic features  
291 only.

292 Releases of the app can be downloaded from:

293 `https://github.com/into-cps/intocps-ui/releases`

294 Four variants are available:

- 295 • `-darwin-x64.zip` – MacOS version
- 296 • `-linux-x64.zip` – Linux (64 bit) version
- 297 • `-win32-ia32.zip` – Windows (32 bit) version
- 298 • `-win32-x64.zip` – Windows (64 bit) version

299 The app itself has no dependencies and requires no installation. Simply unzip  
300 it and run the executable. However, certain app features require Git<sup>3</sup> and  
301 Java 8<sup>4</sup> to be already installed.

---

<sup>3</sup><https://git-scm.com/>

<sup>4</sup><http://www.oracle.com/technetwork/java/javase/overview/java8-2100321.html>

## 302 4.2 Projects

303 An INTO-CPS project contains all the artifacts used and produced by the  
 304 tool chain. The project artifacts are grouped into folders. You can create  
 305 as many folders as you want and they will all be displayed in the project  
 306 browser. The default set of folders for a new project, shown in Figure 16, is:

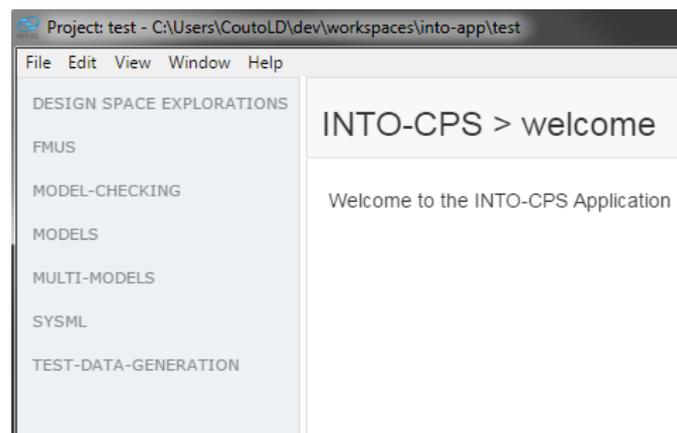


Figure 16: INTO-CPS project shown in the project browser.

307

308 **Design Space Explorations** Scripts and configuration files for performing  
 309 DSE experiments.

310 **FMUs** FMUs for the constituent models of the project.

311 **Model Checking** Configuration files for performing Model Checking exper-  
 312 iments.

313 **Models** Sources for the constituent models of the project.

314 **Multi-Models** The multi-models of the project, using the project FMUs.  
 315 This folder also holds configuration files for performing co-simulations.

316 **SysML** Sources for the SysML model that defines the architecture and con-  
 317 nections of the project multi-model.

318 **Test-Data-Generation** Configuration files for performing test data gener-  
 319 ation experiments.

320 In order to create a new project, select *File* → *New Project*, as shown in  
 321 Figure 17a. This opens the dialog shown in Figure 17b, where you must  
 322 choose the project name and location – the chosen location will be the root

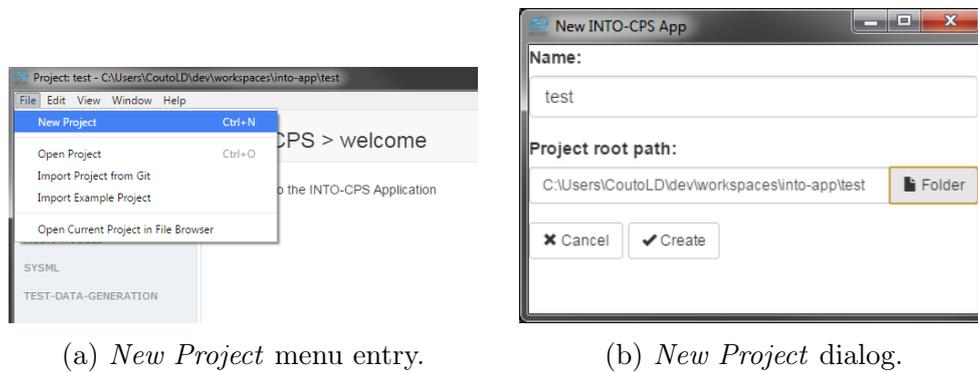


Figure 17: Creating a new INTO-CPS project.

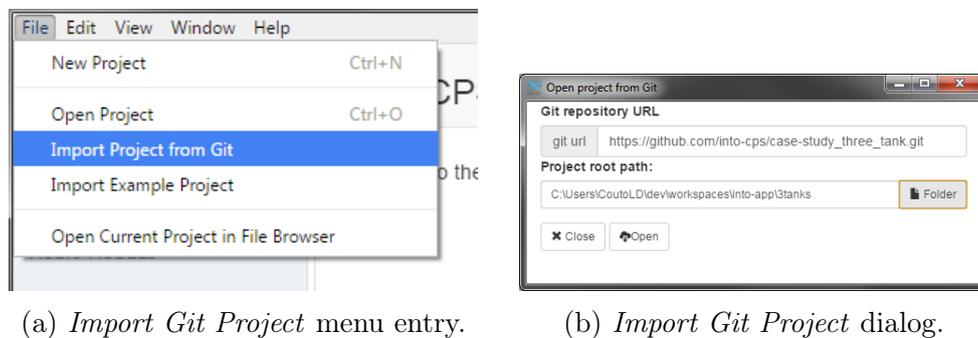


Figure 18: Importing a Git project.

323 of the project, so you should manually create a new folder for it. To open an  
 324 existing project, select *File* → *Open Project*, then navigate to the project's  
 325 root folder and open it.

326 To import a project stored in the Git version control system, select *File* →  
 327 *Import Project from Git*, as shown in Figure 18a. This opens the dialog shown  
 328 in Figure 18b, where you must choose the project location and also provide  
 329 the Git URL. The project is checked out using Git, so any valid Git URL  
 330 will work. You must also have Git available in your PATH environment  
 331 variable in order for this feature to work. It is possible to import several  
 332 public example projects that show off the various features of the INTO-CPS  
 333 tool chain. These examples are described in Deliverable D3.5 [PGP<sup>+</sup>16]. To  
 334 import an example, select *File* → *Import Example Project*, as shown in Figure  
 335 19a. This opens the dialog box shown in Figure 19b, where you must select  
 336 which example to import and a project location. The example is checked out  
 337 via Git, so you must have Git available in your path in order for this feature  
 338 to work. For both Git projects and examples, once you begin the import

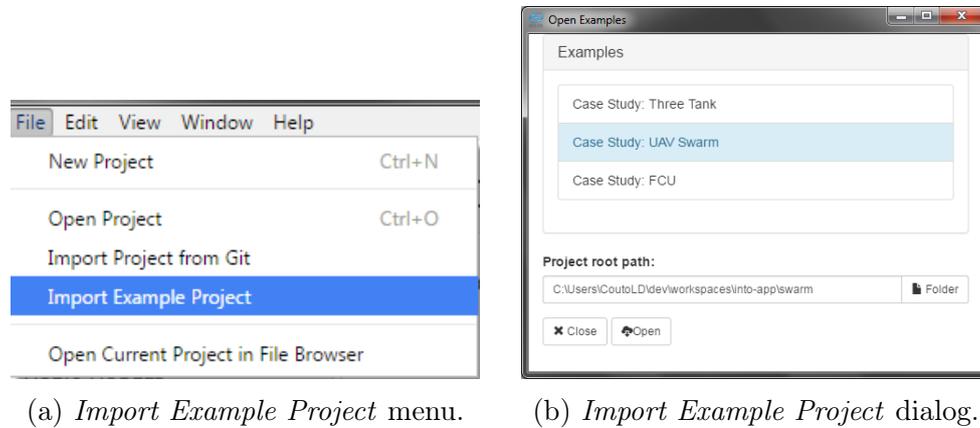


Figure 19: Importing examples.

339 process, a process dialog is displayed, as shown in Figure 20.

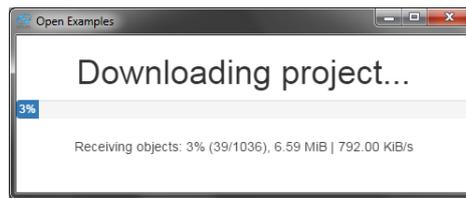


Figure 20: Progress of project imports through Git.

### 340 4.3 Multi-Models

341 For any given project, the app allows you to create and edit multi-models  
 342 and co-simulation configurations. To create a new multi-model, right click  
 343 the *Multi-models* node in the project browser and select *New multi-model*,  
 344 as shown in Figure 21. After creation, the new multi-model is automatically  
 345 opened for editing. To select an existing multi-model for editing, double-  
 346 click it. Once a multi-model is open, the multi-model view, shown in Figure  
 347 22 is displayed. The top box, *Overview*, displays an overview of the input  
 348 and output variables in the FMUs, as shown in Figure 23. The bottom box,  
 349 *Configuration*, enables the user to configure the multi-model. In order to  
 350 configure a multi-model, it must first be unlocked for editing by clicking the  
 351 *Edit* button at the bottom of the *Configuration* box. There are four main  
 352 areas dedicated to configuring various aspects of a multi-model.

353 The *FMUs* area, shown in Figure 24, allows you to remove or add FMUs

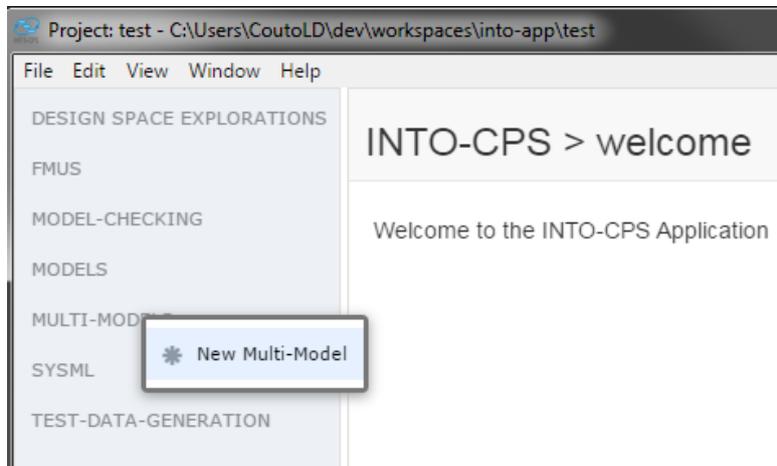


Figure 21: Creating a new multi-model.

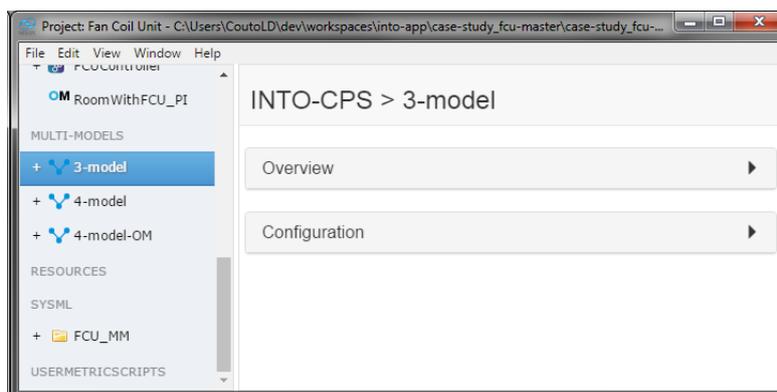


Figure 22: Main multi-model view.

354 and to associate the FMUs with their files by browsing to, or typing, the  
 355 path of the FMU file. For each FMU file a marker is displayed indicating  
 356 whether the FMU is supported by the app and can be used for co-simulation  
 357 on the current platform. The *FMU instances* area, shown in Figure 25,  
 358 allows you to create or remove FMU instances and name them. A multi-  
 359 model consists of one or more interconnected instances of various FMUs.  
 360 More than one instance may be created for a given FMU. As a convenient  
 361 workflow shortcut, the *Connections* area, shown in Figure 26, allows you  
 362 to connect output variables from an FMU instance into input variables of  
 363 another:

- 364 1. Click the desired output FMU instance in the first column. The output  
 365 variables for the selected FMU appear in the second column.

Overview	
Outputs	Inputs
{environmentFMU}.env.RAT_OUT	{controllerFMU}.controller.RATSP
{environmentFMU}.env.OAT_OUT	{roomheatingFMU}.room.OAT
{controllerFMU}.controller.valveOpen	{roomheatingFMU}.room.valveopen
{controllerFMU}.controller.fanSpeed	{roomheatingFMU}.room.fanspeed

Figure 23: Multi-model overview.

Configuration			
FMUs +			
Keys	Paths		
control	FCUController_Limited.fmu	 File	 Folder
	<b>Supported</b>		
room	RoomHeating.fmu	 File	 Folder
	<b>Supported</b>		
env	Environment.fmu	 File	 Folder
	<b>Supported</b>		

Figure 24: FMUs configuration.

- 366 2. Click the desired output variable in the second column. The input  
367 instances appear in the third column.
- 368 3. Click the desired FMU input instance in the third column. The input  
369 variables for the selected FMU appear in the fourth column.
- 370 4. Check the box for the desired input variable in the fourth column.

371 This facility makes it unnecessary to return to Modelio whenever small  
372 changes must be made to the connection topology of the multi-model. The  
373 *Initial values of parameters* area, shown in Figure 27, allows you to set the

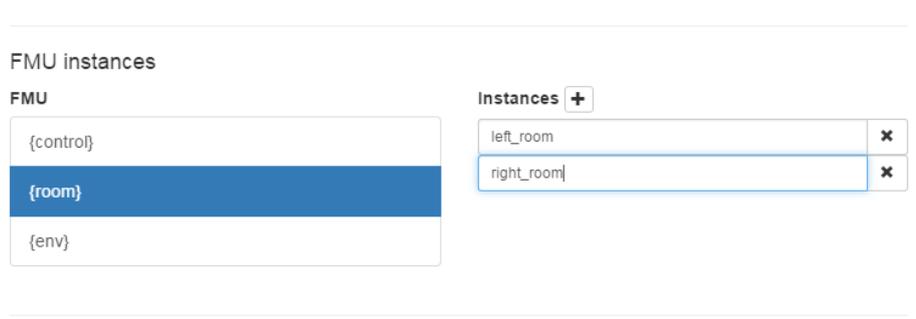


Figure 25: FMU instances configuration.

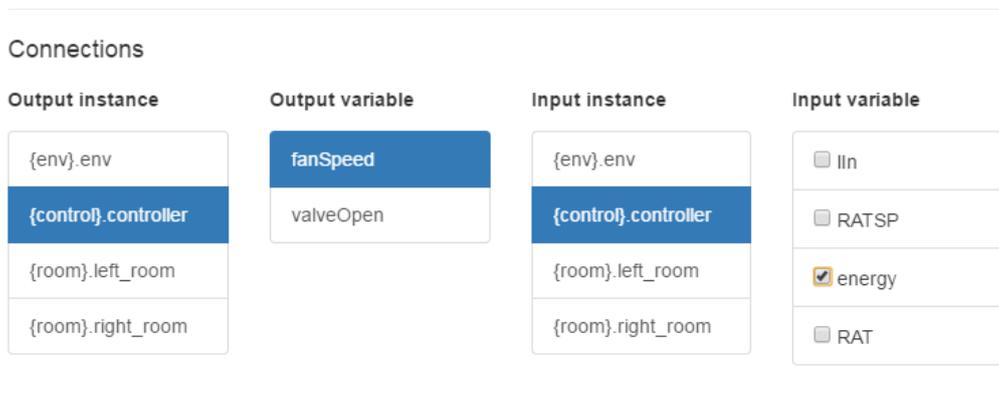


Figure 26: Connections configuration.

374 initial values of any parameters defined in the FMUs:

- 375 1. Click the desired FMU instance in the *Instance Column*.
- 376 2. Select the desired parameter in the *Parameters* dropdown box and click  
377 *Add*.
- 378 3. Type the parameter value in the box that appears.

379 Once the multi-model configuration is complete, click the *Save* button at the  
380 bottom of the *Configuration* box.

Initial values of parameters

Instance	Parameters	
{env}.env		controllerFrequency
<b>{control}.controller</b>		<b>controllerFrequency</b>
{room}.left_room		
{room}.right_room		

(a) Parameter selection.

Initial values of parameters

Instance	Parameters	
{env}.env		
<b>{control}.controller</b>		
{room}.left_room		
{room}.right_room		

(b) Parameter value input.

Figure 27: Initial values of parameters configuration.

## 381 4.4 Co-simulations

382 With the INTO-CPS tool chain it is possible to distribute a co-simulation  
383 across several computing nodes such that FMUs need not be co-located with  
384 the COE on the same node. This capability caters to situations in which  
385 FMUs are restricted to simulation on specific platforms for reasons of legacy  
386 technology, licensing *etc.* In the current version of the tool chain this func-  
387 tionality is not fully integrated with the app, and requires the user to start  
388 the simulation procedure manually. This is discussed in Section 4.6 below.  
389 The remainder of this section discusses standard co-simulations on a single  
390 computing node.

391 To execute co-simulations of a multi-model, a co-simulation configuration is  
392 needed. To create a co-simulation configuration, right click the desired multi-  
393 model and select *Create Co-Simulation Configuration*, as shown in Figure  
394 28. After creation, the new configuration automatically opens for editing.  
To select an existing co-simulation configuration, double-click it. Once a

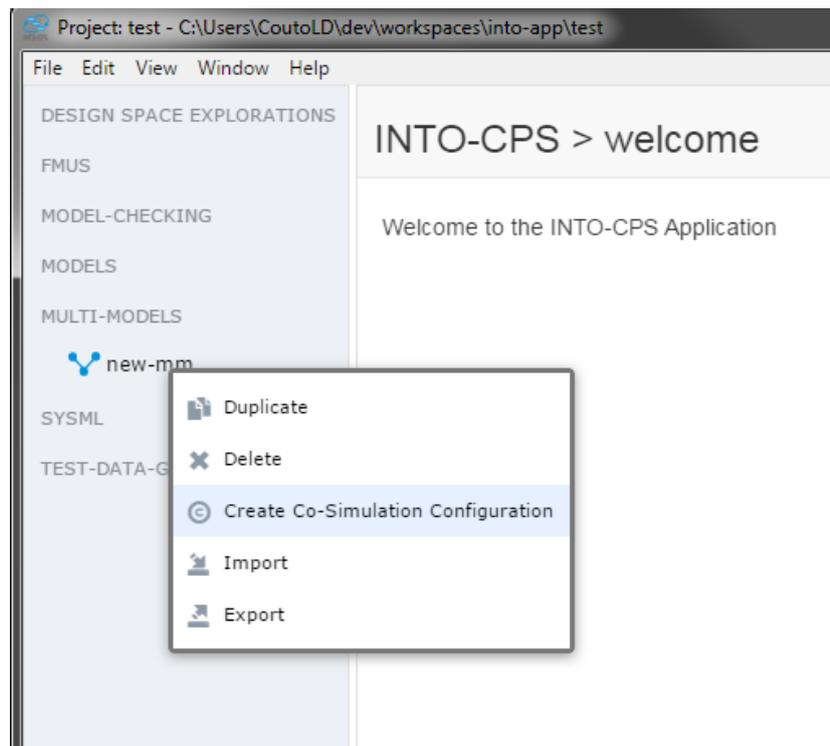


Figure 28: Creating a co-simulation configuration.

395 configuration is open, the co-simulation configuration, shown in Figure 29, is  
396

397 displayed. The top box, *Configuration*, lets you configure the co-simulation.  
 The bottom box, *Simulation*, lets you execute the co-simulation. In order to

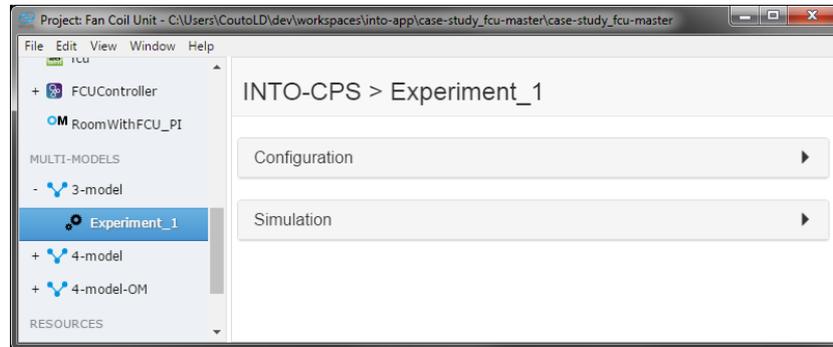


Figure 29: Main co-simulation configuration view.

398  
 399 configure a co-simulation, the configuration must first be unlocked for editing  
 400 by clicking the *Edit* button at the bottom of the *Configuration* box. There  
 401 are three things to configure for a co-simulation, discussed next.

402 The top area, shown in Figure 30, allows you to select the start and end  
 time for the co-simulation as well as the master algorithm to be used. For

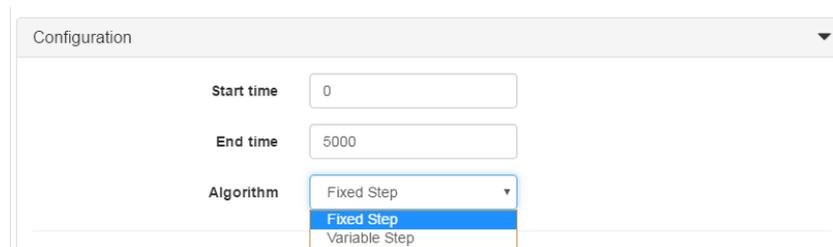
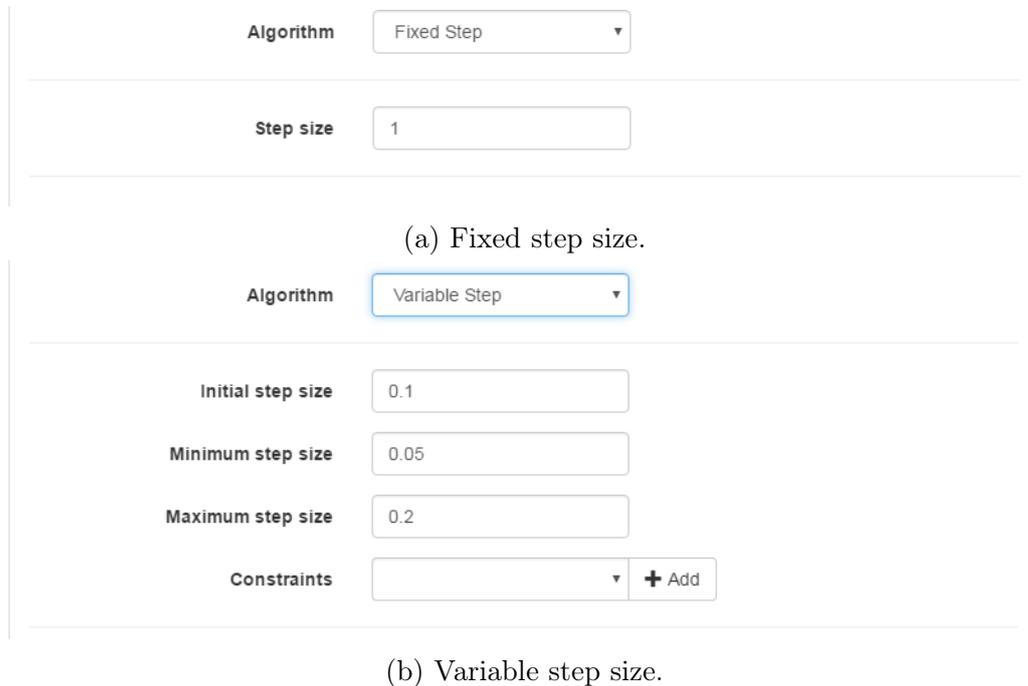


Figure 30: Start/End time and master algorithm configuration.

403  
 404 every algorithm, there are configuration parameters that can be set. These  
 405 are displayed below the top area, as shown in Figure 31. These parameters  
 406 differ with the master algorithm chosen. The *Livestream Configuration* area,  
 407 shown in Figure 32, allows you to select which variables to live stream and  
 408 plot during the co-simulation. Every instance in the multi-model is displayed  
 409 and the output variables are shown for each instance. Check the box for each  
 410 variable that you wish to live stream. Once the co-simulation configuration is  
 411 complete, click the *Save* button at the bottom of the *Configuration* box.

412 The *Simulation* box, shown in Figure 33, allows you to launch a co-simulation.  
 413 To run a co-simulation, the COE must be online. The area at the top of the



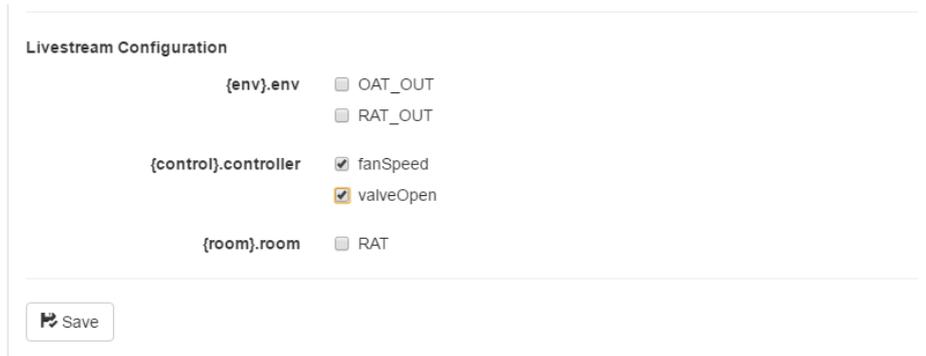
(a) Fixed step size.

(b) Variable step size.

The figure shows two screenshots of a configuration interface. Screenshot (a) shows the 'Algorithm' dropdown set to 'Fixed Step' and the 'Step size' input field set to '1'. Screenshot (b) shows the 'Algorithm' dropdown set to 'Variable Step', with 'Initial step size' at 0.1, 'Minimum step size' at 0.05, and 'Maximum step size' at 0.2. There is also a 'Constraints' dropdown and an '+ Add' button.

Figure 31: Master algorithm configuration.

414 *Simulation* box displays the status of the COE. If the COE is offline, you  
415 may click the *Launch* button to start it. Once a co-simulation is in progress,  
416 any variables chosen for live streaming are plotted in real time in the simula-  
417 tion box, as shown in Figure 34. A progress bar is also displayed. When the  
418 simulation is complete, the live stream plot can be explored or exported as  
419 a PNG image. In addition, an `outputs.csv` file is created containing the  
420 values of every FMU output variable at every point in time in the simula-  
421 tion. This file can be double-clicked and it will open with the default system  
422 program for CSV files. It can also be imported into programs such as R,  
423 MATLAB or Excel for more complex analysis. Furthermore, it is possible  
424 to add a Post-processing script that receives the csv file name and the total  
425 simulation time as arguments. It is also possible to configure the amount of  
426 logging performed by the Co-Simulation Orchestration Engine.

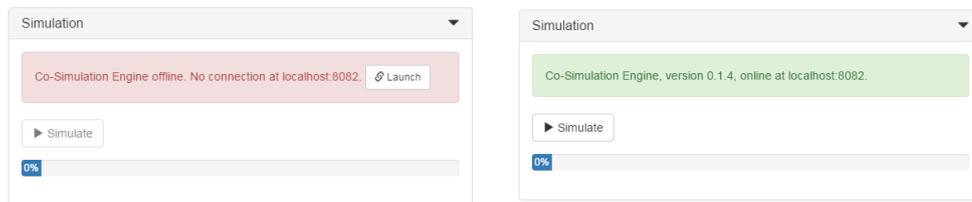


Livestream Configuration

<code>{env}.env</code>	<input type="checkbox"/> OAT_OUT
	<input type="checkbox"/> RAT_OUT
<code>{control}.controller</code>	<input checked="" type="checkbox"/> fanSpeed
	<input checked="" type="checkbox"/> valveOpen
<code>{room}.room</code>	<input type="checkbox"/> RAT

 Save

Figure 32: Livestream configuration.



(a) COE offline.

(b) COE online.

Figure 33: Launching a co-simulation.

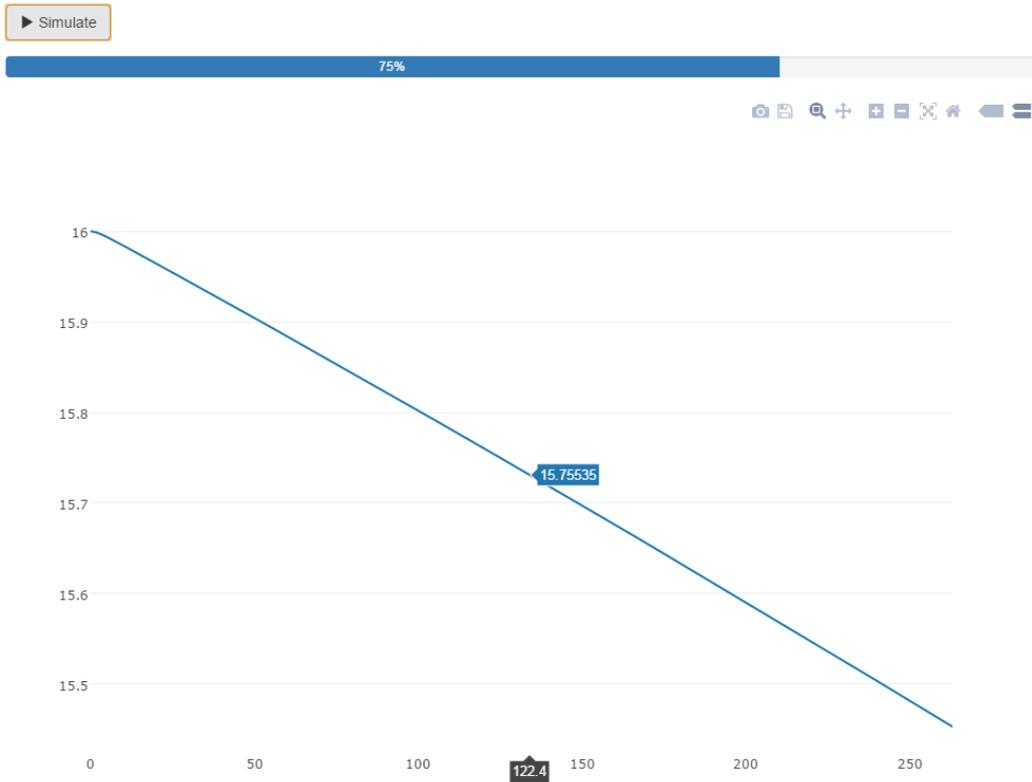


Figure 34: Live stream variable plot.

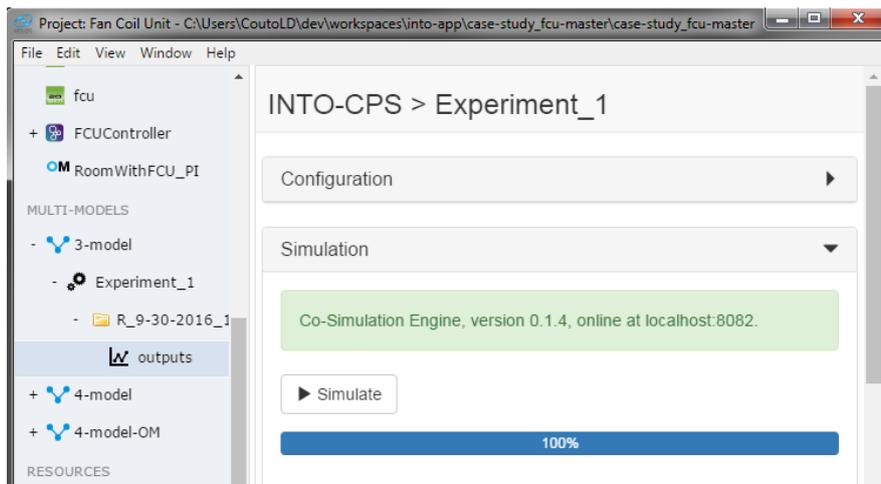


Figure 35: Co-simulation results file.

## 427 4.5 Additional Features

428 The app has several secondary features, most of them accessible through  
429 the *Window* menu, as shown in Figure 36. They are briefly explained be-  
430 low.

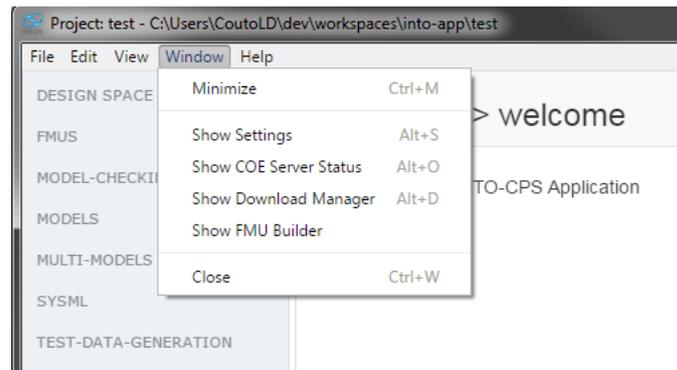


Figure 36: Additional features.

431 **Show Settings** displays a settings page where various default paths can  
432 be set. Development mode can also be enabled from this page, but this  
433 feature is primarily meant to be used by app developers for testing.

434 **Show COE Server Status** displays a page where you can launch and  
435 stop the COE as well as observe its log.

436 **Show Download Manager** displays a page where installers can be down-  
437 loaded for the various tools of the INTO-CPS tool chain, including the  
438 COE.

439 **Show FMU Builder** displays a page that links to a service where source  
440 code FMUs can be uploaded and cross-compiled for various platforms.  
441 Note that this is not a secure service and users are discouraged from  
442 uploading proprietary FMUs.

## 443 4.6 The Co-Simulation Orchestration Engine

444 The heart of the INTO-CPS Application is the Co-Simulation Orchestration  
445 Engine (COE). This is the engine that orchestrates the various simulation  
446 tools (described below), carrying out their respective roles in the overall co-  
447 simulation. It runs as a stand-alone server hosting the co-simulation API on

448 port 8080. It can be started from the app, but it may be started manually at  
449 the command prompt for testing and specialist purposes by executing:

```
450     java -jar coe.jar 8082
```

451 TCP port 8082 will be chosen by default if it is omitted in the command  
452 above. The COE is entirely hidden from the end user of the INTO-CPS app,  
453 but parts of it are transparently configured through the main interface. The  
454 design of the COE is documented in deliverable D4.1d [LLW<sup>+</sup>15].

455 The COE is controlled using simple HTTP requests. These are documented  
456 in the API manual, which can be obtained from the COE's own web page by  
457 navigating to `http://localhost:8082`. Port 8082 should be changed to  
458 that specified when the COE is started.

459 Following the protocol detailed in the API document, a co-simulation session  
460 can be controlled manually from the command prompt using, for example,  
461 the `curl` utility, as demonstrated in the following example.

462 With the COE running, a session must first be created:

```
463     curl http://localhost:8082/createSession
```

464 This command will return a `sessionId` that is used in the following com-  
465 mands.

466 Next, assuming a COE configuration file called `coeconf.json` has been  
467 created as described in the API manual, the session must be initialized:

```
468     curl -H "Content-Type: application/json"  
469     --data @coeconf.json  
470     http://localhost:8082/initialize/sessionID
```

471 Assuming start and end time information has been saved to a file, say  
472 `startend.json`, the co-simulation can now be started:

```
473     curl -H "Content-Type: application/json"  
474     --data @coeconf.json  
475     http://localhost:8082/simulate/sessionID
```

476 Once the co-simulation run ends, the results can be obtained as follows:

```
477     curl -o results.zip  
478     http://localhost:8082/result/sessionID/zip
```

479 The session can now be terminated:

```
480     curl http://localhost:8082/destroy/sessionID
```

481 The app fundamentally controls the COE in this way.

482 **Distributed co-simulations** Presently the app can only control the COE  
483 in this way for non-distributed co-simulations. In order to run a distributed  
484 co-simulation, a distributed version of the COE, `dcoe`, must be controlled  
485 from the command prompt manually, as illustrated above. The distributed  
486 COE can be downloaded using the App's *Download Manager*.

487 In a distributed co-simulation the COE and (some) FMUs execute on physi-  
488 cally different compute nodes. The FMUs local to the COE computing node  
489 are handled in the same way as in standard co-simulations.

490 Each FMU on the remote nodes is served externally by a daemon process.  
491 This process must be started on the remote node manually as follows:

```
492     java -jar daemon*-jar-with-dependencies.jar -host  
493         <public-ip> -ip4
```

494 Here, `<public-ip>` is the IPv4 address of the compute node.

495 Next, the distributed COE process must be started manually from the com-  
496 mand prompt on its own node, with options specific to distributed co-simulation:

```
497     java -Dcoe.fmu.custom.factory=  
498         org.intocps.orchestration.coe.distribution.  
499         DistributedFmuFactory  
500         -jar dcoe*-jar-with-dependencies.jar
```

501 The second difference is the way in which the location of the remote FMUs  
502 is specified. For a standard co-simulation, the "fmus" clause of the co-  
503 simulation configuration file (`coeconfg.json`, in our example) contains el-  
504 ements of the form

```
505     "file://fmu-1-path.fmu"
```

506 These must be modified for each remote FMU to the following URI scheme:

```
507     "uri://<public-ip>/FMU/#file://local-fmu-path.fmu"
```

508 The COE configuration file can, of course, be written manually in its entirety,  
509 but it is possible to take a faster route, as follows.

510 This configuration file is only generated when a co-simulation is executed. It  
511 is therefore possible to assemble a "dummy" co-simulation that is similar to  
512 the desired distributed version, but with a local FMU topology. Since it is  
513 likely that the remote FMUs are not supported on the COE platform itself,

514 it is necessary here to construct “dummy” FMUs with the same interface.  
515 If this local co-simulation is then executed briefly, a COE configuration file  
516 will be emitted that can be easily modified as described above. The app  
517 will name this file `config.json` and emit it to the `Multi-models` folder  
518 under each co-simulation run. This modified configuration can then be used  
519 to execute the distributed co-simulation.

## 520 5 Using the Separate Modelling and Simula- 521 tion Tools

522 This section provides a tutorial introduction to the FMI-specific functionality  
523 of each of the modelling and simulation tools. This functionality is centered  
524 on the role of FMUs for each tool. For more general descriptions of each tool,  
525 please refer to Appendix B.

### 526 5.1 Overture

527 Overture implements export of both tool-wrapper as well as standalone FMUs.  
528 It also has the ability to import a `modelDescription.xml` file in order to  
529 facilitate creating an FMI-compliant model from scratch. A typical workflow  
530 in creating a new FMI-compliant VDM-RT model starts with the import  
531 of a `modelDescription.xml` file created using Modelio. This results in  
532 a minimal project that can be exported as an FMU. The desired model is  
533 then developed in this context. This section discusses the complete work-  
534 flow.

#### 535 5.1.1 Installing the FMI import/export plugin for Overture

536 In order to use the FMI integration in Overture it is necessary to install a  
537 plugin. Below is a guide to install the plugin:

- 538 1. Open Overture.
- 539 2. Select *Help -> Install New Software*.
- 540 3. Click *Add...*
- 541 4. In the *Name:* field write *Overture FMU*.

- 542 5. In the *Location:* field there are two options:
- 543 **INTO-CPS Application:** Download the *Overture FMU Import / Ex-*  
 544 *porter - Overture FMI Support* using the Download Manager men-  
 545 tioned in Section 4.5. Locate the file using the *Archive...* button  
 546 next to the *Location:* field.
- 547 **Update site:** Enter the following URL in the *Location:* field:  
 548 *http://overture.au.dk/into-cps/vdm-tool-wrapper/master/latest.*
- 549 6. Check the box next to *Overture FMI Integration* as shown in Figure  
 550 37.
- 551 7. Click *Next* or *Finish* to accept and install.

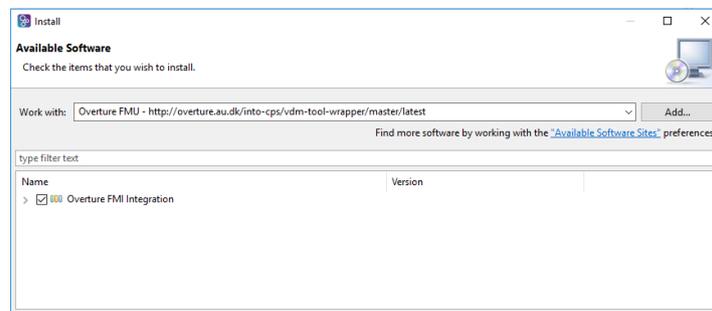


Figure 37: Installing Overture FMI Integration

### 552 5.1.2 Import of `modelDescription.xml` File

553 A `modelDescription.xml` file is easily imported into an existing, typ-  
 554 ically blank, VDM-RT project from the project explorer context menu as  
 555 shown in Figure 38. This results in the project being populated with the  
 556 classes necessary for FMU export:

- 557 • A VDM-RT system class named “System” containing the system def-  
 558 inition. The corresponding “System” class for the water tank controller  
 559 FMU is shown in Listing 39.
- 560 • A standard VDM-RT class named “World”. This class is conventional  
 561 and only provides an entry point into the model. The corresponding  
 562 “World” class for the water tank controller FMU is shown in Listing 40.
- 563 • A standard VDM-RT class named “HardwareInterface”. This class con-  
 564 tains the definition of the input and output ports of the FMU. Its struc-

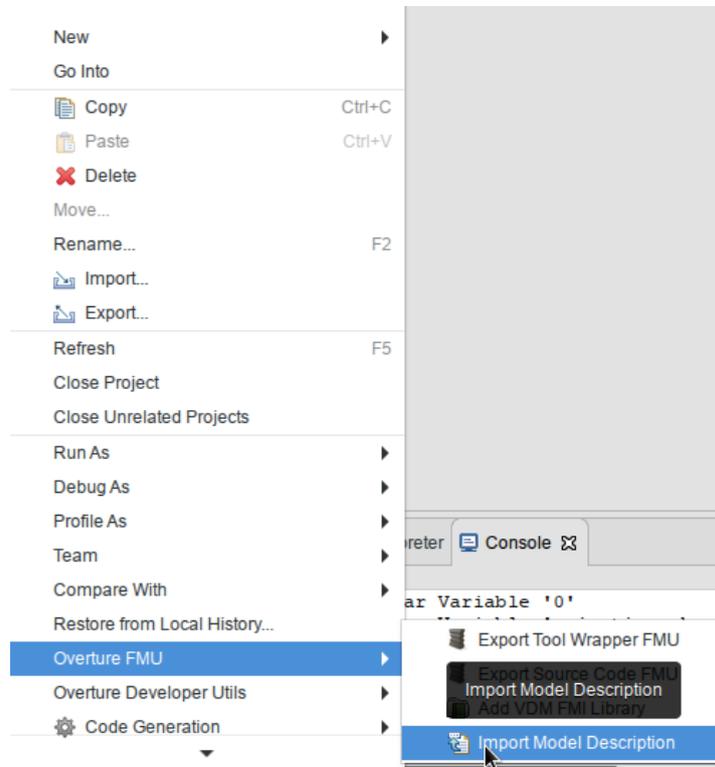


Figure 38: Importing a modelDescription.xml file.

- 565       ture is enforced, and a self-documenting annotation scheme<sup>5</sup> is used  
 566       such that the “HardwareInterface” class may be hand-written. The  
 567       corresponding “HardwareInterface” class for the water tank controller  
 568       FMU is shown in Listing 41.
- 569       • The library file `Fmi.vdmrt` which defines the hardware interface port  
 570       types used in “HardwareInterface”.

<sup>5</sup>The annotation scheme is documented on the INTO-CPS website [into-cps.github.io](https://into-cps.github.io) under “Constituent Model Development → Overture → FMU Import/Export”.

```
system System

instance variables

-- Hardware interface variable required by FMU Import/Export
public static hwi: HardwareInterface := new
    HardwareInterface();

instance variables

    public levelSensor : LevelSensor;
    public valveActuator : ValveActuator;
    public static controller : [Controller] := nil;

    cpu1 : CPU := new CPU(<FP>, 20);
operations

public System : () ==> System
System () ==
(
    levelSensor := new LevelSensor(hwi.level);
    valveActuator := new ValveActuator(hwi.valveState);

    controller := new Controller(levelSensor, valveActuator);

    cpu1.deploy(controller, "Controller");
);

end System
```

Figure 39: “System” class for water tank controller.

```
class World

operations

public run : () ==> ()
run() ==
  (start (System`controller);
   block();
  );

private block : () ==> ()
block() ==
  skip;

sync

  per block => false;

end World
```

Figure 40: “World” class for water tank controller.

```
class HardwareInterface

values
  -- @ interface: type = parameter, name="minlevel";
  public minlevel : RealPort = new RealPort(1.0);
  -- @ interface: type = parameter, name="maxlevel";
  public maxlevel : RealPort = new RealPort(2.0);

instance variables
  -- @ interface: type = input, name="level";
  public level : RealPort := new RealPort(0.0);

instance variables
  -- @ interface: type = output, name="valve";
  public valveState : BoolPort := new BoolPort(false);

end HardwareInterface
```

Figure 41: “HardwareInterface” class for water tank controller.

571 The port structure used in the “HardwareInterface” class is a simple inheri-  
572 tance structure, with a top-level generic “Port”, subclassed by ports for spe-  
573 cific values: booleans, reals, integers and strings. The hierarchy is shown in  
574 Listing 42. When a model is developed without the benefit of an existing  
575 `modelDescription.xml` file, this library file can be added to the project  
576 from the project context menu, also under the category “Overture FMU”.  
577

578 With all the necessary FMU scaffolding in place, the VDM-RT model can be  
579 developed as usual.

### 580 **5.1.3 Tool-Wrapper FMU Export**

581 Models exported as tool-wrapper FMUs require the Overture tool to sim-  
582 ulate. Export is implemented such that the VDM interpreter and its FMI  
583 interface are included in the exported FMU. Overture tool-wrapper FMUs  
584 currently support Win32, Win64, Linux64, Darwin64 and require Java 1.7  
585 to be installed and available in the PATH environment variable.

586 A tool-wrapper FMU is easily exported from the project context menu as  
587 shown in Figure 43. The FMU will be placed in the generated folder.  
588

```
class Port

types
  public String = seq of char;
  public FmiPortType = bool | real | int | String;

operations

  public setValue : FmiPortType ==> ()
  setValue(v) == is subclass responsibility;

  public getValue : () ==> FmiPortType
  getValue() == is subclass responsibility;

end Port

class IntPort is subclass of Port

instance variables
  value: int:=0;

operations
  public IntPort: int ==> IntPort
  IntPort(v)==setValue(v);

  public setValue : int ==> ()
  setValue(v) ==value :=v;

  public getValue : () ==> int
  getValue() == return value;

end IntPort

class BoolPort is subclass of Port

instance variables
  ...
```

Figure 42: Excerpt of “Fmi.vdmrt” library file defining FMI interface port hierarchy.

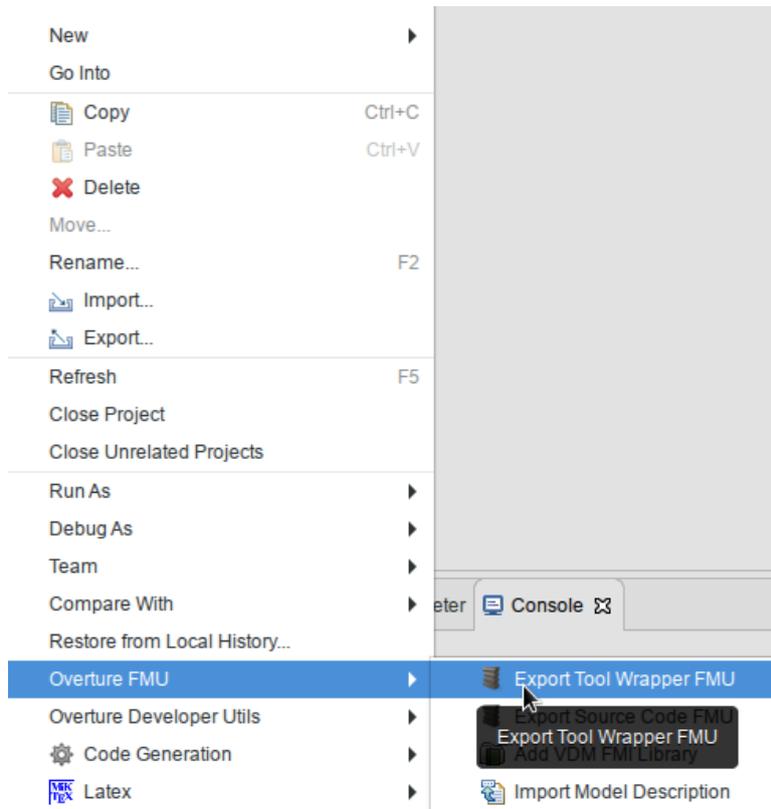


Figure 43: Exporting a tool-wrapper FMU.

#### 589 5.1.4 Standalone FMU Export

590 In contrast to tool-wrapper FMUs, models exported as standalone FMUs  
591 do not require Overture in order to simulate. Instead, they are first passed  
592 through Overture's C code generator such that a standalone implementation  
593 of the model is first obtained. Once compiled, this executable model then  
594 replaces the combination of VDM interpreter and model, and the FMU ex-  
595 ecutes natively on the co-simulation platform. Currently Mac OS, Windows  
596 and Linux are supported, with embedded platform support for SiL and HiL  
597 simulation under development.

598 The export process consists of two steps. First, a source code FMU is ob-  
599 tained from Overture as shown in Figure 44. Second, the INTO-CPS Appli-  
600 cation must be used to upload the resulting FMU to the FMU compilation  
601 server using the built-in facility described in Section 4.5. This is accessed by  
602 navigating to *Window* → *Show FMU Builder*.

603 Please note that only some features of VDM-RT are currently supported by  
604 the C code generator. This is discussed in more detail in Section 9.

## 605 5.2 20-sim

606 This section explains the FMI and INTO-CPS related features of 20-sim<sup>6</sup>.  
607 We focus on the import of `modelDescription.xml` files, standalone and  
608 tool-wrapper FMU export (FMU slave), 3D visualization of FMU operation  
609 and an experimental FMU import (FMU master) feature. The complete  
610 20-sim tool documentation can be found in the 20-sim Reference Manual  
611 [KGD16].

### 612 5.2.1 Import of `modelDescription.xml` File

613 In Modelio it is possible to export the desired interface for a new FMU  
614 from a multi-model as a `modelDescription.xml` file (see Section 3.2.  
615 20-sim can automatically generate an empty 20-sim submodel<sup>7</sup> from this  
616 `modelDescription.xml` file with this desired FMU interface. To use

---

<sup>6</sup>Note that 20-sim is Windows-only. However, it can run fine using Wine [Win16] on other platforms. For details on using 20-sim under Wine, contact Controllab.

<sup>7</sup>Please note that the term “submodel” here should not be confused with the INTO-CPS notion of a “constituent model”. A submodel here is a part in a graphical 20-sim model.

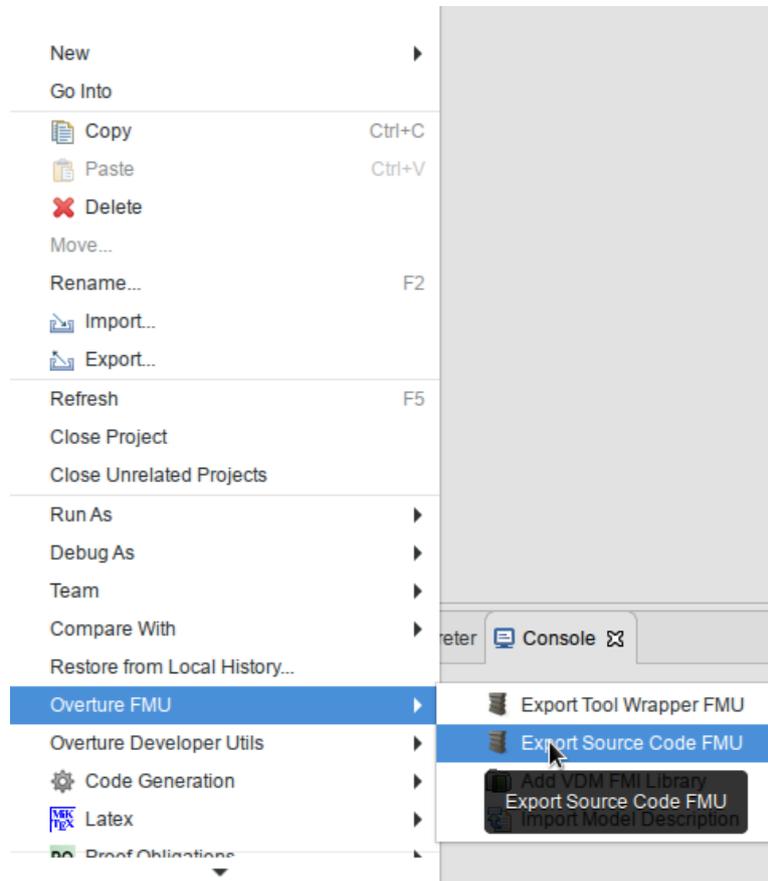


Figure 44: Exporting a standalone FMU.

617 the `modelDescription.xml` import, you will need to use the “4.6.2-  
 618 `intocps`” version of 20-sim<sup>8</sup>, since this feature is still under development. A  
 619 `modelDescription.xml` file can be imported into 20-sim by using Win-  
 620 dows Explorer to drag the `modelDescription.xml` file onto your 20-sim  
 621 model (see Figure 45). This creates a new empty submodel with a blue icon  
 622 that has the same inputs and outputs as defined in the `modelDescription`  
 623 `.xml` file.

### 624 5.2.2 Tool-wrapper FMU Export

625 A tool-wrapper FMU is a communication FMU that opens the original model  
 626 in the modelling tool and takes care of remotely executing the co-simulation

<sup>8</sup>You can download the INTO-CPS version of 20-sim using the Download Manager in the INTO-CPS Application.

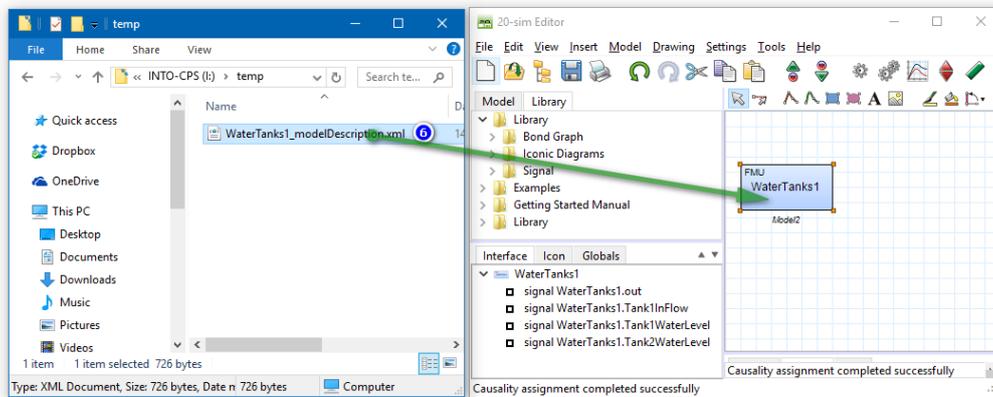


Figure 45: Import a ModelDescription in 20-sim.

627 steps inside the modelling using some tool-supported communication mecha-  
 628 nism. 20-sim supports co-simulation using the XML-RPC-based DESTECs  
 629 co-simulation interface [LRVG11]. The generation of a tool-wrapper FMU  
 630 involves two steps that will be explained below:

- 631 1. Extend the model with co-simulation inputs, outputs and shared design  
 632 parameters.
- 633 2. Generate a model-specific tool-wrapper FMU.

634 The tool-wrapper approach involves communication between the co-simula-  
 635 tion engine (COE) and the 20-sim model through the tool-wrapper FMU.  
 636 The 20-sim model should be extended with certain variables that can be  
 637 set or read by the COE. These variables are the co-simulation inputs and  
 638 outputs. They can be defined in the model in an equation section called  
 639 externals:

```
640 externals
641     real global export mycosimOutput;
642     real global import mycosimInput;
```

645 To make it possible to set or read a parameter by the co-simulation engine,  
 646 it should be marked as 'shared':

```
647 parameters
648     // shared design parameters
649     real mycosimParameter ('shared') = 1.0;
```

652 The next step is to generate a tool-wrapper FMU for the prepared model.

653 This requires at least the “4.6.3-intocps” version of 20-sim<sup>9</sup>. This version of  
654 20-sim comes with a Python script that generates a tool-wrapper FMU for  
655 the loaded model.

656 To generate the tool-wrapper FMU:

- 657 1. Make sure that the tool-wrapper prepared 20-sim model is saved at  
658 a writable location. The tool-wrapper FMU will be generated in the  
659 same folder as the model.
- 660 2. Open the prepared 20-sim model in 20-sim.
- 661 3. Run the BATCH script:  
662 *C:\Program Files (x86)\20-sim 4.6\addons\FMI\  
663 ToolwrapperFMUexport\generate.bat*  
664 Note that the (x86) is only for 64-bit versions of Windows.
- 665 4. You can find the generated tool-wrapper fmu as *<modelname>.fmu* in  
666 the same folder as your model.

### 667 5.2.3 Standalone FMU Export

668 Starting with 20-sim version 4.6, the tool has a built-in option to generate  
669 standalone co-simulation FMUs for both FMI 1.0 and 2.0 (note that version  
670 2.0 must be used here).

671 To export a 20-sim submodel as a standalone FMU, make sure that the part  
672 of the model that you want to export as an FMU is contained in a submodel  
673 and simulate your model to confirm that it behaves as desired.

674 Next, follow these steps (see also Figure 46):

- 675 1. In the Simulator window, choose from the menu: *Tools*.
- 676 2. Select *Real Time Toolbox*.
- 677 3. Click *C-Code Generation*.
- 678 4. Select the *FMU 2.0 export for 20-sim submodel* target.
- 679 5. Select the submodel to export as an FMU.
- 680 6. Click OK to generate the FMU. This will pop-up a blue window.

---

<sup>9</sup>You can download the INTO-CPS version of 20-sim using the Download Manager in the INTO-CPS Application.

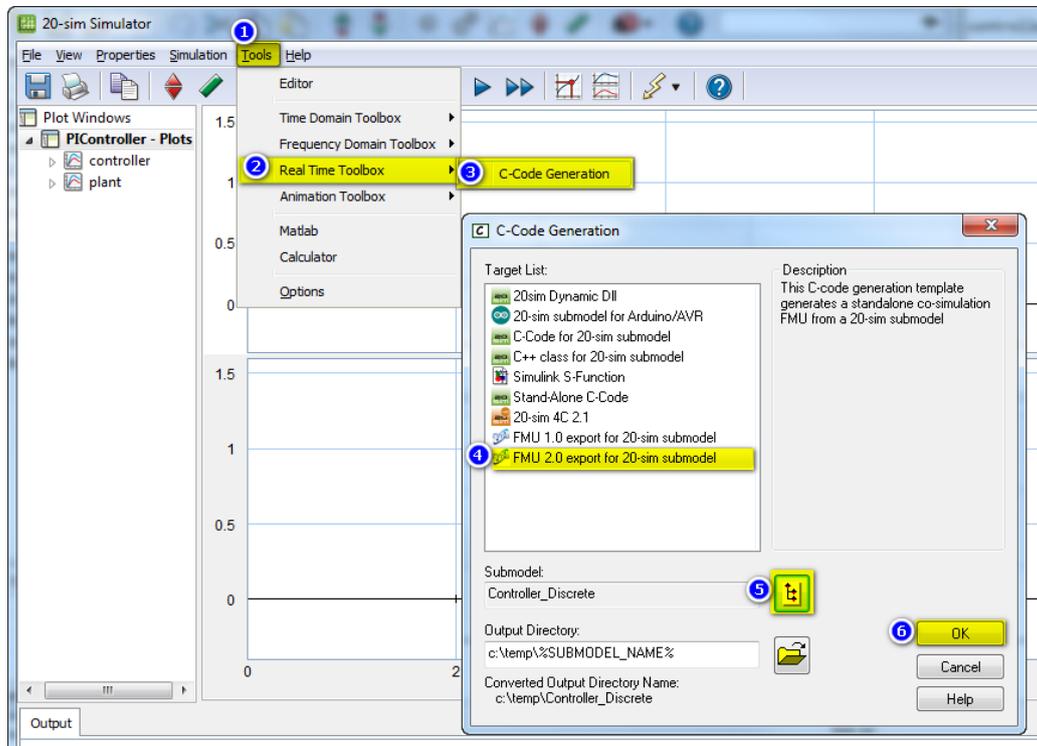


Figure 46: Export an FMU from 20-sim.

681 Note that to automatically compile the FMU, you will need the Microsoft  
 682 Visual C++ 2010, 2013 or 2015 compiler installed (normally included with  
 683 Microsoft Visual Studio, either Express or Community edition). If 20-sim  
 684 can find one of the supported VC++ compilers, it starts the compilation  
 685 and reports where you can find the newly generated FMU. The 20-sim FMU  
 686 export also generates a *Makefile* that allows you to compile the FMU on  
 687 Windows using Cygwin, MinGW, MinGW64 or on Linux or MacOS X.  
 688 20-sim can currently export only a subset of the supported modelling lan-  
 689 guage elements as standalone C-code. Full support for all 20-sim features is  
 690 only possible through the tool-wrapper FMU approach (described shortly in  
 691 Section 5.2.2). The original goal for the 20-sim code generator was to export  
 692 control systems into ANSI-C code to run the control system under a real-  
 693 time operating system. As a consequence, 20-sim currently only allows code  
 694 generation for discrete-time submodels or continuous-time submodels using  
 695 a fixed-step integration method. Support for variable step size integration  
 696 methods is not yet included by default in the official 20-sim 4.6 release, but it  
 697 is already included in the 20-sim “4.6.2-intocps” release and on GitHub (see  
 698 below). Other language features that are not supported, (or are only partly

699 supported) for code generation, are:

- 700 • **Hybrid models:** Models that contain both discrete- and continuous-  
701 time sections cannot be generated at once. However, it is possible to  
702 export the continuous and discrete blocks separate.
- 703 • **File I/O:** The 20-sim “Table2D” block is supported; the “datafromfile”  
704 block is not yet supported.
- 705 • **External code:** Calls to external code are not supported. Examples  
706 are: `DLL()`, `DLLDynamic()` and the MATLAB functions.
- 707 • **Variable delays:** The `tdelay()` function is not supported due to  
708 the requirement for dynamic memory allocation.
- 709 • **Event functions:** `timeevent()`, `frequencyevent()` statements  
710 are ignored in the generated code.
- 711 • **Fixed-step integration methods:** *Euler*, *Runge-Kutta 2* and *Runge-*  
712 *Kutta 4* are supported.
- 713 • **Implicit models:** Models that contain unsolved algebraic loops are  
714 not supported.
- 715 • **Variable-step integration methods:** *Vode-Adams* and *Modified Back-*  
716 *ward Differential Formula* (MeBDF) are available on GitHub (see below  
717 for the link).

718 The FMU export feature of 20-sim is being improved continuously based on  
719 feedback from INTO-CPS members and other customers. To benefit from  
720 bug fixes and to try the latest FMU export features like variable step size  
721 integration methods (*e.g.* Vode-Adams and MeBDF), you can download the  
722 latest version of the 20-sim FMU export template from:

723 `https://github.com/controllab/fmi-export-20sim`

724 Detailed instructions for the installation of the GitHub version of the 20-sim  
725 FMU export template can be found on this GitHub page. The GitHub FMU  
726 export template can be installed alongside the existing built-in FMU export  
727 template.

#### 728 5.2.4 3D Animation FMU

729 It is possible to visualize a 20-sim simulation as a live 3D animation. This 20-  
730 sim 3D animation can be exported as a 3D animation FMU that can be used

731 for visualization purposes in a FMI co-simulation experiment. An example  
732 of a 3D animation FMU in action is shown in Figure 47.

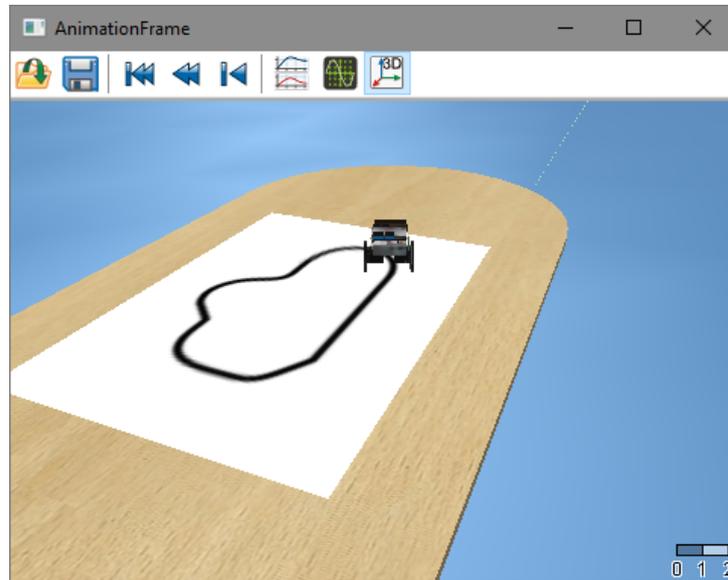


Figure 47: 3D animation FMU

733 To create a 3D animation FMU, you will need to create a 3D animation in  
734 20-sim that reacts to some signals first (identical to the creation of standard  
735 3D animation in 20-sim):

- 736 1. Open your 20-sim model.
- 737 2. Open the simulator and add a new 3D animation window using *View*  
738 *→ New 3D animation window*.
- 739 3. Create a new 3D animation scene by following the instructions from  
740 the Animation toolbox section in the 20-sim Getting Started manual  
741 [KG16].
- 742 4. For elements that should move or change color based on external sig-  
743 nals, create one equation submodel in 20-sim with all required input  
744 signals for the animation.
- 745 5. Connect the 3D animation object to the signals from this animation  
746 submodel.

747 The next step is to export the 3D animation as standalone scenery:

- 748 1. Go to the 3D animation plot in your 20-sim model.

- 749 2. Right-click in the 3D animation plot and select *Plot properties*.
- 750 3. Choose *File* → *Save scene*.
- 751 4. Select *Yes* to save the whole scenery.
- 752 5. Save the scenery under the name `scenery.scn`.

753 The 3D animation FMU uses the just exported `scenery.scn` file. Since  
754 the 3D animation is only a view of the simulation results, the FMU only has  
755 a list of inputs. To generate a `modelDescription.xml` file with the right  
756 FMU interface, a Python script must be executed which collects the list of  
757 external signals referred to by the exported scenery. This Python script and  
758 other required resources can be found in the following Controllab GitHub  
759 repository:

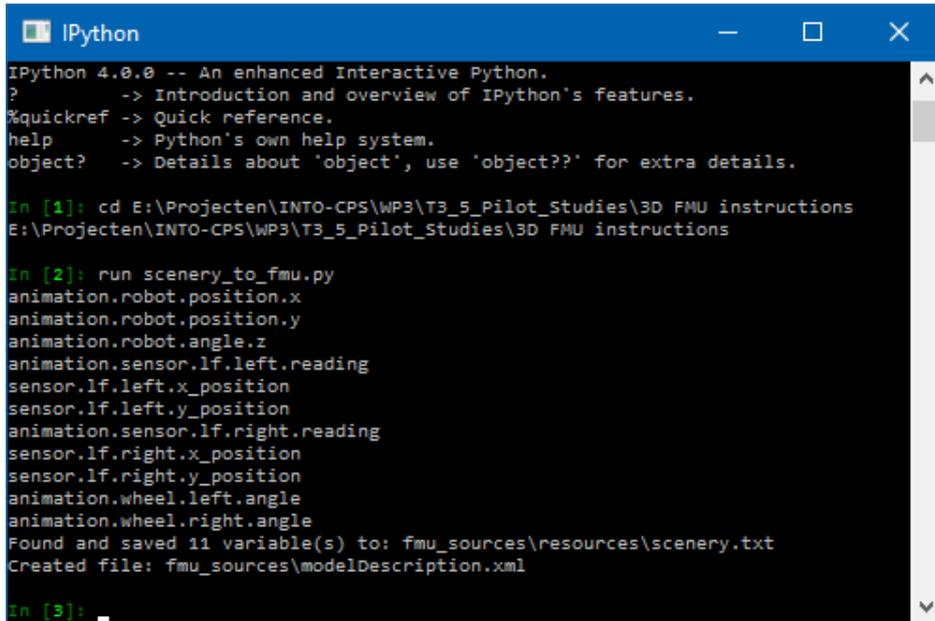
760 `https://github.com/controllab/fmi-3D-animation`

761 To generate the FMU `modelDescription.xml` file, do the following:

- 762 1. Copy the generated `scenery.scn` in the `fmu_sources\resources`  
763 folder under *3D FMU instructions*.
- 764 2. Update `FMU_GUID` in the `scenery_to_fmu.py` Python script with  
765 a new GUID for your 3D Animation FMU.
- 766 3. Execute the `scenery_to_fmu.py` Python script, *e.g.* using the Python  
767 installation that comes with 20-sim 4.6:
  - 768 • Start *IPython* found under *20-sim 4.6* in the Windows Start Menu.
  - 769 • `cd <my 3D FMU instructions path>`
  - 770 • `run scenery_to_fmu.py`  
771 This parses the `scenery.scn` file for objects that point to vari-  
772 ables/parameters (references). The variables/parameters are trans-  
773 lated to FMU inputs and FMU parameters. The 3D scenery does  
774 not contain any information that indicates whether the referred  
775 name is a variable or a parameter. As a workaround, all names  
776 that start with *parameter.* are marked as as FMU parameters  
777 (`causality = parameter`), while all others are generated as inputs  
778 (`variability = continuous`). This script also generates a `scenery.`  
779 `txt` file with the list of found references. This file is read by the  
780 3D animation DLL to couple the FMU interface to the 3D scenery  
781 objects. The output resembles that shown in Figure 48.

- 782 4. Create the actual FMU:

- 783
- Copy all needed textures to the `fmu_sources\resources` folder.
- 784
- Zip the `fmu_sources` folder.
- 785
- Rename the Zip file, *e.g.* `3DAnimationFMU.fmu`.



```
IPython 4.0.0 -- An enhanced Interactive Python.
?      -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help    -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.

In [1]: cd E:\Projecten\INTO-CPS\WP3\T3_5_Pilot_Studies\3D FMU instructions
E:\Projecten\INTO-CPS\WP3\T3_5_Pilot_Studies\3D FMU instructions

In [2]: run scenery_to_fmu.py
animation.robot.position.x
animation.robot.position.y
animation.robot.angle.z
animation.sensor.lf.left.reading
sensor.lf.left.x_position
sensor.lf.left.y_position
animation.sensor.lf.right.reading
sensor.lf.right.x_position
sensor.lf.right.y_position
animation.wheel.left.angle
animation.wheel.right.angle
Found and saved 11 variable(s) to: fmu_sources\resources\scenery.txt
Created file: fmu_sources\modelDescription.xml

In [3]:
```

Figure 48: Generating `modelDescription.txt` file from 3D scenery.

### 786 5.2.5 FMI 2.0 Import

787 The “4.6.2-intocps” version of 20-sim has an experimental option to import  
788 an FMU directly in 20-sim for co-simulation within 20-sim itself. This is  
789 useful for quickly testing exported FMUs without the need to set-up a full  
790 co-simulation experiment in the app. Presently only FMI 2.0 co-simulation  
791 FMUs can be imported.

792 The procedure for importing an FMU as 20-sim submodel is similar to im-  
793 porting a `modelDescription.xml` file. Follow these steps to import an  
794 FMU in 20-sim:

- 795 1. Copy/move the FMU to the same folder as your model. This is not  
796 required but recommended to prevent embedding hardcoded paths in  
797 your model.
- 798 2. Using Windows Explorer, drag the FMU file on your 20-sim model (see  
799 Figure 49).

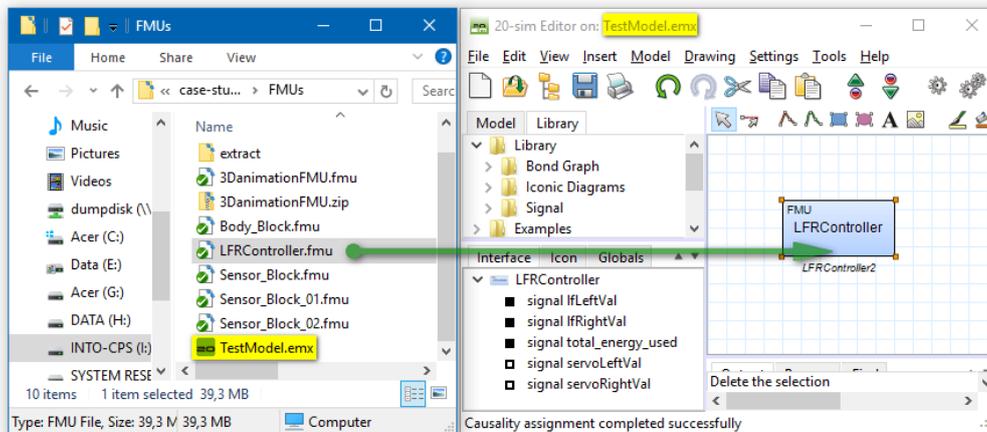


Figure 49: Importing an FMU in 20-sim.

800 This creates a new submodel with a blue icon that acts as an FMU wrap-  
 801 per. FMU inputs and outputs are translated into 20-sim submodel input  
 802 and output signals. FMU parameters (scalar variables with causality “pa-  
 803 rameter”) are also available in 20-sim. This means that you can alter the  
 804 default values of these FMU parameters in 20-sim. The altered FMU param-  
 805 eters are transferred to the FMU during the initialization mode phase of the  
 806 FMU.

### 807 5.3 OpenModelica

808 This section explains the FMI and INTO-CPS related features of Open-  
 809 Modelica. The focus is on import of `modelDescription.xml` files, and  
 810 standalone and tool-wrapper FMU export.

#### 811 5.3.1 Import of `modelDescription.xml` File

812 OpenModelica can import `modelDescription.xml` interface files creat-  
 813 ed using Modelio and create Modelica models from them. To use the  
 814 `modelDescription.xml` import feature, you will need to use OpenMod-  
 815 elica nightly-builds versions, as this extension is rather new. Nightly builds  
 816 can be obtained through the main INTO-CPS GitHub site:

817 <http://into-cps.github.io>

818 To import a `modelDescription.xml` file in OpenModelica one can use:

819 1. The OpenModelica Connection Editor GUI (OMEdit): *FMI* → *Import*  
820 *FMI Model Description*.

821 2. A MOS script, *i.e.* `script.mos`, see below.

```
822 // start script.mos
823 // import the FMU modelDescription.xml
824 importFMUModelDescription("path/to/modelDescription.xml");
825   getErrorString();
826 // end script.mos
827
828
```

829 The MOS script can be executed from command line via:

```
830 // on Linux and Mac OS
831 > path/to/omc script.mos
832 // on Windows
833 > %OPENMODELICAHOME%\bin\omc script.mos
834
835
```

836 The result is a generated file with a Modelica model containing the inputs  
837 and outputs specified in `modelDescription.xml`. For instance:

```
838 model Modelica_Blocks_Math_Gain_cs_FMU "Output the product
839   of a gain value with the input signal"
840   Modelica.Blocks.Interfaces.RealInput u "Input signal
841     connector" annotation (Placement (transformation (extent
842       ={{-120, 60}, {-100, 80}})));
843   Modelica.Blocks.Interfaces.RealOutput y "Output signal
844     connector" annotation (Placement (transformation (extent
845       ={{100, 60}, {120, 80}})));
846 end Modelica_Blocks_Math_Gain_cs_FMU; "
847
848
```

849 This functionality will ultimately be integrated in the OMEdit (the Open-  
850 Modelica Connection Editor) graphical user interface.

### 851 5.3.2 FMU Export

852 Currently all FMUs exported from OpenModelica are standalone. There are  
853 two ways to export an FMU:

- 854 1. From a command prompt.
- 855 2. From OMEdit (OpenModelica Connection Editor).

856 **FMU export from a command prompt** To export an FMU for co-  
857 simulation from a Modelica model a Modelica script file `generateFMU.mos`  
858 containing the following calls to the OMC compiler can be used:

```
859 // load Modelica library  
860 loadModel(Modelica); getErrorString();  
861  
862 // load other libraries if needed  
863 // loadModel(OtherLibrary); getErrorString();  
864  
865 // generate the FMU: PathTo.MyModel.fmu  
866 translateModelFMU(PathTo.MyModel, "2.0", "cs");  
867 getErrorString();  
868
```

870 Next, the OMC compiler must be invoked on the `generateFMU.mos` script:

```
871 // on Linux and Mac OS  
872 > path/to/omc generateFMU.mos  
873 // on Windows  
874 > %OPENMODELICAHOME%\bin\omc generateFMU.mos  
875  
876
```

877 **FMU export from OMEdit** One can also use OMEdit (the OpenMod-  
878 elica Connection Editor) to export an FMU as detailed in the figures be-  
879 low.

- 880 • Open OMEdit (see Figure 50).
- 881 • Load the model in OMEdit (see Figure 51).
- 882 • Open the model in OMEdit (see Figure 52).
- 883 • Use the menu to export the FMU (see Figure 53).
- 884 • The FMU is now generated (see Figure 54).

885 The generated FMU will be saved to `%TEMP%\OpenModelica\OMEdit.`

886

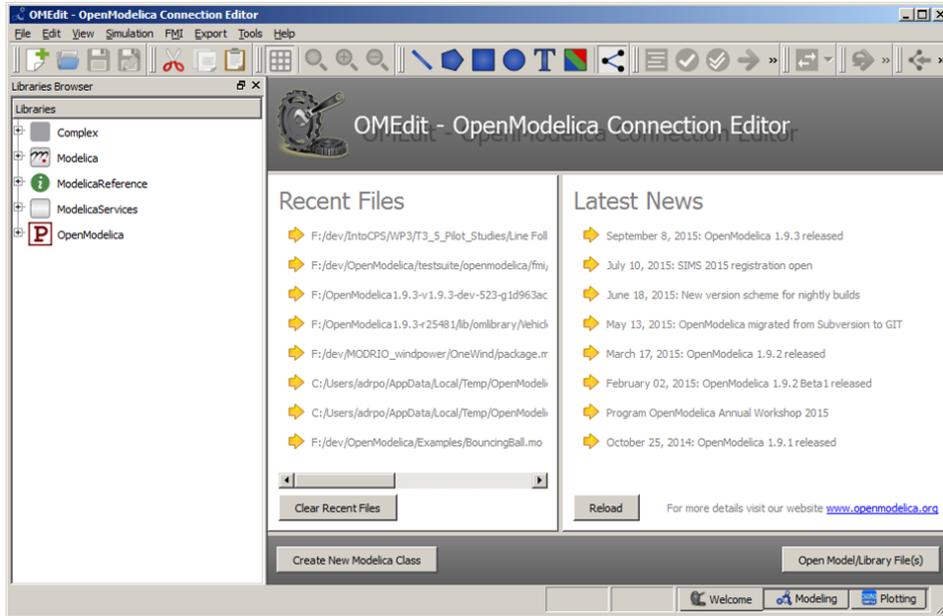


Figure 50: Opening OMEdit.

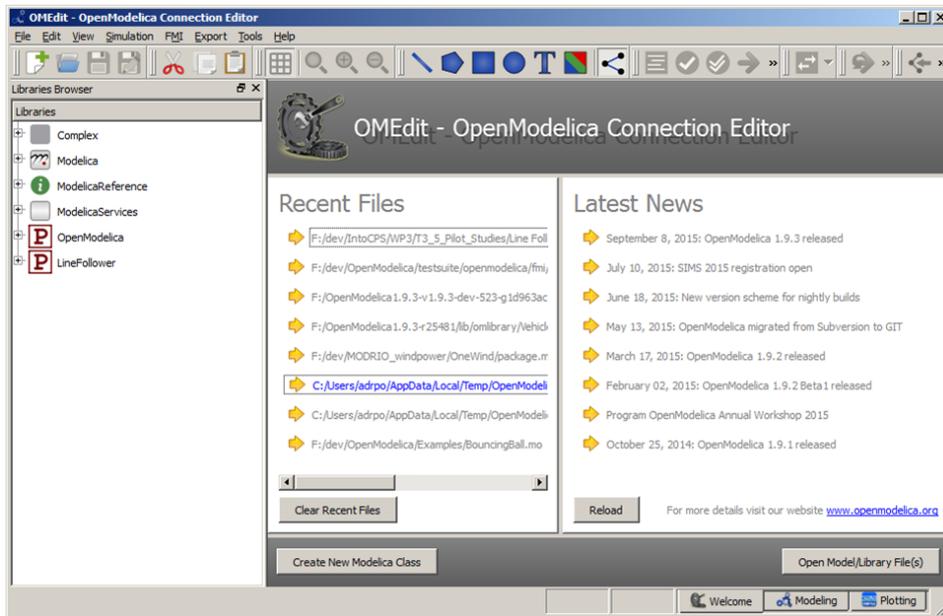


Figure 51: Loading the Modelica model in OMEdit.

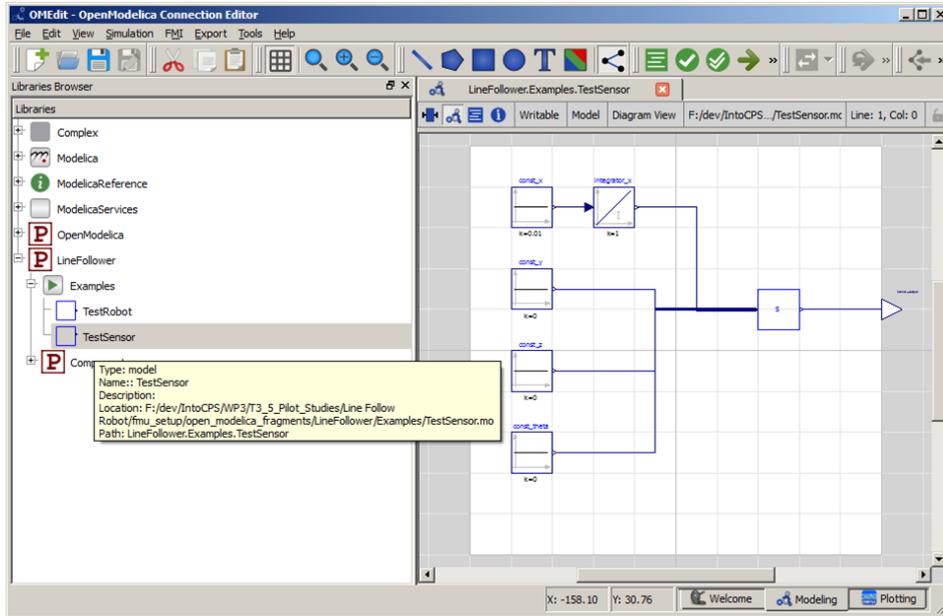


Figure 52: Opening the Modelica model in OMEdit.

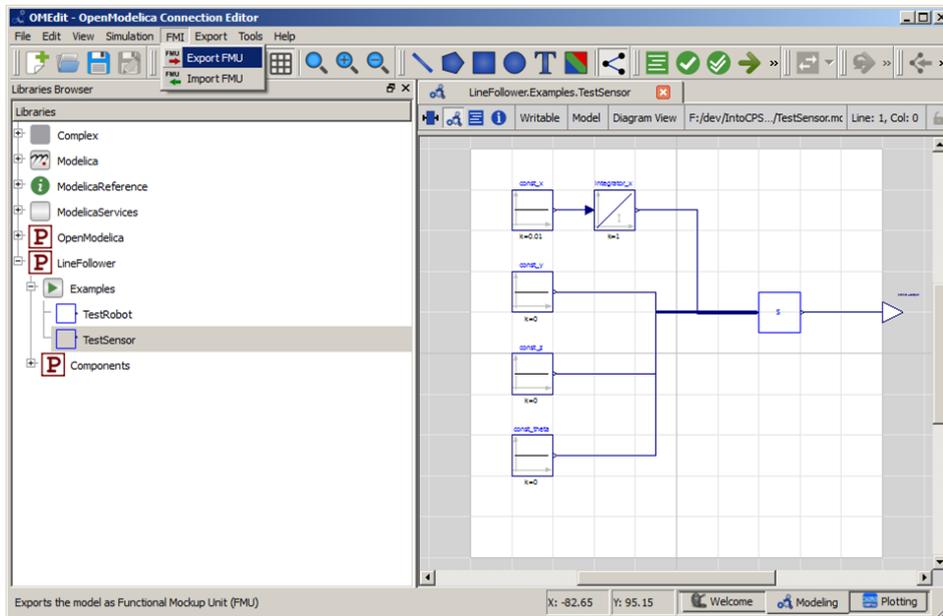


Figure 53: Exporting the FMU.

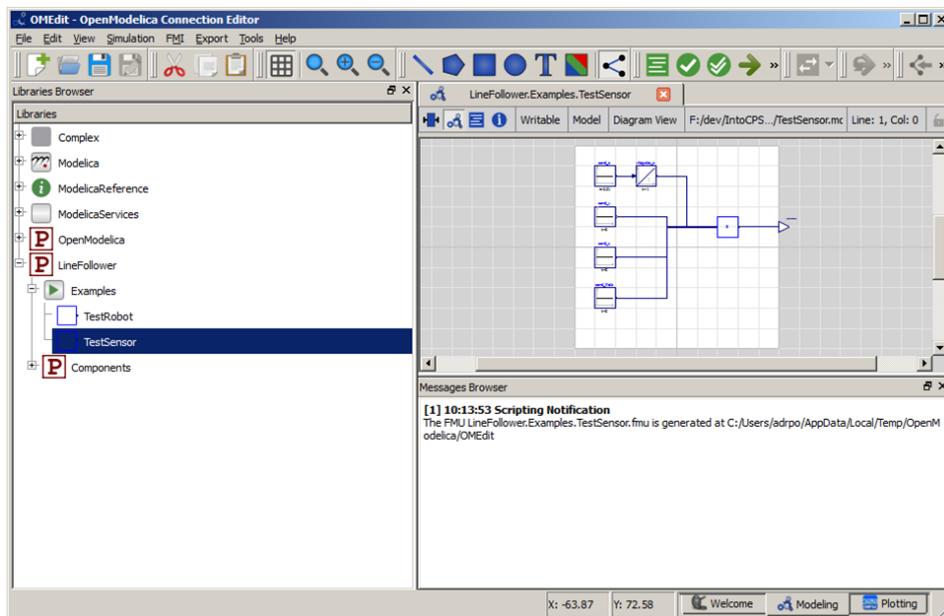


Figure 54: Final step of FMU export.

## 887 6 Design Space Exploration for INTO-CPS

888 This section provides a description of tool support for design space explo-  
889 ration (DSE) developed as part of the INTO-CPS project. Presently the  
890 INTO-CPS Application does not provide support for automated creation of  
891 the configuration files required to define a DSE experiment. Therefore, this  
892 section is split into three parts. Section 6.1 describes how the INTO-CPS  
893 Application can be used to launch a DSE using an existing configuration  
894 file and Section 6.2 describes how the results from DSE are generated and  
895 stored. Section 6.3 describes the structure of the DSE configuration file, giv-  
896 ing enough detail for the user to be able to edit one for their purposes.

### 897 6.1 How to Launch a DSE

898 To launch a DSE we need to provide the INTO-CPS Application with the  
899 path to two files. The first is the DSE configuration, defining the parameters  
900 of the design space, how it should be searched, measured and the results com-  
901 pared. The second is the multi-model configuration, defining the base model  
902 that will be used for the search. A DSE configuration is selected by double  
903 clicking on one of the configurations listed in the *Design Space Explorations*  
904 section of the INTO-CPS Application project explorer; these configurations  
905 are identified with the  icon. If the COE is not already running, the  
906 DSE page is shown with a red “*Co-simulation engine not running*” status,  
907 as shown in Figure 55.

908 If this is the case, click on the *Launch* button to start the COE. This re-  
909 sults in a green co-simulation engine status (see Figure 56). With the DSE  
910 configuration selected and the COE running, the next step is to select the  
911 multi-model to use. One can be selected from the *Co-simulation Configura-*  
912 *tion* drop-down box, as shown in Figure 57. Pressing the *Simulate* button  
913 starts the DSE background process.



Figure 55: Status when COE is not running.



Figure 56: Status when COE is running.



Figure 57: Selecting a multi-model.



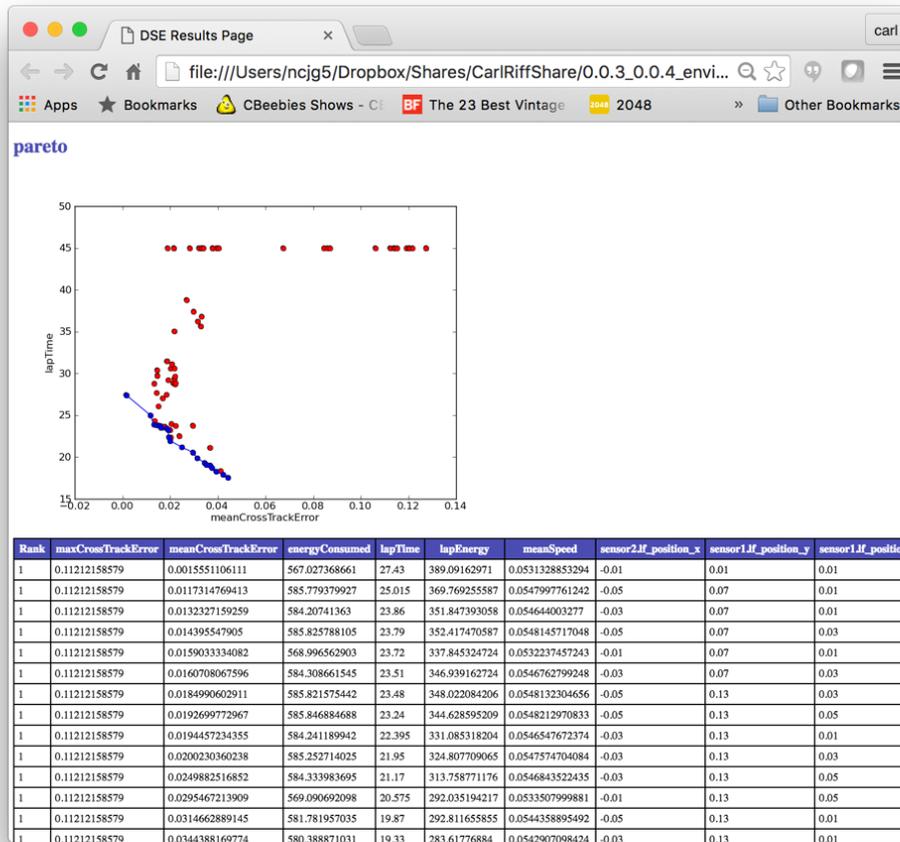


Figure 59: A page of DSE results.

933 **6.3.1 File Creation**

934 The suggested procedure for creating a new configuration is to make a copy  
 935 of an existing one and then to edit the required sections. The individual  
 936 configurations are located in their own folders within the Design Space  
 937 Exploration folder of the INTO-CPS Application project directory, such  
 938 as the pilot study with the line following robot “LFR-2SensorPositions” con-  
 939 figuration shown in Figure 60 (see [PGP+16]). Using your OS’s file browser,  
 940 create a new folder under DSEs and then copy in and rename a DSE configu-  
 941 ration. The names of the new folder and configuration folder can be chosen at  
 942 will, but the configuration file must have the extension `.dse.json`.

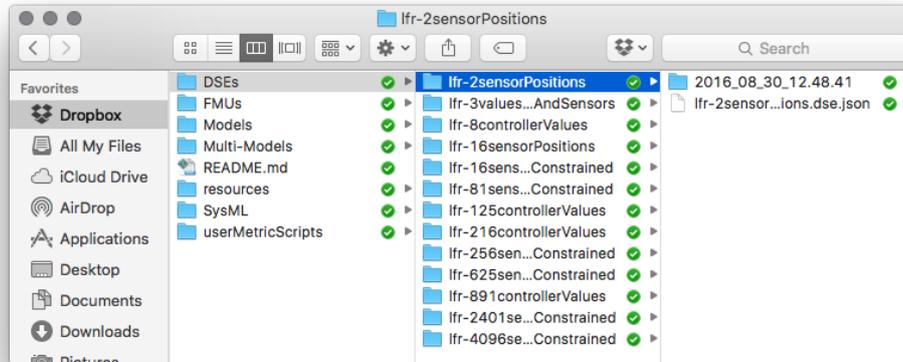


Figure 60: Location of DSE configurations.

### 943 6.3.2 Parameters

944 The parameters section is used to define a list of values for each parameter  
 945 to be explored. Figure 61 shows the definition of four parameters, each with  
 946 two values. If a parameter is included in the DSE configuration file, then it  
 947 must have at least one value defined. The order of the values in the list is  
 948 not important. If a parameter that is to be explored is not in the list, its ID  
 949 may be found in the three ways listed below.

- 950 1. If the parameter is listed in the multi-model configuration, then copy  
 951 it from there.
- 952 2. If the parameter is not in the multi-model parameters list then its name  
 953 may be found by examining the model description file in the associated  
 954 FMU. In this case it will be necessary to prepend the parameter ID  
 955 with the ID for the FMU and the instance ID of the FMU, for example  
 956 in “{sensor1FMU}.sensor1.lf\_position\_x”.
  - 957 • the ID of the FMU is {sensor1FMU}.
  - 958 • the instance ID of the FMU in the multi-model is sensor1.
  - 959 • the parameter ID is lf\_position\_x.
- 960 3. The IDs for each parameter may also be found on the Architecture  
 961 Structure Diagram in the SysML models of the system. The full name  
 962 for use in the multi-model may then be constructed as above.

```
"parameters": {
  "{sensor1FMU}.sensor1.lf_position_x": [
    0.01,
    0.03
  ],
  "{sensor1FMU}.sensor1.lf_position_y": [
    0.07,
    0.13
  ],
  "{sensor2FMU}.sensor2.lf_position_x": [
    -0.01,
    -0.03
  ],
  "{sensor2FMU}.sensor2.lf_position_y": [
    0.07,
    0.13
  ]
},
```

Figure 61: Example parameter definitions.

### 963 6.3.3 Parameter Constraints

964 It may be the case that not all combinations of the parameter values defined  
965 in the previous section are valid. So, it is necessary to be able to define  
966 constraints over the design parameters such that no time is wasted simulating  
967 invalid designs. For example, in the line follower robot we define ranges for  
968 the x and y co-ordinates of the left and right sensors separately, and running  
969 all combinations of these leads to asymmetric designs that do not have the  
970 same turning behaviour on left and right turns. To prevent this we can define  
971 boolean expressions based upon the design parameters and evaluate these  
972 before a simulation is launched. Figure 62 shows two constraints defined for  
973 the line follower DSE experiment that ensure only symmetrical designs are  
974 allowed. The first constraint ensures the y co-ordinates of both sensors are  
975 the same, while the second constraint ensures that the x co-ordinate of the  
976 left sensor is the same, but negated as the x co-ordinate of the right sensor.  
977 Note that the names used when defining such constraints have the same  
978 `FMU_ID.instance_ID.parameter_ID` format as used when defining a  
979 parameter range (see Section 6.3.2)

980 Since the constraints are processed using the Python `eval` function, any  
boolean expression compatible with it may be used here.

```
"parameterConstraints": [
  "{sensor1FMU}.sensor1.lf_position_y == {sensor2FMU}.sensor2.lf_position_y",
  "{sensor1FMU}.sensor1.lf_position_x == - {sensor2FMU}.sensor2.lf_position_x"
],
```

Figure 62: Example parameter constraints.

981

### 982 6.3.4 Scenario List

983 The DSE scripts currently have limited support for scenarios referring to a  
984 specific set of conditions against which the multi-model is to be tested. In  
985 the example of the line following robot, the scenario refers to the map the  
986 robot has to follow, along with its starting co-ordinates. For instance, in  
987 one scenario the robot would go around a circular track in one direction,  
988 predominantly turning left, whereas in a different scenario the same track  
989 would be followed in the opposite direction, predominantly turning right. In  
990 both scenarios the map of the track is the same.

991 Changing a scenario may involve changing one or more different parts of  
992 the multi-model and its analysis, such as the specific FMUs used, paramet-  
993 ers passed to an FMU, the multi-model the DSE is based upon, along with  
994 any data files used by the objective scripts (Section 6.3.6) to evaluate perfor-  
995 mance. This feature is currently under development and so only the objective  
996 data file selection is implemented presently.

### 997 6.3.5 Objective Definitions: Internal

998 There are two means for defining the objectives used to assess the perfor-  
999 mance of a simulated model. The first of these, described here, is using the  
1000 internal functions included in the DSE scripts. This is a set of simple func-  
1001 tions that can be applied to any of the values recorded by the COE during  
1002 simulation. The current set of internal functions is:

1003 **max** Returns the maximum value of a variable during a simulation.

1004 **min** Returns the minimum value of a variable during a simulation.

1005 **mean** Returns the mean value of a variable during a simulation (*n.b.*, a fixed  
1006 simulation step size is currently assumed.)

1007 Defining an internal objective requires three pieces of information:

1008 **name** This is the name that the objective value will be stored under in the  
1009 objectives file.

1010 **type** This selects the function to be applied. The key `objectiveType` is  
1011 used in the DSE configuration file.

1012 **variable** This defines the variable to which the function is to be applied.  
1013 The key `columnID` is used to denote this parameter in the DSE con-  
1014 figuration file.

```
"energyConsumed": {  
  "columnID": "{bodyFMU}.body.total_energy_used",  
  "objectiveType": "max"  
}
```

Figure 63: Definition of an internal objective.

1015 Figure 63 shows the definition of an objective named `energyConsumed`,  
1016 which records the maximum value of the variable  
1017 `{bodyFMU}.body.total_energy_used`. This objective is recorded and  
1018 may be used later, primarily for the purpose of ranking designs, but it could  
1019 also be used for any other analysis required.

### 1020 6.3.6 Objective Definitions: External Scripts

1021 The second form of objective definition makes use of user-defined Python  
1022 scripts to allow bespoke analysis of simulation results to be launched auto-  
1023 matically and results recorded using the common format. The definition has  
1024 two parts: the construction of the Python script to perform the analysis and  
1025 the definition of the script's required parameters in the DSE configuration  
1026 file, these two steps are described below.

1027 **Construction of the Script** The outline functionality of an analysis script  
1028 is that, at the appropriate times, a DSE script calls it, passing four or more  
1029 arguments. The script uses these arguments to locate a raw simulation results  
1030 file (`results.csv`), processes those results and then writes the objective  
1031 values into an objectives file (`objectives.json`) for that simulation.

1032 The first three arguments sent to the script are common to all scripts. These  
1033 are listed below.

1034 **argv 1** The absolute path to the folder containing the `results.csv` re-  
1035 sults file. This is also the path where the script finds the  
1036 `objectives.json` file.

1037 **argv 2** The name of the objective. This is the key against which the script  
1038 should save its results in the objectives file.

1039 **argv 3** The name of the scenario.

1040 With this information the script can find the raw simulation data and also  
1041 determine where to save its results. The name of the scenario allows the script

1042 to locate any data files it needs relating to the scenario. For example, in the  
 1043 case of the script measuring cross track error for the line following robot,  
 1044 the script makes use of a data file that contains a series of coordinates that  
 1045 represent the line to be followed. The name of this data file is `map1px.csv`.  
 1046 It is placed into a folder with the same name as the scenario, which in this  
 1047 case is `studentMap`. That folder is located in the `userMetricScripts`  
 1048 folder, as shown in Figure 64. Using this method, the developer of an external  
 1049 analysis script needs only to define the name of the data file they will need and  
 1050 know that at runtime the script will be passed a path to a folder containing  
 the data file suitable for the scenario under test.

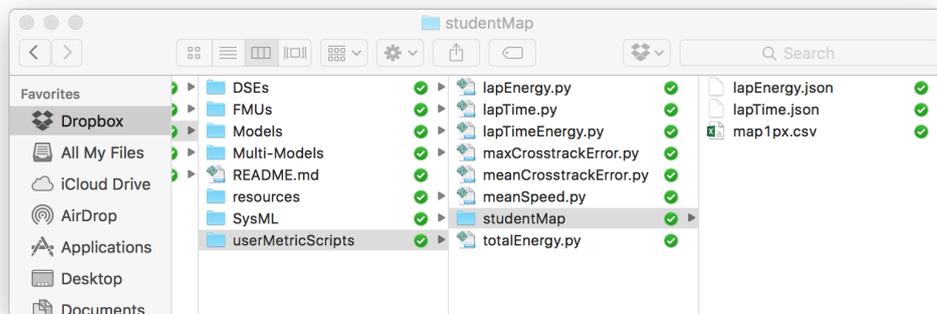


Figure 64: External analysis script data files for the “studentMap” scenario.

1051

1052 Figure 65 shows an example of an external analysis script. In this case it  
 1053 computes the cumulative deviation of the water level from some target level.  
 1054 There are two distinct sections in the file, we shall refer to them as the  
 1055 ‘common’ and ‘script specific’ sections.

1056 The common section contains core functions that are common to all ex-  
 1057 ternal scripts. It reads in the three arguments that are common to all  
 1058 scripts, and contains functions to help the user retrieve the data needed  
 1059 by the analysis script, and to write the computed objective value into the  
 1060 `objectives.json` file. It is recommended that this section be copied to  
 1061 form the basis of any new external analysis scripts.

1062 The second part of the example script shown is specific to the analysis to  
 1063 be performed. The purpose of this section is to actually compute the value  
 1064 of the objective from the results of a simulation. Generally it will have  
 1065 three parts: reading in any analysis specific arguments such as the ID of  
 1066 data in the results that it needs, using the data in `results.csv` to cal-

```

import csv,os, sys, json, io

def getColumnFor(colName, row):
    index = 0
    for thisName in row:
        if thisName.strip() == colName.strip():
            return index
        else:
            index +=1
    return index

def writeObjectiveToOutfile(key, val):
    parsed_json = {}
    if os.path.isfile(objectivesFile):
        json_data = open(objectivesFile)
        parsed_json = json.load(json_data)
    parsed_json[key] = val
    dataString = json.dumps(parsed_json, sort_keys=True, indent=4, separators=(',', ': '))
    with io.open(objectivesFile, 'w', encoding='utf-8') as f:
        f.write(unicode(dataString))

resultsFileName = "results.csv"
resultsFile = sys.argv[1] + os.path.sep + resultsFileName
objectivesFileName = "objectives.json"
objectivesFile = sys.argv[1] + os.path.sep + objectivesFileName
objectiveName = sys.argv[2]
scenarioDataFolder = sys.argv[3]
csvfile = open(resultsFile)
csvdata = csv.reader(csvfile, delimiter=',')

levelColumnID = sys.argv[4]
targetLevel = float(sys.argv[5])

cumulativeDeviation = 0.0
levelColumn = 0
stepSizeColumn = 0
firstRow = True

for row in csvdata:
    if firstRow:
        levelColumn = getColumnFor(levelColumnID, row)
        stepSizeColumn = getColumnFor('step-size', row)
        firstRow = False
    else:
        level = float(row[levelColumn])
        stepSize = float(row[stepSizeColumn])
        cumulativeDeviation += abs ((level - targetLevel)*stepSize)

writeObjectiveToOutfile(objectiveName, cumulativeDeviation)

```

Common  
Section

Script  
Specific  
Section

Figure 65: External analysis script to calculate cumulative deviation in the water tank example

1067 culate the value of the objective and finally write the objective value into  
1068 `objectives.json`.

1069 In the 'Script Specific Section' of Figure 65 we see the example of the script  
1070 calculating the cumulative deviation of the water level from a target level in  
1071 the water tank model. It starts by reading a further two arguments passed  
1072 when the script is launched and initializes the variables. The script then it-  
1073 erates through all rows of data in `results.csv` to calculate the cumulative  
1074 deviation which is then written to the `objectives.json` file in the final  
1075 line.

```
"externalScripts": {  
  "lapTime": {  
    "scriptFile": "lapTime.py",  
    "scriptParameters": {  
      "1": "time",  
      "2": "{bodyFMU}.body.robot_x",  
      "3": "{bodyFMU}.body.robot_y",  
      "4": "studentMap"  
    }  
  },  
  "meanCrossTrackError": {  
    "scriptFile": "meanCrosstrackError.py",  
    "scriptParameters": {  
      "1": "{bodyFMU}.body.robot_x",  
      "2": "{bodyFMU}.body.robot_y"  
    }  
  }  
},
```

Figure 66: Definition of the external analysis functions for the line follower robot.

1076 **Definition of External Analysis in DSE Configuration** With the  
1077 analysis scripts constructed, the next step is to define their use in the DSE  
1078 configuration file. The definition essentially contains three parts: a name for  
1079 the objective, the file name of the script and a list arguments to pass. The  
1080 name given to the objective allows it to be referenced in the objectives con-  
1081 straints and ranking sections of the DSE configuration. The file name tells  
1082 the DSE scripts which script to launch and the arguments define additional  
1083 data (over the standard three arguments described earlier) that the script  
1084 needs, such as the names of data it needs or constant values.

1085 In Figure 67 we find the definition of the external analysis used in the three  
1086 tank water tank example. There are two analysis defined, the first is named  
1087 'cumulativeDeviation' and the second is 'vCount'. In each there are two  
1088 parameters defined, the 'scriptFile' contains the file name of the script file to  
1089 run in each case, while the 'scriptParameters' parameter contains the list of

1090 additional arguments each needs.

```

"objectiveDefinitions": {
  "externalScripts": {
    "cumulativeDeviation": {
      "scriptFile": "cumulativeDeviation.py",
      "scriptParameters": {
        "1": "{tank2}.tank2.level",
        "2": "1.0"
      }
    },
    "vCount": {
      "scriptFile": "valveChanges.py",
      "scriptParameters": {
        "1": "{controller}.controller.wt3_valve"
      }
    }
  },
  "internalFunctions": {}
},

```

Figure 67: Definition of the external analysis functions for the three water tank model.

1091 The purpose of both internal and external analysis functions is to populate  
 1092 the `objectives.json` file with values that characterize the performance  
 1093 of the designs being explored. Figure 68 shows an example objectives file  
 1094 generated during a DSE of the three water tank example. There is an instance  
 1095 of the objectives file created for each simulation in DSE, its primary use being  
 1096 to inform the ranking of designs, but it may be used for any other analysis a  
 user wishes to define.

```

{
  "cumulativeDeviation": 20.47140614676141,
  "vCount": 1
}

```

Figure 68: Contents of `objectives.json` file for a single simulation of the three tank water tank

1097

### 1098 6.3.7 Ranking

1099 The final part of a DSE configuration file concerns the placing of designs in a  
 1100 partial order according to their performance. The DSE currently supports a  
 1101 Pareto method of ranking, as was shown earlier in Figure 59. The purpose of  
 1102 the ranking section of the configuration is to define the pair of objectives that  
 1103 will be used to rank the designs, and whether to maximize or minimize each.  
 1104 Figure 69 shows an example of a ranking definition from the line following  
 1105 robot example. Here the user has specified that the lap time and mean

1106 cross track error objectives will be used to rank. The use of '-' after each  
1107 indicates that the aim is to minimize both, whereas a '+' indicates the desire  
to maximize.

```
"ranking": {  
  "pareto": {  
    "lapTime": "-",  
    "meanCrossTrackError": "-"  
  }  
},
```

Figure 69: Defining parameters and their preferred directions for ranking.

1108

1109 Combining all these sections results in a complete DSE configuration, as  
1110 shown in Figure 70.

```

{
  "algorithm": {},
  "objectiveConstraints": {},
  "objectiveDefinitions": {
    "externalScripts": {
      "lapTime": {
        "scriptFile": "lapTime.py",
        "scriptParameters": {
          "1": "time",
          "2": "{bodyFMU}.body.robot_x",
          "3": "{bodyFMU}.body.robot_y",
          "4": "studentMap"
        }
      },
      "meanCrossTrackError": {
        "scriptFile": "meanCrosstrackError.py",
        "scriptParameters": {
          "1": "{bodyFMU}.body.robot_x",
          "2": "{bodyFMU}.body.robot_y"
        }
      }
    },
    "internalFunctions": {}
  },
  "parameterConstraints": [
    "{sensor1FMU}.sensor1.lf_position_y == {sensor2FMU}.sensor2.lf_position_y",
    "{sensor1FMU}.sensor1.lf_position_x == - {sensor2FMU}.sensor2.lf_position_x"
  ],
  "parameters": {
    "{sensor1FMU}.sensor1.lf_position_x": [
      0.01,
      0.03
    ],
    "{sensor1FMU}.sensor1.lf_position_y": [
      0.07,
      0.13
    ],
    "{sensor2FMU}.sensor2.lf_position_x": [
      -0.01,
      -0.03
    ],
    "{sensor2FMU}.sensor2.lf_position_y": [
      0.07,
      0.13
    ]
  },
  "ranking": {
    "pareto": {
      "lapTime": "-",
      "meanCrossTrackError": "-"
    }
  },
  "scenarios": [
    "studentMap"
  ]
}

```

Figure 70: A complete DSE configuration for the line follower robot example.

## 1111 7 Test Automation and Model Checking

1112 Test Automation and Model Checking for INTO-CPS is provided by the RT-  
1113 Tester RTT-MBT tool. This section first describes installation and configu-  
1114 ration of RT-Tester MBT in Section 7.1. It then describes test automation  
1115 in Section 7.2 and model checking in Section 7.3. Note, that these features  
1116 are explained in more detail in the deliverables D5.2a [PLM16] and D5.2b  
1117 [BLM16], respectively.

### 1118 7.1 Installation of RT-Tester RTT-MBT

1119 In order to use RTT-MBT, a number of software packages must be installed.  
1120 These software packages have been bundled into two installers:

- 1121 • **VSI tools dependencies bundle:**

1122 This bundle is required on the Windows platform and installs the fol-  
1123 lowing third party software:

- 1124 – Python 2.7.
- 1125 – GCC 4.9 compiler suite, used to compile FMUs.

- 1126 • **VSI tools – VSI Test Tool Chain:**

- 1127 – RT-Tester 6.0, a stripped version of the RT-Tester core test system  
1128 that contains the necessary functionality for INTO-CPS.
- 1129 – RT-Tester MBT 9.0, the model-based testing extension of RT-  
1130 Tester.
- 1131 – RTTUI 3.9, the RT-Tester graphical user interface.
- 1132 – Utility scripts to run RTT-MBT.
- 1133 – Examples for trying out RTT-MBT.

1134 These bundles can be downloaded via the download manager of the INTO-  
1135 CPS Application.

#### 1136 7.1.1 Setup of the RT-Tester User Interface

1137 When the RT-Tester User Interface (RTTUI) is first started, a few configu-  
1138 ration settings must be made.

- 1139      • User name and company name (Figure 71a).
- 1140      • Location of Bash shell (Figure 71b): You can safely skip this step by  
1141      clicking *Next*.
- 1142      • Path to Python 2.7 executable (Figure 71c): Click *Detect* and then  
1143      *Installation Path* for auto-detection, or *Browse* to select manually.
- 1144      • Location of RT-Tester (Figure 71d): Click *Browse* to select the direc-  
1145      tory of your RT-Tester installation. Note that if you did not specify  
1146      the Bash shell location in step 7.1.1, the version number might not be  
1147      properly detected.

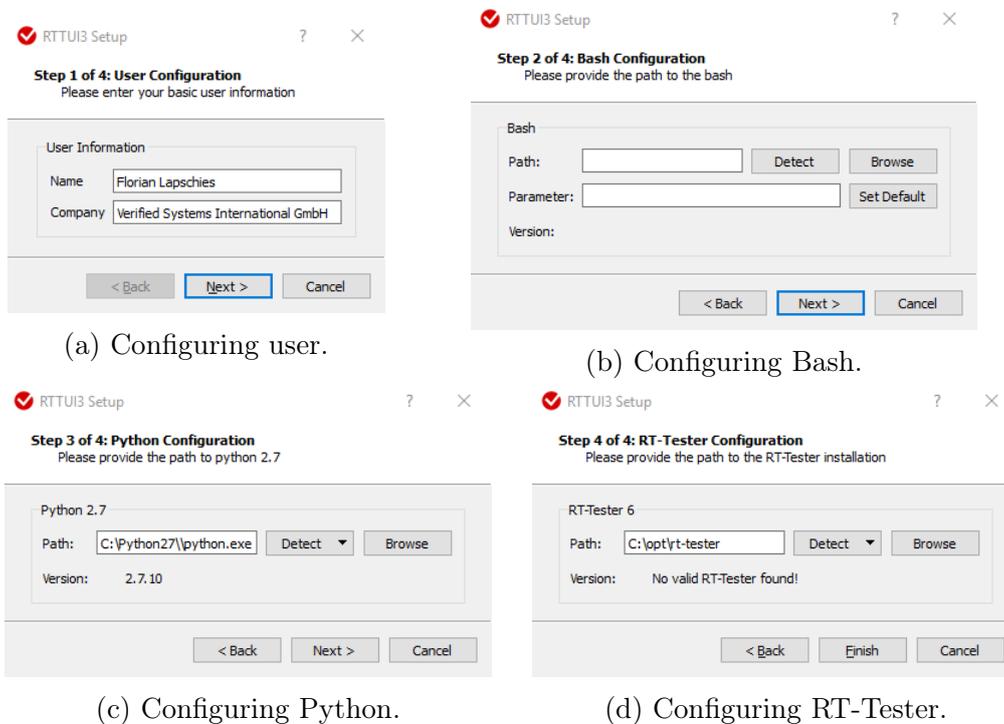


Figure 71: RT-Tester GUI configuration.

## 1148 7.2 Test Automation

1149 Configuring and using a Test Project involves several activities. These are:

- 1150      • Creating a test project.
- 1151      • Defining tests.

- 1152 • Compiling test driver FMUs.
- 1153 • Setting up test runs.
- 1154 • Running tests.
- 1155 • Evaluating test results.

1156 These activities can be performed either solely using the RT-Tester graphical  
1157 user interface, or using a combination of the INTO-CPS Application and the  
1158 RT-Tester GUI. In this section we focus on describing the latter, since it  
1159 supports the complete set of features necessary for test automation. The  
1160 INTO-CPS Application currently only exposes a subset of these. A more  
1161 comprehensive description of the test automation workflow can be found in  
1162 deliverable D5.2a [PLM16].

1163 In the INTO-CPS Application test automation functionality can be found  
1164 below the main activity *Test-Data-Generation* in the project browser. Before  
1165 using most of the test automation utilities, the license management process  
1166 has to be started. To this, end right-click on *Test-Data-Generation* and select  
*Start RT-Tester License Dongle* (see Figure 72).

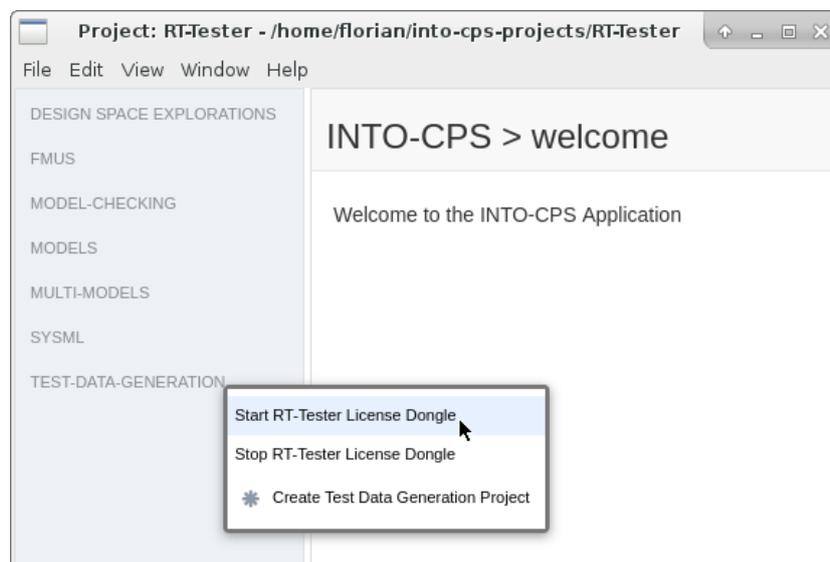


Figure 72: Starting the license management process.

1167  
1168 After developing the behavioural model in Modelio and exporting it to an  
1169 XMI file, test automation projects can be created from the INTO-CPS Ap-  
1170 plication. Such a project is then added as a sub-project within a containing  
1171 INTO-CPS Application project. To create a project, do the following:

- 1172 1. Right-click on *Test-Data-Generation* in the project browser and select  
 1173 *Create Test Data Generation Project* (see Figure 73).
- 1174 2. Specify a name for the project, select the XMI file containing the test  
 1175 model and press *Create*, as shown in Figure 74.

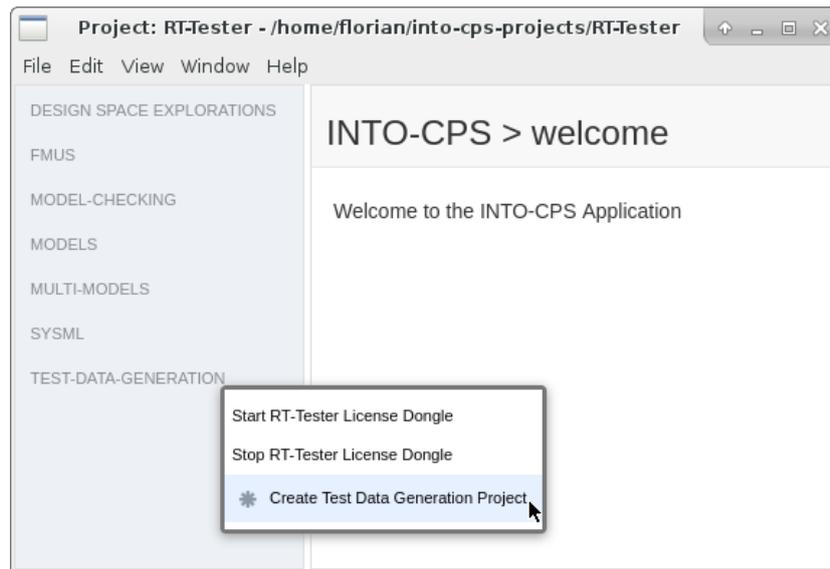


Figure 73: Creating a test automation project.

1176 The newly created sub-project and its directory hierarchy is displayed in the  
 1177 project browser. Some directories and files of the RT-Tester project that  
 1178 are not of great importance to the INTO-CPS workflow are hidden from the  
 1179 browser. The following two folders are of special significance:

- 1180 • `TestProcedures` contains symbolic test procedures where test objec-  
 1181 tives are specified in an abstract way, for example by specifying Linear  
 1182 Temporal Logic (LTL) formulas.
- 1183 • From these symbolic test procedures, concrete executable (RT-Tester 6)  
 1184 test procedures are generated, which then reside in the folder `RTT_`  
 1185 `TestProcedures`.

1186 The specification of test objectives is done using the RT-Tester GUI. The  
 1187 relevant files can be opened in the RT-Tester GUI directly from the INTO-  
 1188 CPS Application by double-clicking them:

- 1189 • `conf/generation.mbtconf` allows you to specify the overall test  
 1190 objectives of the test procedure. Test objectives can be specified as  
 1191 LTL formulas, which must then be fulfilled during a test run. Test

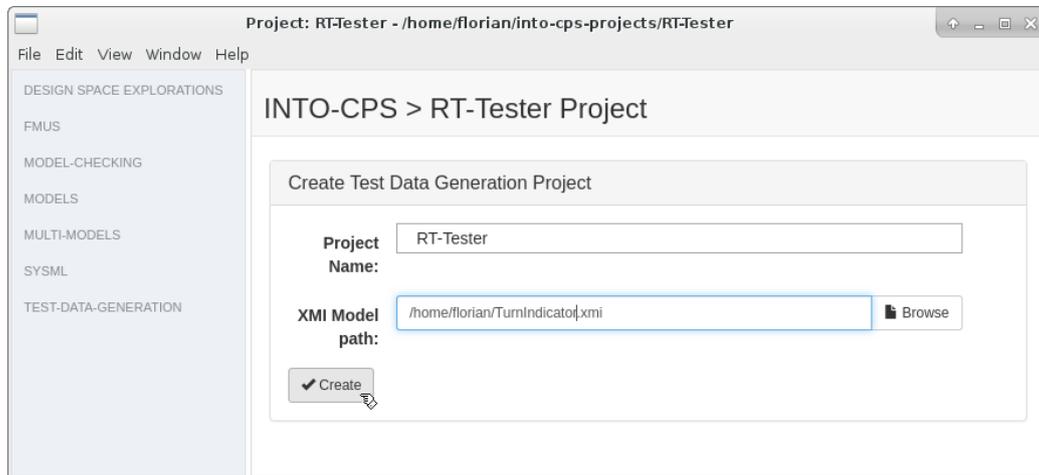


Figure 74: Test automation project specifics.

1192 goals can also be specified by selecting structural elements from a tree  
 1193 representation of the test model and then choosing a coverage metric  
 1194 for that element. For example, the user might select a sub-component  
 1195 of the System Under Test (SUT) and specify that all Basic Control  
 1196 States (BCS) must be reached (see Figure 75), or that all transitions  
 1197 must be exercised (TR) in a test run.

- 1198 • `conf/signalmap.csv` allows you to configure the input and output  
 1199 signals of the system under test (see Figure 76). This includes defining  
 1200 the admissible signal latencies for checking the SUT's outputs in a test  
 1201 run. This file also allows you to restrict the range of the signals in order  
 1202 to constrain these values during test data generation.

1203 More details on the definition of tests can be found in deliverable D5.2a  
 1204 [PLM16].

1205 After defining the test objectives, a concrete test case can be created by right-  
 1206 clicking on the symbolic test case under *TestProcedures* and then selecting  
 1207 *Solve* (see Figure 77).

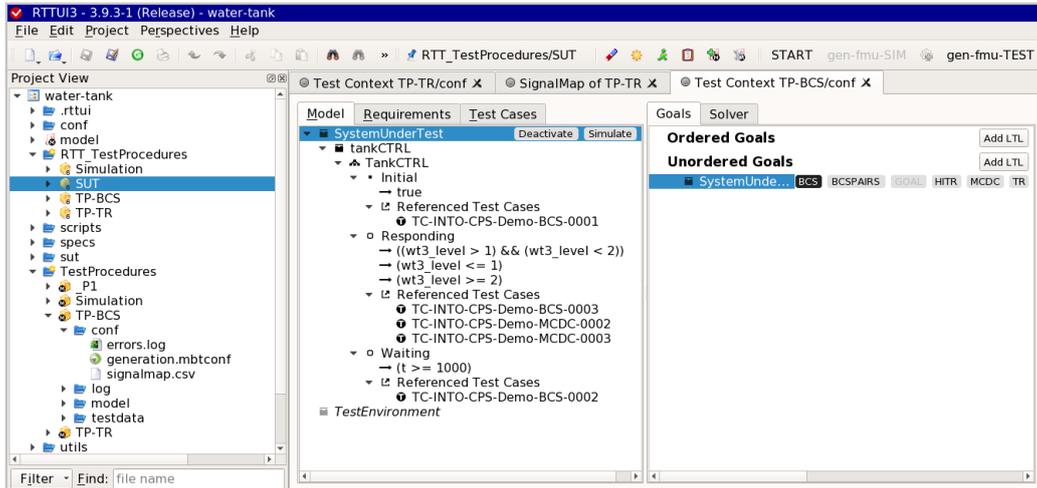


Figure 75: Configuring a test goal.

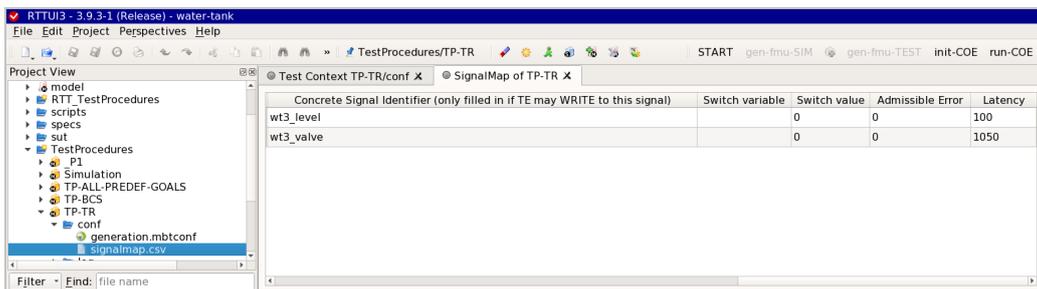


Figure 76: Configuring signals.

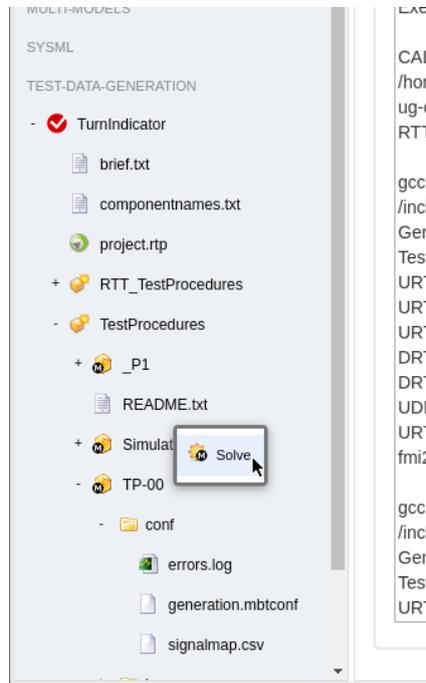


Figure 77: Generating a concrete test procedure.

1208 A solver component then computes the necessary timed inputs to realize the  
 1209 test objectives. A concrete test procedure is generated that feeds a system  
 1210 under test with these inputs and observes its outputs against expected results  
 1211 derived from the test model. This test procedure will be placed in `RTT_  
 1212 TestProcedures` and has the same name as the symbolic test procedure.  
 Figure 78 shows how test generation progresses.

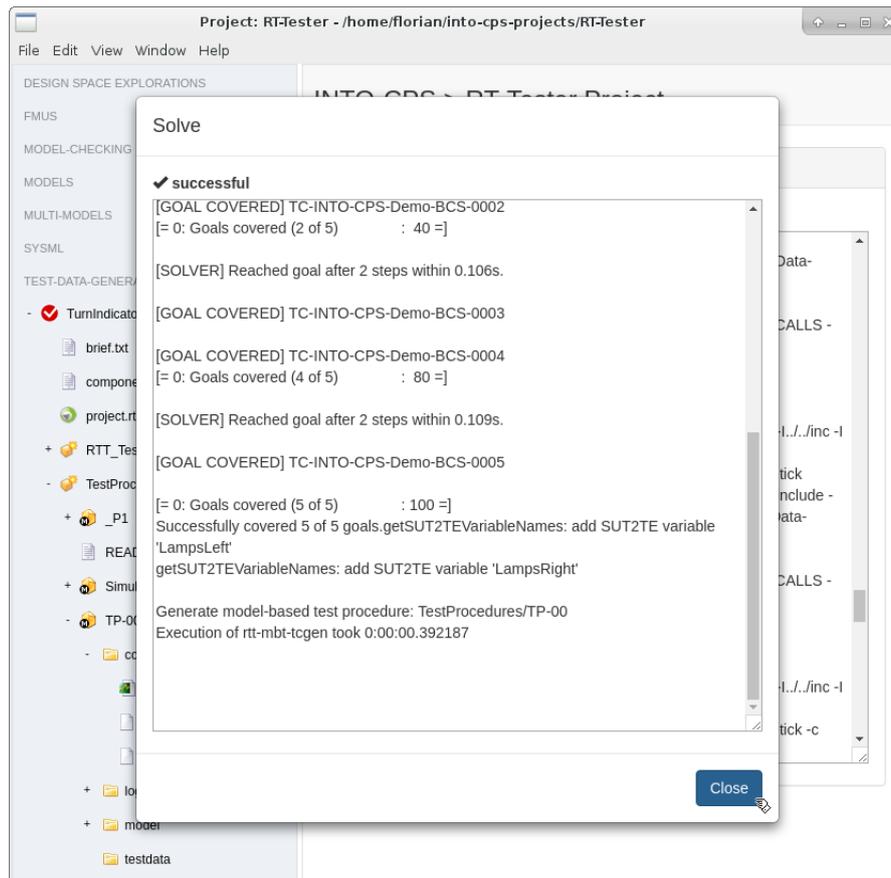


Figure 78: Test data generation progress.

1213

1214 A generated test procedure can be cast into an FMU, which can then be  
 1215 run in a co-simulation against the system under test. To this end, right  
 1216 click on the concrete test procedure and select *Generate Test FMU* (see  
 1217 Figure 79). In cases where a real and perhaps physical system under test is  
 1218 not available, a simulation of the system under test can be generated from  
 1219 the behavioural model. To generate such an FMU, right-click on *Simulation*  
 1220 and select *Generate Simulation FMU* as depicted in Figure 80.

1221 In order to run a test, right-click on the test procedure and select *Run Test*

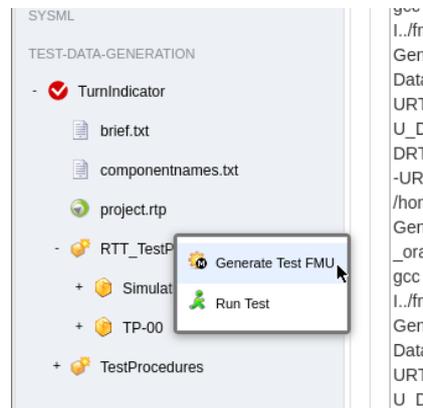


Figure 79: Generating a test FMU.

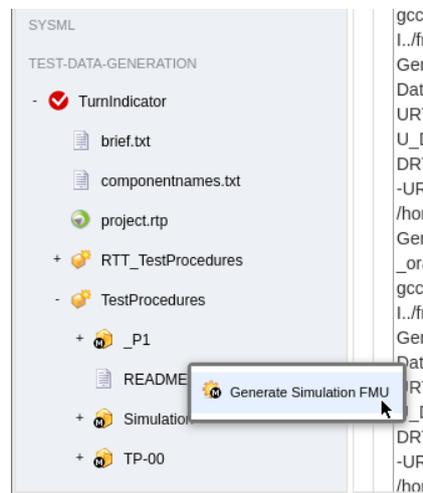


Figure 80: Generating a simulation FMU.

1222 (see Figure 81). Then specify the FMU of the system under test. If the sys-  
 1223 tem under test is to be replaced by a simulation, press on the corresponding  
 1224 *Simulation* button. The duration of the test is derived during test data gen-  
 1225 eration and does not need to be manually specified. However, an appropriate  
 1226 step size must be set. Finally, after making sure the COE is running, press  
 1227 *Run* to start the test (see Figure 82).

1228 Every test execution yields as its result an evaluation of test cases, *i. e.*, each is  
 1229 associated with a verdict of PASS, FAIL, or INCONCLUSIVE.<sup>10</sup> The details  
 1230 are found in the test log files below the folder `testdata`. See the RT-Tester

<sup>10</sup>The verdict can also be NOT TESTED. This means a test case has been included in a test procedure, but a run that reaches it is still missing.

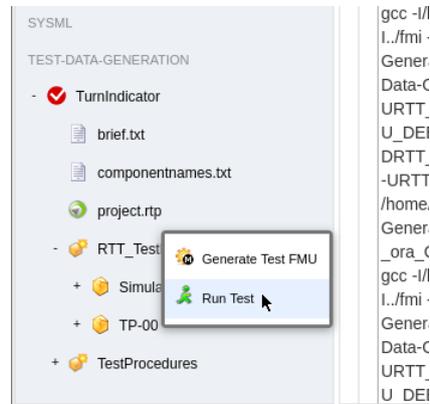


Figure 81: Running a test.

1231 user manual [Ver15a] for details.

1232 The file `testcase_tags.txt` gives a condensed record of test case, ver-  
 1233 dict, and point in a `*.log` file where a corresponding PASS, FAIL, or—  
 1234 in case of INCONCLUSIVE—test case occurrence without assertion can  
 1235 be found. The project-wide test-case verdict summary as well the require-  
 1236 ment verdict summary can be found in the folder `RTT_TestProcedures/`  
 1237 `verification`. More details on the evaluation of test runs can be found  
 1238 in deliverable D5.2a [PLM16].

### 1239 7.3 Model Checking

1240 This section describes how to use the INTO-CPS Application as a front-  
 1241 end to the LTL model checker of RT-Tester RTT-MBT. More details on the  
 1242 algorithms used and the syntax of LTL formulas can be found in deliverable  
 1243 D5.2b [BLM16].

1244 Once an INTO-CPS project has been created (see Section 4.2), model check-  
 1245 ing functionality can be found under the top-level activity *Model Checking* in  
 1246 the project browser. Before getting started, the RT-Tester license manage-  
 1247 ment process must be launched. To this end, right-click on *Model Checking*  
 1248 and select *Start RT-Tester License Dongle* (see Figure 83). Model checking  
 1249 projects are presented as sub-projects of INTO-CPS Application projects. In  
 1250 order to add a new project,

- 1251 1. Right-click on the top-level activity *Model Checking* in the project  
 1252 browser and select *Create Model Checking Project* (see Figure 84).

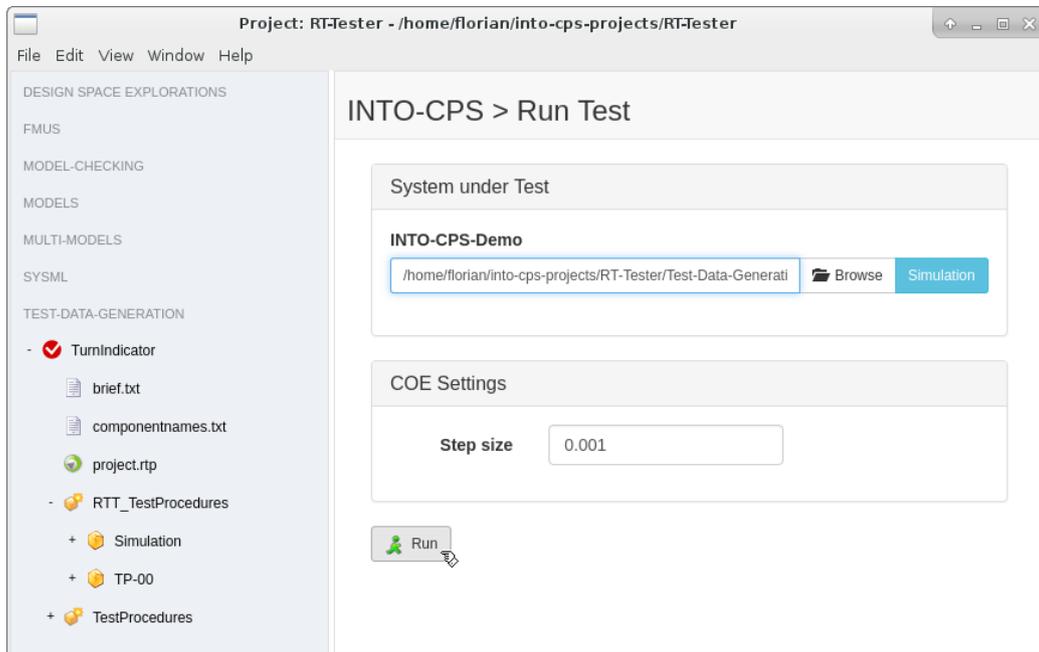


Figure 82: Configuring a test.

- 1253 2. Provide a project name and the model that has been exported to XMI  
1254 from Modelio.

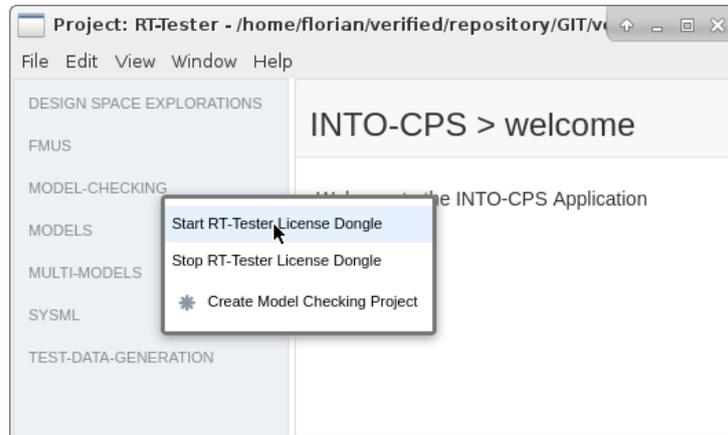


Figure 83: Starting the RT-Tester license dongle.

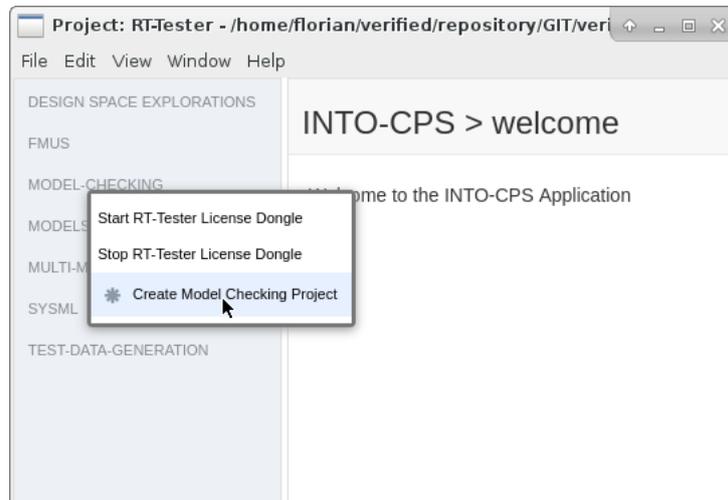


Figure 84: Creating a model checking project.

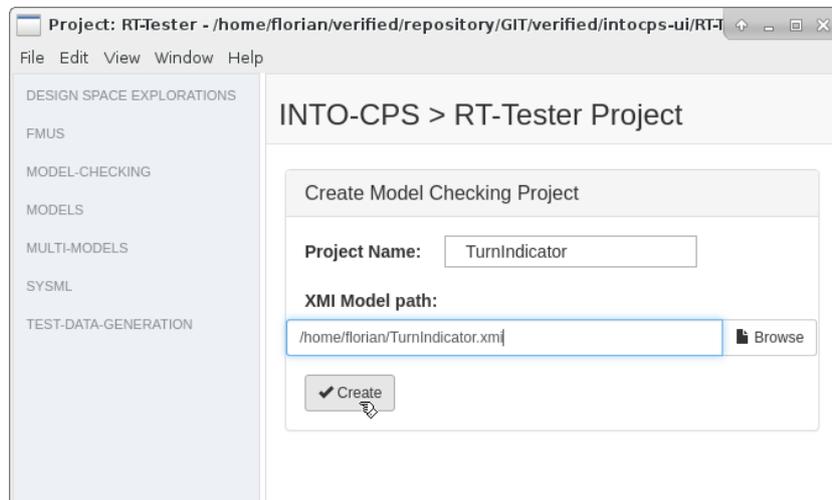


Figure 85: Specifying the model checking project.

1255 After pressing *Create*, a new node representing the model checking project is  
 1256 added to the project browser.

1257 The next step is to add LTL queries to the project:

- 1258 1. Right click on the project and select *Add LTL Query* (see Figure 86).
- 1259 2. Enter a name for the new query (see Figure 87).
- 1260 3. To edit the LTL query, double click on the corresponding node in the  
 1261 project browser (see Figure 88). The LTL formula can then be edited in  
 1262 a text field. Note that the editor supports auto-completion for variable  
 1263 names and LTL operators (see Figure 89).
- 1264 4. Provide the upper bound for the bounded model checking query.

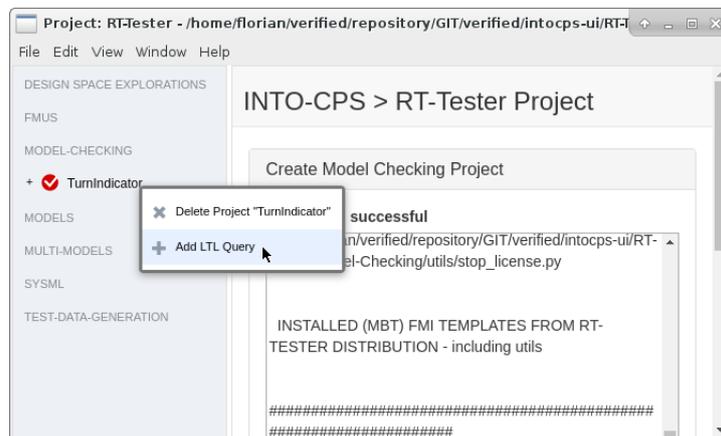


Figure 86: Adding an LTL formula.

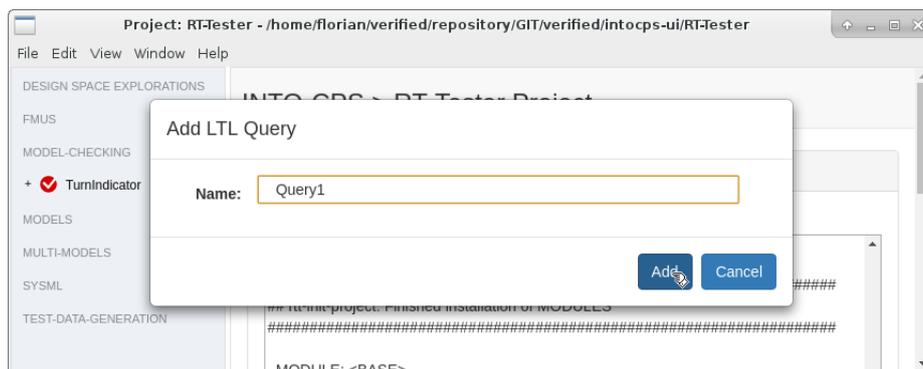


Figure 87: Naming the new LTL formula.

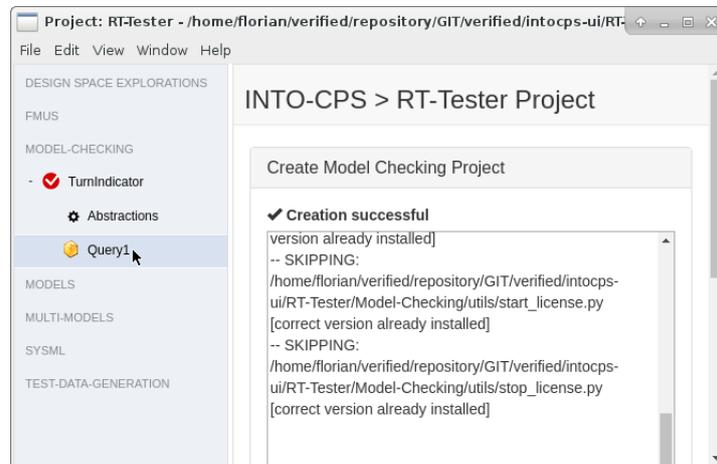


Figure 88: Opening the LTL formula editor.

1265 To check the query, press *Save & Check*. A window opens and is filled with  
 1266 the output of the model checking tool. The tool either reports that the query  
 1267 holds within the specified number of steps — as depicted in Figure 90 — or  
 1268 it prints a counterexample to demonstrate that the property does not hold.  
 1269

1270 It is possible to configure abstractions<sup>11</sup> for a particular model checking  
 1271 project. To do so, double-click on the corresponding *Abstractions* node below  
 1272 that project in the project browser. It is then possible to choose an abstrac-  
 1273 tion method for each output variable of an environment component along  
 1274 with making the associated setting. In Figure 91 the interval abstraction has  
 1275 been selected for the output variable `voltage`. This abstraction has further  
 1276 been configured to restrict the variable's value within the interval `[10, 12]`.  
 1277 After pressing *Save*, this abstraction is applied to all model checking queries  
 1278 in the current model checking project.

<sup>11</sup>Information on abstractions and their associated configuration items can be found in deliverable D5.2b [BLM16].

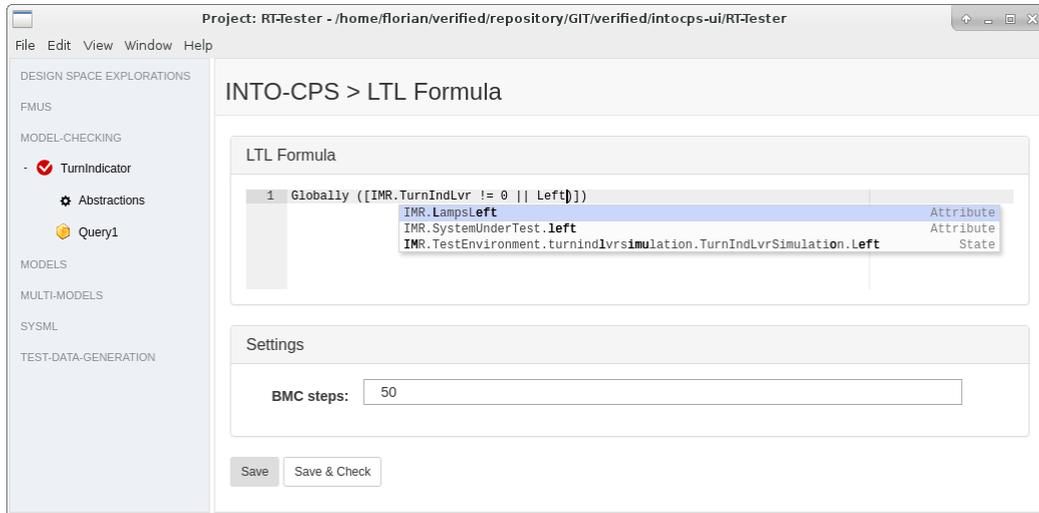


Figure 89: LTL formula editor.

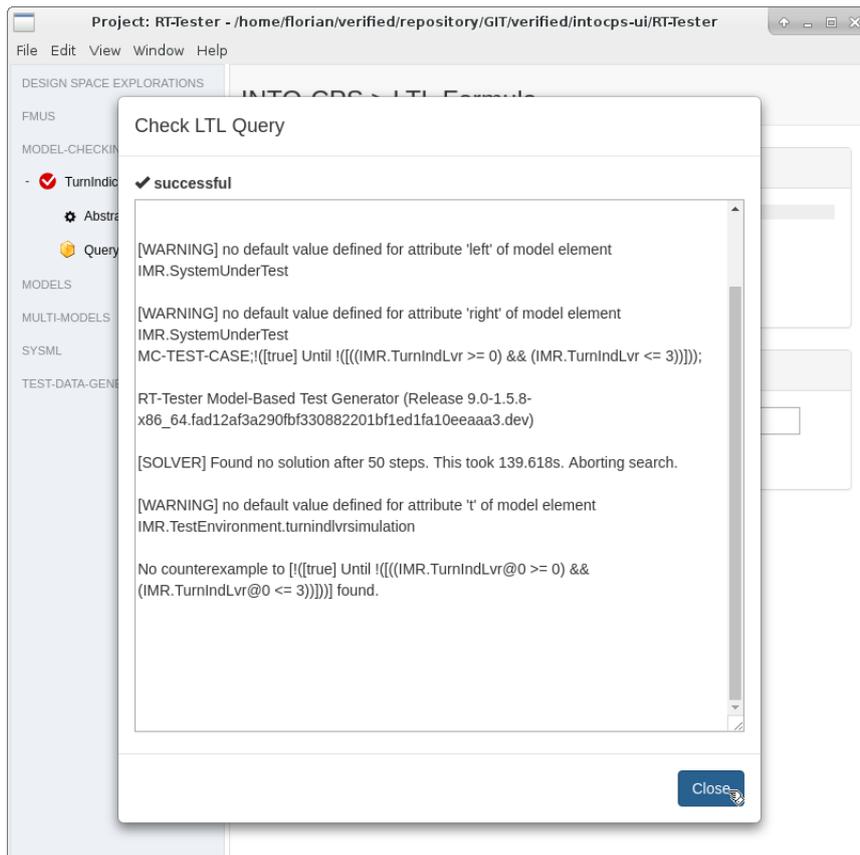


Figure 90: Model checking result.

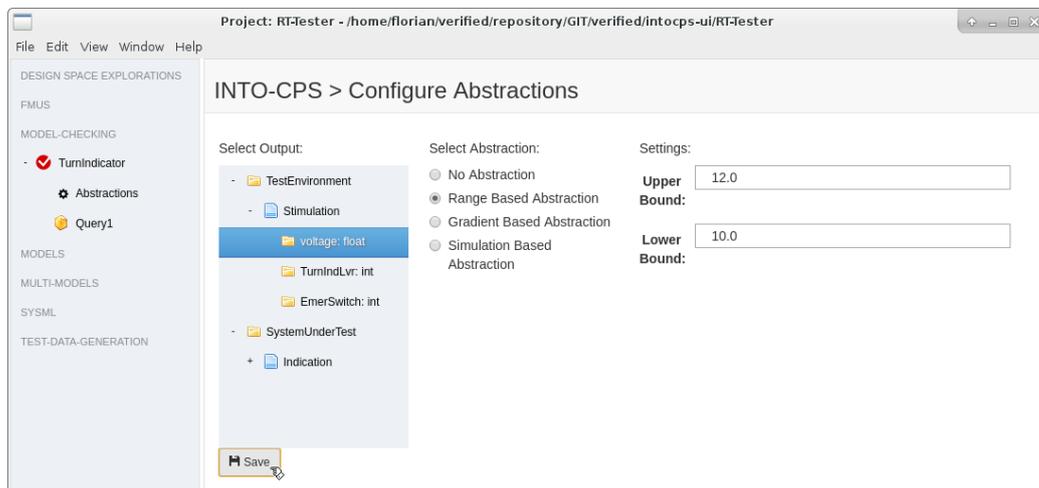


Figure 91: Configuring abstractions.

## 1279 8 Traceability support for INTO-CPS

1280 This section provides a description of tool support for traceability developed  
1281 as part of the INTO-CPS project.

### 1282 8.1 Overview

1283 Traceability support is divided into two steps: sending data from the tools  
1284 to the traceability database, and retrieving information from the database.  
1285 Currently, only the first part is available in prototypes in the different tools.  
1286 This is documented below.

### 1287 8.2 INTO-CPS application

1288 The traceability daemon is now (since INTO-CPS App 2.1.19 RC) inte-  
1289 grated in the App and it starts with the App. Only Neo4J has to be down-  
1290 loaded. To do so, one can use the download-manager of the INTO-CPS App.  
1291 When downloaded, Neo4J needs to be extracted by hand into the folder  
1292 <user>/into-cps-projects/install (the archive file is located at  
1293 <user>/into-cps-projects/install\_downloads after download).  
1294 Note that Neo4J is a singleton, so make sure all other instances of Neo4J are  
1295 down before starting the App.

1296 Traceability information is captured by the traceability daemon and stored  
1297 in a Neo4J database. The database is project specific and is deployed  
1298 on project change within the App. When running, Neo4J is accessible at  
1299 <http://localhost:7474>. Here one can view the current traceability  
1300 graph.

1301 Username and password of the databases are always:

```
1302 username = intoCPSApp  
1303 password = KLHJiK8k2378HKsg823jKKLJ89sjklJHBNf8j8JH7FxE
```

1304

1305 To view the raw data from the database, right-click on the “traceability” entry  
1306 in the project browser (in the App) and select “view traceability graph” (see  
1307 figure 92). Select the database symbol, and click in “relationship types” on  
1308 “Trace”. This shows you the graph database. By default, the view is limited

1309 to 25 entries. To change this, edit the line `MATCH p= () - [r:Trace] -> ()`  
 1310 `RETURN p LIMIT 25` and set the limit to a different value.

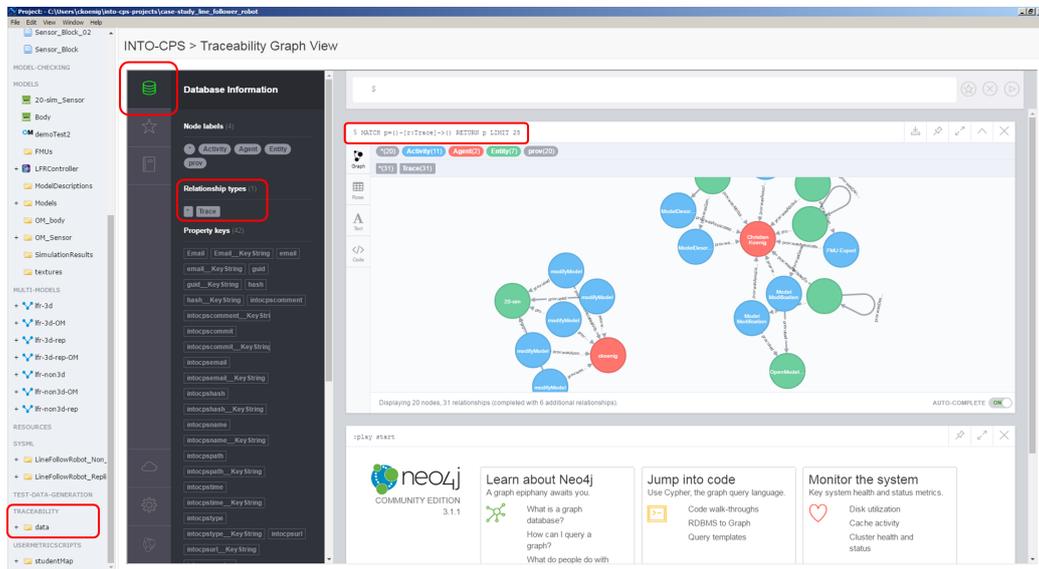


Figure 92: Current view of the traceability in the app

### 1311 8.3 Modelio

1312 The Modelio module can be downloaded here: [https://www.dropbox.com/s/bad36t9f8x4n0g1/INTOCPS\\_1.1.03.jmdac?dl=0](https://www.dropbox.com/s/bad36t9f8x4n0g1/INTOCPS_1.1.03.jmdac?dl=0). Modelio  
 1313 supports traceability for the following modelling activities:

- 1315 • Model creation
- 1316 • Model modification

1317 Steps:

1318 Go to *Configuration > Modules...* Select *INTO-CPS* and set the parameters.  
 1319 To commit a change, right click on any element and use the *INTO-CPS >*  
 1320 *Commit* command.

### 1321 8.4 OpenModelica

1322 The latest nightly builds of OpenModelica support traceability:

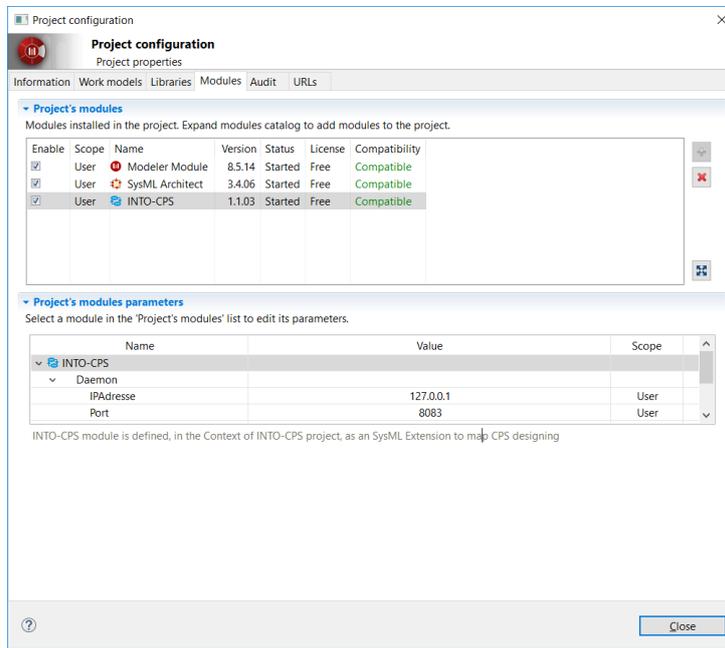


Figure 93: Configuration of traceability features in Modelio

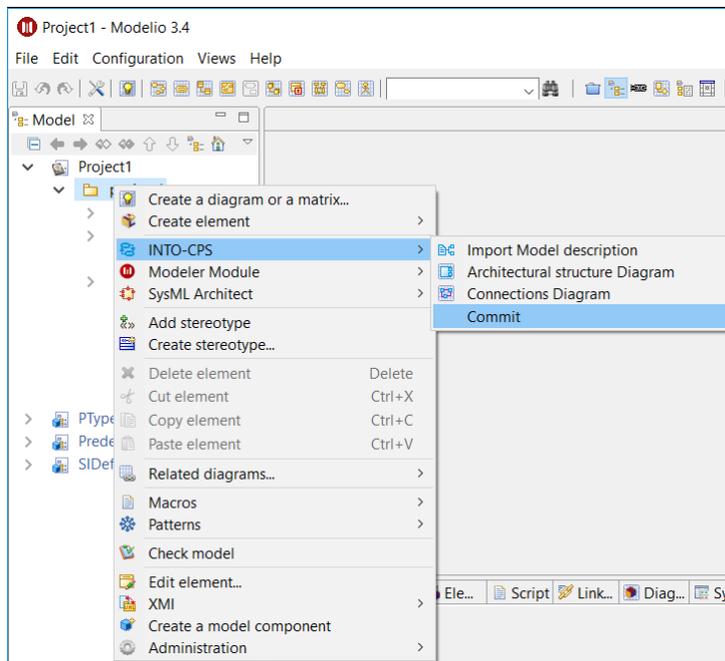


Figure 94: Commit the traceability information in Modelio

1323 Win32: `https://build.openmodelica.org/omc/builds/windows/  
 1324 nightly-builds/32bit/OpenModelica-latest.exe` Win64: `https:  
 1325 //build.openmodelica.org/omc/builds/windows/nightly-builds/  
 1326 64bit/OpenModelica-latest.exe`

1327 OpenModelica supports tracing the following modeling activities:

- 1328 • Model creation
- 1329 • Model modification
- 1330 • FMU export
- 1331 • Model description XML import

1332 As a prerequisite for traceability support, Git should be installed in the  
 1333 system.

1334 To configure the traceability support, go to *Tools > Options > Traceability*,  
 1335 select the traceability checkbox and set all the fields, the traceability daemon  
 1336 IP-Address and Port (see Figure 95). By default, the port is 8083.

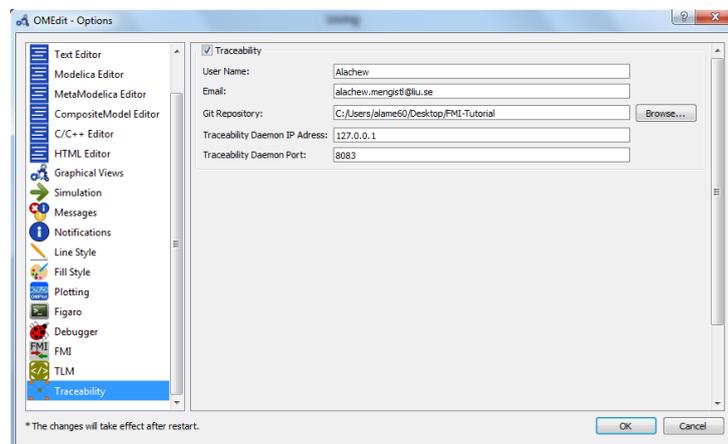


Figure 95: Configure the traceability settings in OpenModelica

1337 Then, go to *Tools > Options > General* and set the working directory to  
 1338 which you would like to export the FMU (see Figure 96).

1339 Create a Modelica model via *File > New Modelica Class* or load a model via  
 1340 *File > Open Model/LibraryFile(s)*, see Figure 97.

1341 After modification of the model/class, click the *File > Save* button, press  
 1342 Ctrl + s or click the Save button from the menu bar shown below in the

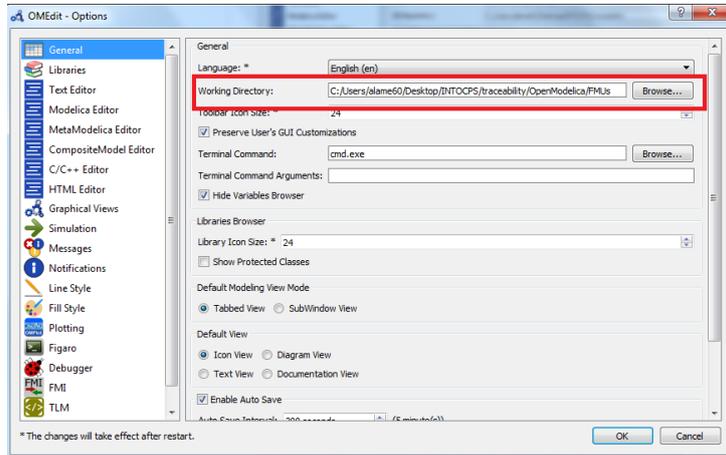


Figure 96: Set the FMU export directory in OpenModelica

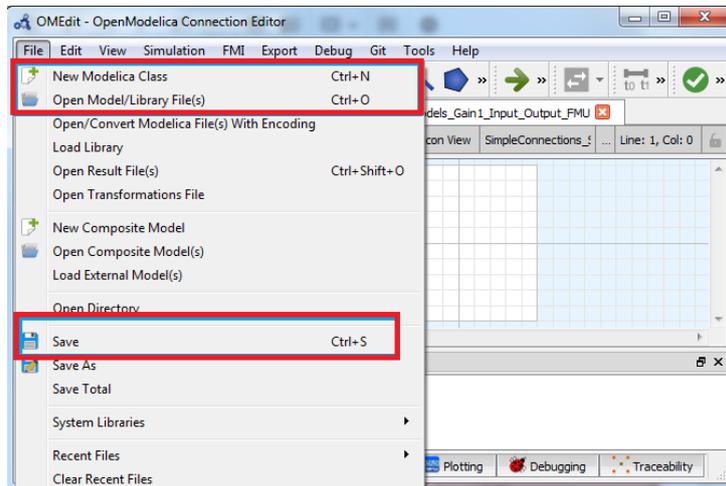


Figure 97: Create or open a Class in OpenModelica

1343 Figure 98. A dialog as shown below in Figure 99 will appear to enter the  
1344 commit description.

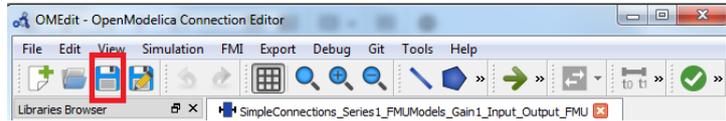


Figure 98: Save a model in OpenModelica

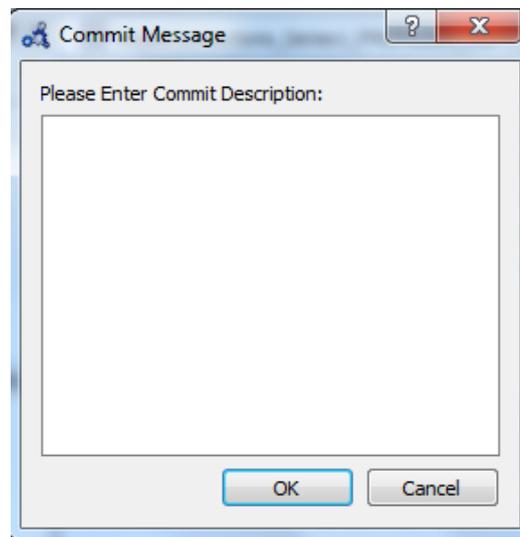


Figure 99: The commit message in OpenModelica

1345 To trace the export of an FMU, load the Modelica model or create a new  
1346 model, then go to *FMI > Export FMU* (see Figure 100). OpenModelica  
1347 generates the FMU, commits and send the traceability information to daemon  
1348 automatically.

1349 To import a `modelDescription.xml` file, go to *FMI > Import FMU*  
1350 *Model Description*. A dialog as shown in Figure 101 will appear. Select the  
1351 `modelDescription.xml` file and the output directory then press *OK*. The  
1352 Modelica model with SysML block inputs and outputs will be generated and  
1353 automatically loaded (see the left part of Figure 101). To send the traceability  
1354 information, double click on the model then go to *Git > Traceability > Push*  
1355 *Traceability Information*.

1356 To visualize the traceability graph, click on the Traceability perspective but-  
1357 ton shown below.

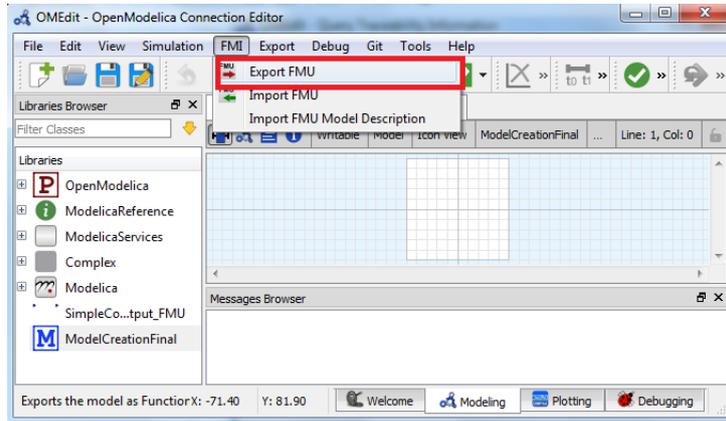


Figure 100: FMU export in OpenModelica

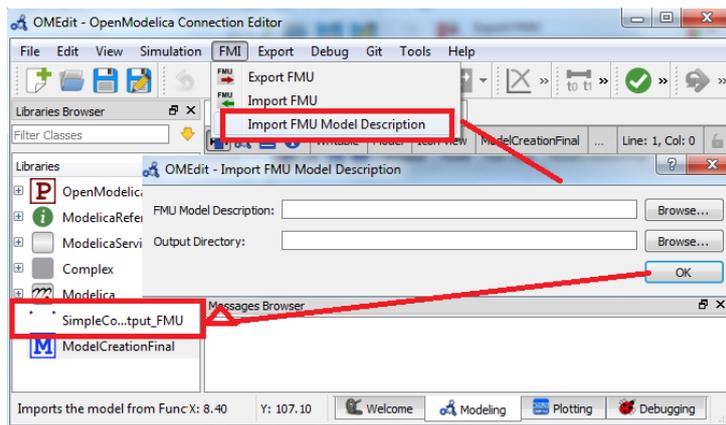


Figure 101: FMI model description import in OpenModelica

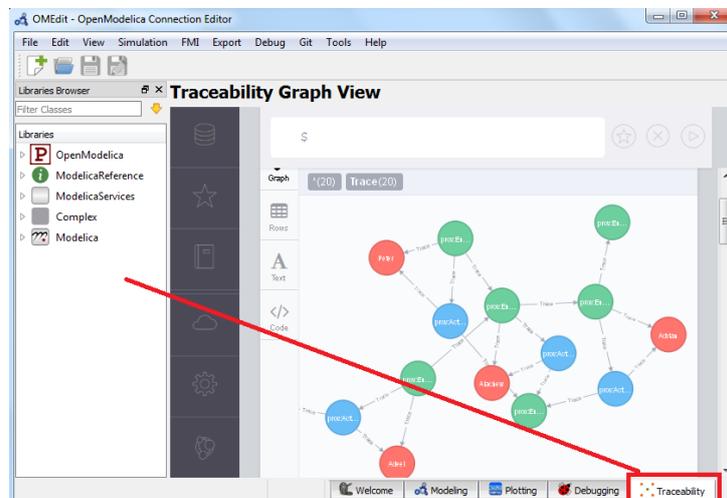


Figure 102: View traceability data in OpenModelica

## 1358 8.5 20-sim

1359 Use any version of 20-sim 4.6.3-intocps or higher. The one in the download  
 1360 manager for version 2.1.19 RC is not sufficient. The first suitable release  
 1361 bundle is 0.0.12.

1362 The download can be found here:

1363 [https://www.dropbox.com/s/lgr461ddb97kl8h/20-sim-4.6.](https://www.dropbox.com/s/lgr461ddb97kl8h/20-sim-4.6.3.7711-intocps-win32.exe?dl=1)  
 1364 [3.7711-intocps-win32.exe?dl=1.](https://www.dropbox.com/s/lgr461ddb97kl8h/20-sim-4.6.3.7711-intocps-win32.exe?dl=1)

1365 During installation, make sure you keep the Python option enabled. This is  
 1366 necessity, even if you already have another Python installation on your PC.  
 1367 This Python version will only overwrite Python versions you installed earlier  
 1368 with 20-sim, it will not install other Python versions.

1369 Currently, the actions “create model” and “modify model” are supported by  
 1370 20-sim

1371 In 20-sim, go to *Tools > Version Control Toolbox > Traceability*. First enable  
 1372 “GIT version control” and insert a GIT repository, which can be an existing  
 1373 GIT repository or a folder in the local file system. The model will be com-  
 1374 mitted to this repository on a “save” (modify) or “save as” (create) action. If  
 1375 the model does not reside in the GIT repository, it will also be copied to the  
 1376 GIT repository on a “save” or “save as” action.

1377 You can leave the “Write custom save messages” option unchecked, as it is  
 1378 not currently fully functional.

1379 If you would like to send data to the traceability daemon as well, then you  
1380 can enable “INTO-CPS Traceability Daemon”. Below, you can then enter the  
1381 IP-address and Port of the daemon. If you run the INTO-CPS application  
1382 and traceability daemon locally, the IP-address is *localhost* and the port is  
1383 8083 by default.

1384 Now, pressing “save” or “save as” in any form, will (copy and) commit your  
1385 model to the GIT repository, and then send the action you just performed  
1386 to the traceability daemon.

## 1387 9 Code Generation for INTO-CPS

1388 Of all the INTO-CPS tools, Overture, OpenModelica and 20-sim have the  
1389 ability, to varying degrees, to translate models into platform-independent C  
1390 source code. Overture can moreover translate VDM models written in the  
1391 executable subset of VDM++ [LLB11] (itself a subset of VDM-RT) to Java,  
1392 but C is the language of interest for the INTO-CPS technology.

1393 The purpose of translating models into source code is twofold. First, the  
1394 source code can be compiled and wrapped as standalone FMUs for co-  
1395 simulation, such that the source tool is not required. Second, with the aid of  
1396 existing C compilers, the automatically generated source code can be com-  
1397 piled for specific hardware targets.

1398 The INTO-CPS approach is to use 20-sim 4C to compile and deploy the code  
1399 to hardware targets, since the tool incorporates the requisite knowledge re-  
1400 garding compilers, target configuration *etc.* This is usually done for control  
1401 software modelled in one of the high-level modelling notations, after valida-  
1402 tion through the INTO-CPS tool chain. Deployment to target hardware is  
1403 also used for SiL and HiL validation and prototyping.

1404 For each of the modelling and simulation tools of the INTO-CPS tool chain,  
1405 code generation is a standalone activity. As such, the reader should refer to  
1406 the tool-specific documentation referenced in Appendix B for guidance on  
1407 code generation. Deliverable D5.1d [HLG<sup>+</sup>15] contains the details of how  
1408 each tool approaches code generation.

1409 The remainder of this section lists information about the code generation  
1410 capabilities of each tool. It describes what the user can expect currently  
1411 from each tool's code generator, in the hopes that this will be helpful in  
1412 eliminating stumbling blocks for new users trying to quickly get started with  
1413 the INTO-CPS tool chain. Extensive guidance on how to tailor models for  
1414 problem-free translation to code can be found in the tools' individual user  
1415 manuals, as referenced in Appendix B.

### 1416 9.1 Overture

1417 A complete description of Overture's C code generator can be found in the  
1418 Overture User Manual, accessible through Overture's Help system. As a  
1419 quick-start guide, this section only provides an introduction to invoking the  
1420 C code generator, and an overview of the features of VDM-RT that are

1421 currently considered stable from a code generation point of view. Please note  
1422 that exporting a source code FMU with Overture (Section 5.1) automatically  
1423 invokes the code generator and packages the result as an FMU.

1424 The C code generator is invoked from the context menu in the Project Explorer as shown in Figure 103. The code generator currently supports the

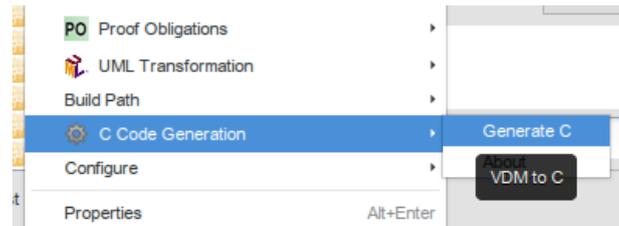


Figure 103: Invoking the code generator.

1425 following VDM-RT language constructs:  
1426

- 1427 • Basic data types and operations: integers, reals, booleans, *etc.*
- 1428 • The `is_` type test for basic types.
- 1429 • Quote types.
- 1430 • `let` expressions.
- 1431 • Pattern matching.
- 1432 • For and while loops.
- 1433 • case expressions.
- 1434 • Record types.
- 1435 • Products.
- 1436 • Aggregate types and operations: sets, sequences, maps (to a limited  
1437 extent).
- 1438 • Object-oriented features: classes and class field access, inheritance,  
1439 method overloading and overriding, the `self` keyword, subclass re-  
1440 sponsibility, *is not yet specified*, multiple constructors, and  
1441 constructor calls within constructors.
- 1442 • The `time` expression.

1443 The following language features are not yet supported:

- 1444 • Lambda expressions.

- 1445     • Pre-conditions, post-conditions and invariants.
- 1446     • Quantifiers.
- 1447     • Type queries on class instances.
- 1448     • File I/O via the I/O library.

1449 Most importantly, the development of Overture's C code generator is now be-  
1450 ing geared toward resource-constrained embedded platforms. Improvements  
1451 are currently being made to enable deployment of the generated code on PIC  
1452 and ATmega microcontrollers.

1453 A key feature of this development is the use of a garbage collector for memory  
1454 management. Generating a VDM-RT model to C code via the context menu  
1455 results in a `main.c` file containing a skeletal `main()` function. This function  
1456 contains calls to `vdm_gc_init()` and `vdm_gc_shutdown()`, the garbage  
1457 collector initialization and shutdown functions. The collector proper can not  
1458 be invoked automatically, so calls to the essential function `vdm_gc()` must  
1459 be inserted manually in the main code, for instance after each repetition of a  
1460 cyclic task. The source code FMU exporter, on the other hand, can handle  
1461 automatic invocation of the garbage collector, so no manual intervention is  
1462 required. Please note that it is generally unsafe to insert calls to `vdm_gc()`  
1463 in the generated code.

## 1464 9.2 20-sim

1465 20-sim supports ANSI-C and C++ code generation through the usage of  
1466 external and user-modifiable code-generation templates. Currently only a  
1467 subset of the supported 20-sim modelling language elements can be exported  
1468 as ANSI-C or C++ code code. The exact supported features depend on the  
1469 chosen template and its purpose and are discussed in Section 5.2.

1470 The main purpose of the 20-sim code generator is to export control systems.  
1471 Therefore the focus is on running code on bare-bone targets (*e.g.* Arduino)  
1472 or as a real-time task on a real-time operating system.

1473 The code generated by 20-sim does not contain any target-related or operat-  
1474 ing system specific code. The exported code is generated such that it can be  
1475 embedded in an external software project. For running 20-sim generated code  
1476 on a target, you can use 20-sim 4C. This is a tool that extends the 20-sim  
1477 generated code with target code based on target templates [Con16].

### 1478 **9.3 OpenModelica**

1479 OpenModelica supports code generation from Modelica to source-code tar-  
1480 geting both ANSI-C and C++. From the generated source code, co-simulation  
1481 and model-exchange FMUs can be built. Currently, the only supported solver  
1482 in the generated co-simulation FMUs is forward Euler. Work to support ad-  
1483 ditional solvers is underway. The ability to deploy the generated code to  
1484 specific hardware targets will be supported via 20-sim 4C.

### 1485 **9.4 RT-Tester/RTT-MBT**

1486 When generating test FMUs from SysML discrete-event state-chart specifi-  
1487 cations using RTTester/RTT-MBT, the user should be aware of the following  
1488 sources of errors:

- 1489 • Livelock resulting from a transition cycle in the state-chart specification  
1490 in which all transition guards are true simultaneously. This can be  
1491 checked separately using a livelock checker.
- 1492 • Race conditions arising from parallel state-charts assigning different  
1493 values to the same variable. Model execution in this case will deadlock.
- 1494 • State-charts specifying a replacement SUT must be deterministic.

## 1495 **10 Issue handling**

1496 Should you experience an issue while using one or more of the INTO-CPS  
1497 tools, please take the time to report the issue to the INTO-CPS project team,  
1498 so we can help you resolve it as soon as possible.

1499 The following three small sub-sections will guide you through the three simple  
1500 steps of issue handling and reporting.

### 1501 **10.1 Are you using the newest INTO-CPS release?**

1502 Before you go any further with your current issue, please check that the  
1503 INTO-CPS version you are using is the newest. The version number is part  
1504 of the file name of the ZIP-bundle of the release. To find the list of released

1505 INTO-CPS bundle versions, and to see what the current version of INTO-  
1506 CPS is, please visit

1507 `https://github.com/into-cps/intocps-ui/releases/`

## 1508 **10.2 Has the issue already been reported?**

1509 To make it easy for you to check whether the issue you are experiencing is  
1510 an already known one, we have created a list of all currently known issues  
1511 across all the INTO-CPS tools, with links directly to the online issue report  
1512 page of the relevant tool supplier. Have a quick look at the list, and if your  
1513 issue is already known, we recommend you follow the link and read more  
1514 about the specifics of the issue. Perhaps someone has found a work-around  
1515 or perhaps you have new information to add that might help the developers  
1516 solve the issue faster.

1517 For the list of currently known issues, please visit

1518 `http://into-cps.github.io/weekly-issue/index.html`

1519 Note that some of the issue tracker sites might require you to register before  
1520 you can view or submit issues. Registration is free.

## 1521 **10.3 Reporting a new issue**

1522 If you have followed the steps in the two previous sections and are now  
1523 certain that you have spotted a new issue relating to a specific INTO-CPS  
1524 tool, please visit the issue tracker site for that tool and report it. To ease  
1525 this process we have listed direct links for each tool to their relevant online  
1526 issue reporting page. To see the list of issue tracker links please visit

1527 `http://into-cps.github.io/report-an-issue.html`

# 1528 **11 Conclusions**

1529 This deliverable is the user manual for the INTO-CPS tool chain after the  
1530 second year of the project. The tool chain supports model-based design and  
1531 validation of CPSs, with an emphasis on multi-model co-simulation.

1532 Several independent simulation tools are orchestrated by a custom co-simu-  
1533 lation orchestration engine, which implements both fixed and variable step  
1534 size co-simulation semantics. A multi-model thus co-simulated can be fur-  
1535 ther verified through automated model-based testing and bounded model  
1536 checking.

1537 The tool chain benefits from a cohesive management interface, the INTO-  
1538 CPS Application, the main gateway to modelling and validation with the  
1539 INTO-CPS technology. Following the manual should give a new user of the  
1540 INTO-CPS tool chain an understanding of all the elements of the INTO-CPS  
1541 vision for co-simulation. This manual is accompanied by tutorial material  
1542 and guidance on the main INTO-CPS tool chain website,

1543 `http://into-cps.github.io`

1544 Features that have not yet been fully developed or integrated with the INTO-  
1545 CPS Application are currently being addressed and are targeted for the final  
1546 year of the INTO-CPS project.

## References

- 1547
- 1548 [ACM<sup>+</sup>16] Nuno Amalio, Ana Cavalcanti, Alvaro Miyazawa, Richard Payne,  
1549 and Jim Woodcock. Foundations of the SysML for CPS modelling.  
1550 Technical report, INTO-CPS Deliverable, D2.2a, December 2016.
- 1551 [BHJ<sup>+</sup>06] Armin Biere, Keijo Heljanko, Tommi A. Juntilla, Timo Latvala,  
1552 and Viktor Schuppan. Linear encodings of bounded LTL model  
1553 checking. *Logical Methods in Computer Science*, 2(5), 2006.
- 1554 [BHPG16] Victor Bandur, Miran Hasanagic, Adrian Pop, and Marcel  
1555 Groothuis. FMI-Compliant Code Generation in the INTO-CPS Tool  
1556 Chain. Technical report, INTO-CPS Deliverable, D5.2c, December  
1557 2016.
- 1558 [BLL<sup>+</sup>15] Victor Bandur, Peter Gorm Larsen, Kenneth Lausdahl, Sune  
1559 Wolff, Carl Gamble, Adrian Pop, Etienne Brosse, Jörg Brauer, Flo-  
1560 rian Lapschies, Marcel Groothuis, and Christian Kleijn. User Man-  
1561 ual for the INTO-CPS Tool Chain. Technical report, INTO-CPS  
1562 Deliverable, D4.1a, December 2015.
- 1563 [BLM16] Jörg Brauer, Florian Lapschies, and Oliver Möller. Implementation  
1564 of a Model-Checking Component. Technical report, INTO-CPS De-  
1565 liverable, D5.2b, December 2016.
- 1566 [Blo14] Torsten Blochwitz. Functional Mock-up Interface for Model Ex-  
1567 change and Co-Simulation. [https://www.fmi-standard.  
1568 org/downloads](https://www.fmi-standard.org/downloads), July 2014.
- 1569 [BQ16] Etienne Brosse and Imran Quadri. SysML and FMI in INTO-CPS.  
1570 Technical report, INTO-CPS Deliverable, D4.2c, December 2016.
- 1571 [Bro97] Jan F. Broenink. Modelling, Simulation and Analysis with 20-Sim.  
1572 *Journal A Special Issue CACSD*, 38(3):22–25, 1997.
- 1573 [CFTW16] Ana Cavalcanti, Simon Foster, Bernhard Thiele, and Jim Wood-  
1574 cock. Initial semantics of Modelica. Technical report, INTO-CPS  
1575 Deliverable, D2.2c, December 2016.
- 1576 [Con13] Controllab Products B.V. <http://www.20sim.com/>, January 2013.  
1577 20-sim official website.
- 1578 [Con16] Controllab Products B.V. <http://www.20sim4C.com/>, October  
1579 2016. 20-sim 4C official website.

- 1580 [CW16] Ana Cavalcanti and Jim Woodcock. Foundations for FMI comod-  
1581 elling. Technical report, INTO-CPS Deliverable, D2.2d, December  
1582 2016.
- 1583 [Fav05] Jean-Marie Favre. Foundations of Model (Driven) (Reverse) Engi-  
1584 engineering : Models – Episode I: Stories of The Fidus Papyrus and of  
1585 The Solarus. In *Language Engineering for Model-Driven Software*  
1586 *Development*, March 2005.
- 1587 [FCC<sup>+</sup>16] Simon Foster, Ana Cavalcanti, Samuel Canham, Ken Pierce, and  
1588 Jim Woodcock. Final Semantics of VDM-RT. Technical report,  
1589 INTO-CPS Deliverable, D2.2b, December 2016.
- 1590 [FE98] Peter Fritzson and Vadim Engelson. Modelica - A Unified Object-  
1591 Oriented Language for System Modelling and Simulation. In *EC-*  
1592 *COP '98: Proceedings of the 12th European Conference on Object-*  
1593 *Oriented Programming*, pages 67–90. Springer-Verlag, 1998.
- 1594 [FGPP16] John Fitzgerald, Carl Gamble, Richard Payne, and Ken Pierce.  
1595 Method Guidelines 2. Technical report, INTO-CPS Deliverable,  
1596 D3.2a, December 2016.
- 1597 [Fri04] Peter Fritzson. *Principles of Object-Oriented Modeling and Simula-*  
1598 *tion with Modelica 2.1*. Wiley-IEEE Press, January 2004.
- 1599 [Gam16] Carl Gamble. DSE in the INTO-CPS Platform. Technical report,  
1600 INTO-CPS Deliverable, D5.2d, December 2016.
- 1601 [GFR<sup>+</sup>12] Anand Ganeson, Peter Fritzson, Olena Rogovchenko, Adeel As-  
1602 ghar, Martin Sjölund, and Andreas Pfeiffer. An OpenModelica  
1603 Python interface and its use in pysimulator. In Martin Otter and  
1604 Dirk Zimmer, editors, *Proceedings of the 9th International Model-*  
1605 *ica Conference*. Linköping University Electronic Press, September  
1606 2012.
- 1607 [HLG<sup>+</sup>15] Miran Hasanagić, Peter Gorm Larsen, Marcel Groothuis, Despina  
1608 Davoudani, Adrian Pop, Kenneth Lausdahl, and Victor Bandur.  
1609 Design Principles for Code Generators. Technical report, INTO-  
1610 CPS Deliverable, D5.1d, December 2015.
- 1611 [KG16] C. Kleijn and M.A. Groothuis. *Getting Started with 20-sim 4.5*.  
1612 Controllab Products B.V., 2016.
- 1613 [KGD16] C. Kleijn, M.A. Groothuis, and H.G. Differ. *20-sim 4.6 Reference*  
1614 *Manual*. Controllab Products B.V., 2016.

- 1615 [KR68] D.C. Karnopp and R.C. Rosenberg. *Analysis and Simulation of*  
1616 *Multiport Systems: the bond graph approach to physical system dy-*  
1617 *namics*. MIT Press, Cambridge, MA, USA, 1968.
- 1618 [KS08] Daniel Kroening and Ofer Strichman. *Decision Procedures - An*  
1619 *Algorithmic Point of View*. Texts in Theoretical Computer Science.  
1620 An EATCS Series. Springer, 2008.
- 1621 [LBF<sup>+</sup>10] Peter Gorm Larsen, Nick Battle, Miguel Ferreira, John Fitzgerald,  
1622 Kenneth Lausdahl, and Marcel Verhoef. The Overture Initiative –  
1623 Integrating Tools for VDM. *SIGSOFT Softw. Eng. Notes*, 35(1):1–6,  
1624 January 2010.
- 1625 [Lin15] Linköping University. <http://www.openmodelica.org/>, August  
1626 2015. OpenModelica official website.
- 1627 [LLB11] Kenneth Lausdahl, Peter Gorm Larsen, and Nick Battle. A Deter-  
1628 ministic Interpreter Simulating A Distributed real time system using  
1629 VDM. In Shengchao Qin and Zongyan Qiu, editors, *Proceedings of*  
1630 *the 13th international conference on Formal methods and software*  
1631 *engineering*, volume 6991 of *Lecture Notes in Computer Science*,  
1632 pages 179–194, Berlin, Heidelberg, October 2011. Springer-Verlag.  
1633 ISBN 978-3-642-24558-9.
- 1634 [LLJ<sup>+</sup>13] Peter Gorm Larsen, Kenneth Lausdahl, Peter Jørgensen, Joey  
1635 Coleman, Sune Wolff, and Nick Battle. Overture VDM-10 Tool  
1636 Support: User Guide. Technical Report TR-2010-02, The Overture  
1637 Initiative, [www.overturetool.org](http://www.overturetool.org), April 2013.
- 1638 [LLW<sup>+</sup>15] Kenneth Lausdahl, Peter Gorm Larsen, Sune Wolf, Victor Ban-  
1639 dur, Anders Terkelsen, Miran Hasanagić, Casper Thule Hansen, Ken  
1640 Pierce, Oliver Kotte, Adrian Pop, Etienne Brosse, Jörg Brauer, and  
1641 Oliver Möller. Design of the INTO-CPS Platform. Technical report,  
1642 INTO-CPS Deliverable, D4.1d, December 2015.
- 1643 [LNH<sup>+</sup>16] Kenneth Lausdahl, Peter Niermann, Jos Höll, Carl Gamble,  
1644 Oliver Mölle, Etienne Brosse, Tom Bokhove, Luis Diogo Couto,  
1645 and Adrian Pop. INTO-CPS Traceability Design. Technical report,  
1646 INTO-CPS Deliverable, D4.2d, December 2016.
- 1647 [LRVG11] Kenneth G. Lausdahl, Augusto Ribeiro, Peter Visser, and Frank  
1648 Groen. D3.2b co-simulation. DESTTECS Deliverable D3.2b, The  
1649 DESTTECS Project (INFSO-ICT-248134), January 2011.
- 1650 [Ope] Open Source Modelica Consortium. OpenModelica User’s Guide.

- 1651 [PBLG15] Adrian Pop, Victor Bandur, Kenneth Lausdahl, and Frank Groen.  
1652 Integration of Simulators using FMI. Technical report, INTO-CPS  
1653 Deliverable, D4.1b, December 2015.
- 1654 [PBLG16] Adrian Pop, Victor Bandur, Kenneth Lausdahl, and Frank Groen.  
1655 Updated Integration of Simulators in the INTO-CPS Platform.  
1656 Technical report, INTO-CPS Deliverable, D4.2b, December 2016.
- 1657 [PGP<sup>+</sup>16] Richard Payne, Carl Gamble, Ken Pierce, John Fitzgerald, Simon  
1658 Foster, Casper Thule, and Rene Nilsson. Examples Compendium 2.  
1659 Technical report, INTO-CPS Deliverable, D3.5, December 2016.
- 1660 [PLM16] Adrian Pop, Florian Lapschies, and Oliver Möller. Test automation  
1661 module in the INTO-CPS Platform. Technical report, INTO-CPS  
1662 Deliverable, D5.2a, December 2016.
- 1663 [Pnu77] Amir Pnueli. The Temporal Logic of Programs. In *18th Symposi-  
1664 um on the Foundations of Computer Science*, pages 46–57. ACM,  
1665 November 1977.
- 1666 [Ver13] Verified Systems International GmbH. RTT-MBT Model-Based  
1667 Test Generator - RTT-MBT Version 9.0-1.0.0 User Manual. Tech-  
1668 nical Report Verified-INT-003-2012, Verified Systems International  
1669 GmbH, 2013. Available on request from Verified System Interna-  
1670 tional GmbH.
- 1671 [Ver15a] Verified Systems International GmbH, Bremen, Germany. *RT-  
1672 Tester 6.0: User Manual*, 2015. [https://www.verified.de/  
1673 products/rt-tester/](https://www.verified.de/products/rt-tester/), Doc. Id. Verified-INT-014-2003.
- 1674 [Ver15b] Verified Systems International GmbH, Bremen, Germany. *RT-  
1675 Tester Model-Based Test Case and Test Data Generator – RTT-  
1676 MBT: User Manual*, 2015. [https://www.verified.de/  
1677 products/model-based-testing/](https://www.verified.de/products/model-based-testing/), Doc. Id. Verified-INT-  
1678 003-2012.
- 1679 [Win16] Wine community. <https://www.winehq.org/>, November 2016. Wine  
1680 website.

1681 **A List of Acronyms**

20-sim	Software package for modelling and simulation of dynamic systems
API	Application Programming Interface
AST	Abstract Syntax Tree
AU	Aarhus University
BCS	Basic Control States
CLE	ClearSy
CLP	Controllab Products B.V.
COE	Co-simulation Orchestration Engine
CORBA	Common Object Request Broker Architecture
CPS	Cyber-Physical Systems
CT	Continuous-Time
DE	Discrete Event
DESTTECS	Design Support and Tooling for Embedded Control Software
DSE	Design Space Exploration
FMI	Functional Mockup Interface
FMI-Co	Functional Mockup Interface – for Co-simulation
FMI-ME	Functional Mockup Interface – Model Exchange
FMU	Functional Mockup Unit
HiL	Hardware-in-the-Loop
HMI	Human Machine Interface
HW	Hardware
ICT	Information Communication Technology
IDE	Integrated Design Environment
LTL	Linear Temporal Logic
M&S	Modelling and Simulation
MARTE	Modeling and Analysis of Real-Time and Embedded Systems
MBD	Model-based Design
MBT	Model-based Testing
MC/DC	Modified Decision/Condition Coverage
MDE	Model Driven Engineering
MiL	Model-in-the-Loop
MIWG	Model Interchange Working Group
OMG	Object Management Group
OS	Operating System
PID	Proportional Integral Derivative
PROV-N	The Provenance Notation
RPC	Remote Procedure Call
RTT	Real-Time Tester

SiL	Software-in-the Loop
SMT	Satisfiability Modulo Theories
ST	Softteam
SUT	System Under Test
SVN	Subversion
SysML	Systems Modelling Language
TA	Test Automation
TE	Test Environment
TR	TRansitions
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 Technologies Research Center
UY	University of York
VDM	Vienna Development Method
VSI	Verified Systems International
WP	Work Package
XML	Extensible Markup Language

## 1682 **B Background on the Individual Tools**

1683 This appendix provides background information on each of the independent  
1684 tools of the INTO-CPS tool chain.

### 1685 **B.1 Modelio**

1686 Modelio is a comprehensive MDE [Fav05] workbench tool which supports  
1687 the UML2.x standard. Modelio adds modern Eclipse-based graphical envi-  
1688 ronment to the solid modelling and generation know-how obtained with the  
1689 earlier Softeam MDE workbench, Objecteering, which has been on the mar-  
1690 ket since 1991. Modelio provides a central repository for the local model,  
1691 which allows various languages (UML profiles) to be combined in the same  
1692 model, abstraction layers to be managed and traceability between different  
1693 model elements to be established. Modelio makes use of extension modules,  
1694 enabling the customization of this MDE environment for different purposes  
1695 and stakeholders. The XMI module allows models to be exchanged between  
1696 different UML modelling tools. Modelio supports the most popular XMI  
1697 UML2 flavors, namely EMF UML2 and OMG UML 2.3. Modelio is one of  
1698 the leaders in the OMG Model Interchange Working Group (MIWG), due to  
1699 continuous work on XMI exchange improvements.

1700 Among the extension modules, some are dedicated to IT system architects.  
1701 For system engineering, SysML or MARTE modules can be used. They  
1702 provide dedicated modelling support for dealing with general, software and  
1703 hardware aspects of embedded or cyber physical systems. In addition, sev-  
1704 eral utility modules are available, such as the Document Publisher which  
1705 provides comprehensive support for the generation of different types of doc-  
1706 ument.

1707 Modelio is highly extendable and can be used as a platform for building  
1708 new MDE features. The tool enables users to build UML2 Profiles, and to  
1709 combine them with a rich graphical interface for dedicated diagrams, model  
1710 element property editors and action command controls. Users can use several  
1711 extension mechanisms: light Python scripts or a rich Java API, both of which  
1712 provide access to Modelio's model repository and graphical interface.

## 1713 B.2 Overture

1714 The Overture platform [LBF<sup>+</sup>10] is an Eclipse-based integrated development  
1715 environment (IDE) for the development and validation of system specifica-  
1716 tions in three dialects of the specification language of the Vienna Develop-  
1717 ment Method. Overture is distributed with a suite of examples and step-by-  
1718 step tutorials which demonstrate the features of the three dialects. A user  
1719 manual for the platform itself is also provided [LLJ<sup>+</sup>13], which is accessible  
1720 through Overture's help system. Although certain features of Overture are  
1721 relevant only to the development of software systems, VDM itself can be used  
1722 for the specification and validation of any system with distinct states, known  
1723 as *discrete-event systems*, such as physical plants, protocols, controllers (both  
1724 mechanical and software) *etc.*, and Overture can be used to aid in validation  
1725 activities in each case.

1726 Overture supports the following activities:

- 1727 • The definition and elaboration of syntactically correct specifications in  
1728 any of the three dialects, via automatic syntax and type validation.
- 1729 • The inspection and assay of automatically generated proof obligations  
1730 which ensure correctness in those aspects of specification validation  
1731 which can not be automated.
- 1732 • Direct interaction with a specification via an execution engine which  
1733 can be used on those elements of the specification written in an exe-  
1734 cutable subset of the language.
- 1735 • Automated testing of specifications via a custom test suite definition  
1736 language and execution engine.
- 1737 • Visualization of test coverage information gathered from automated  
1738 testing.
- 1739 • Visualization of timing behaviours for specifications incorporating tim-  
1740 ing information.
- 1741 • Translation to/from UML system representations.
- 1742 • For specifications written in the special executable subset of the lan-  
1743 guage, obtaining Java implementations of the specified system auto-  
1744 matically.

1745 For more information and tutorials, please refer to the documentation dis-  
1746 tributed with Overture.

1747 The following is a brief introduction to the features of the three dialects of  
1748 the VDM specification language.

1749 **VDM-SL** This is the foundation of the other two dialects. It supports the  
1750 development of monolithic state-based specifications with state transition  
1751 operations. Central to a VDM-SL specification is a definition of the state  
1752 of the system under development. The meaning of the system and how it  
1753 operates is conveyed by means of changes to the state. The nature of the  
1754 changes is captured by state-modifying operations. These may make use of  
1755 auxiliary functions which do not modify state. The language has the usual  
1756 provisions for arithmetic, new dependent types, invariants, pre- and post-  
1757 conditions *etc.* Examples can be found in the VDM-SL tutorials distributed  
1758 with Overture.

1759 **VDM++** The VDM++ dialect supports a specification style inspired by  
1760 object-oriented programming. In this specification paradigm, a system is  
1761 understood as being composed of entities which encapsulate both state and  
1762 behaviour, and which interact with each other. Entities are defined via tem-  
1763 plates known as *classes*. A complete system is defined by specifying *instances*  
1764 of the various classes. The instances are independent of each other, and they  
1765 may or may not interact with other instances. As in object-oriented program-  
1766 ming, the ability of one component to act directly on any other is specified  
1767 in the corresponding class as a state element. Interaction is naturally carried  
1768 out via precisely defined interfaces. Usually a single class is defined which  
1769 represents the entire system, and it has one instance, but this is only a con-  
1770 vention. This class may have additional state elements of its own. Whereas a  
1771 system in VDM-SL has a central state which is modified throughout the life-  
1772 time of the system, the state of a VDM++ system is distributed among all of  
1773 its components. Examples can be found in the VDM++ tutorials distributed  
1774 with Overture.

1775 **VDM-RT** VDM-RT is a small extension to VDM++ which adds two pri-  
1776 mary features:

- 1777 • The ability to define how the specified system is envisioned to be allo-  
1778 cated on a distributed execution platform, together with the commu-  
1779 nication topology.
- 1780 • The ability to specify the timing behaviours of individual components,  
1781 as well as whether certain behaviours are meant to be cyclical.

1782 Finer details can be specified, such as execution synchronization and mutual  
 1783 exclusion on shared resources. A VDM-RT specification has the same  
 1784 structure as a VDM++ specification, only the conventional system class of  
 1785 VDM++ is mandatory in VDM-RT. Examples can be found in the VDM-RT  
 1786 tutorials distributed with Overture.

### 1787 B.3 20-sim

1788 20-sim [Con13, Bro97] is a commercial modelling and simulation software  
 1789 package for mechatronic systems. With 20-sim, models can be created graphically,  
 1790 similar to drawing an engineering scheme. With these models, the behaviour of  
 1791 dynamic systems can be analyzed and control systems can be designed.  
 1792 20-sim models can be exported as C-code to be run on hardware for rapid  
 1793 prototyping and HiL-simulation. 20-sim includes tools that allow an engineer  
 1794 to create models quickly and intuitively. Models can be created using equations,  
 1795 block diagrams, physical components and bond graphs [KR68]. Various tools give  
 1796 support during the model building and simulation. Other toolboxes help to  
 1797 analyze models, build control systems and improve system performance. Figure 104  
 shows 20-sim with a model of a controlled

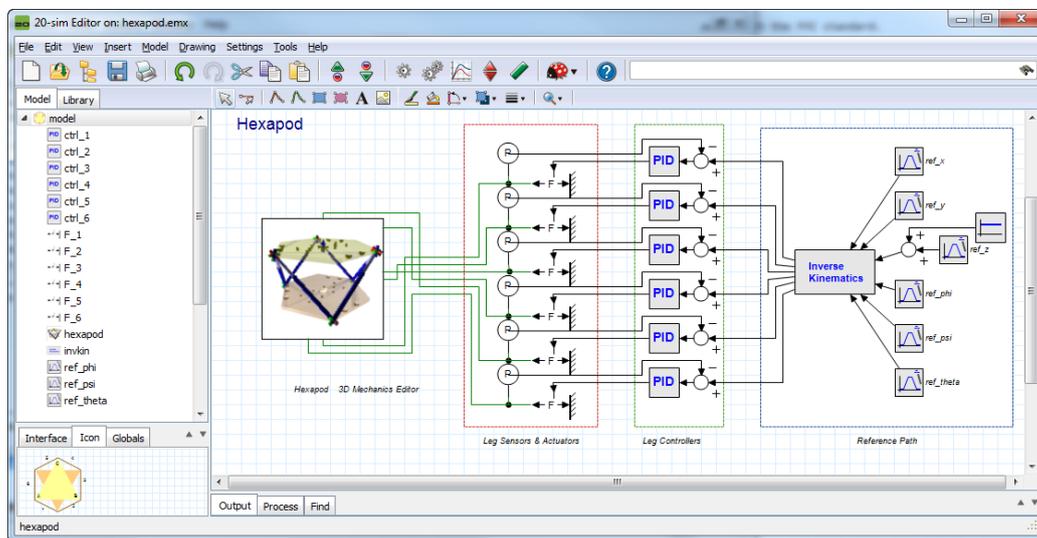


Figure 104: Example of a hexapod model in 20-sim.

1798 hexapod. The mechanism is generated with the 3D Mechanics Toolbox and  
 1799 connected with standard actuator and sensor models from the mechanics library.  
 1800 The hexapod is controlled by PID controllers which are tuned in the  
 1801

1802 frequency domain. Everything that is required to build and simulate this  
1803 model and generate the controller code for the real system is included inside  
1804 the package.

1805 The 20-sim Getting Started manual [KG16] contains examples and step-by-  
1806 step tutorials that demonstrate the features of 20-sim. More information on  
1807 20-sim can be found at <http://www.20sim.com> and in the user manual  
1808 at <http://www.20sim.com/webhelp> [KGD16]. The integration of 20-  
1809 sim into the INTO-CPS tool-chain is realized via the FMI standard.

## 1810 **B.4 OpenModelica**

1811 OpenModelica [Fri04] is an open-source Modelica-based modelling and sim-  
1812 ulation environment. Modelica [FE98] is an object-oriented, equation based  
1813 language to conveniently model complex physical systems containing, e.g.,  
1814 mechanical, electrical, electronic, hydraulic, thermal, control, electric power  
1815 or process-oriented subcomponents. The Modelica language (and OpenMod-  
1816 elica) supports continuous, discrete and hybrid time simulations. OpenMod-  
1817 elica already compiles Modelica models into FMU, C or C++ code for simula-  
1818 tion. Several integration solvers, both fixed and variable step size, are avail-  
1819 able in OpenModelica: euler, rungekutta, dassl (default), radau5, radau3,  
1820 radau1.

1821 OpenModelica can be interfaced to other tools in several ways as described  
1822 in the OpenModelica user's manual [Ope]:

- 1823 • via command line invocation of the omc compiler
- 1824 • via C API calls to the omc compiler dynamic library
- 1825 • via the CORBA interface
- 1826 • via OMPython interface [GFR<sup>+</sup>12]

1827 OpenModelica has its own scripting language, Modelica script (mos files),  
1828 which can be used to perform actions via the compiler API, such as load-  
1829 ing, compilation, simulation of models or plotting of results. OpenModelica  
1830 supports Windows, Linux and Mac Os X.

1831 The integration of OpenModelica into the INTO-CPS tool chain is realized  
1832 via compliance with the FMI standard, and is described in deliverable D4.1b  
1833 [PBLG15].

## 1834 B.5 RT-Tester

1835 The RT-Tester [Ver15a] is a test automation tool for automatic test gener-  
1836 ation, test execution and real-time test evaluation. Key features include a  
1837 strong C/C++-based test script language, high performance multi-threading,  
1838 and hard real-time capability. The tool has been successfully applied in avionics,  
1839 rail automation, and automotive test projects. In the INTO-CPS tool  
1840 chain, RT-Tester is responsible for model-based testing, as well as for model  
1841 checking. This section gives some background information on the tool from  
1842 these two perspectives.

### 1843 B.5.1 Model-based Testing

1844 The RT-Tester Model Based Test Case and Test Data Generator (RTT-  
1845 MBT) [Ver15b] supports model-based testing (MBT), that is, automated  
1846 generation of test cases, test data, and test procedures from UML/SysML  
1847 models. A number of common modelling tools can be used as front-ends for  
1848 this. The most important technical challenge in model-based test automation  
1849 is the extraction of test cases from test models. RTT-MBT combines an SMT  
1850 solver with a technique akin to bounded model checking so as to extract finite  
1851 paths through the test model according to some predefined criterion. This  
1852 criterion can, for instance, be MC/DC coverage, or it can be requirements  
1853 coverage (if the requirements are specified as temporal logic formulae within  
1854 the model). A further aspect is that the environment can be modelled within  
1855 the test model. For example, the test model may contain a constraint such  
1856 that a certain input to the system-under-test remains in a predefined range.  
1857 This aspect becomes important once test automation is lifted from single test  
1858 models to multi-model cyber-physical systems. The derived test procedures  
1859 use the RT-Tester Core as a back-end, allowing the system under test to be  
1860 provided on real hardware, software only, or even just simulation to aid test  
1861 model development.

1862 Further, RTT-MBT includes requirement tracing from test models down to  
1863 test executions and allows for powerful status reporting in large scale testing  
1864 projects.

### 1865 B.5.2 Model Checking of Timed State Charts

1866 RTT-MBT applies model checking to behavioural models that are specified  
1867 as timed state charts in UML and SysML, respectively. From these models,

1868 a transition relation is extracted and represented as an SMT formula in bit-  
 1869 vector theory [KS08], which is then checked against LTL formulae [Pnu77]  
 1870 using the algorithm of Biere *et al.* [BHJ<sup>+</sup>06]. The standard setting of RTT-  
 1871 MBT is to apply model checking to a single test model, which consists of the  
 1872 system specification and an environment.

- 1873 • A component called *TestModel* that is annotated with stereotype *TE*.
- 1874 • A component called *SystemUnderTest* that is annotated with stereo-  
 1875 type *SUT*.

1876 RTT-MBT uses the stereotypes to infer the role of each component. The in-  
 1877 teraction between these two parts is implemented via input and output inter-  
 1878 faces that specify the accessibility of variables using UML stereotypes.

- 1879 • A variable that is annotated with stereotype *SUT2TE* is written by  
 1880 the system model and readable by the environment.
- 1881 • A variable that is annotated with stereotype *TE2SUT* is written by  
 1882 the environment and read by the system model as an input.

1883 A simple example is depicted in Figure 105, which shows a simple composite  
 1884 structure diagram in Modelio for a turn indication system. The purpose  
 1885 of the system is to control the lamps of a turn indication system in a car.  
 1886 Further details are given in [Ver13]. The test model consists of the two  
 1887 aforementioned components and two interfaces:

- 1888 • **Interface1** is annotated with stereotype *TE2SUT* and contains three  
 1889 variables `voltage`, `TurnIndLvr` and `EmerSwitch`. These variables  
 1890 are controlled by the environment and fed to the system under test as  
 1891 inputs.
- 1892 • **Interface2** is annotated with stereotype *SUT2TE* and contains two  
 1893 variables `LampsLeft` and `LampsRight`. These variables are con-  
 1894 trolled by the system under test and can be read by the environment.

Observe that the two variables `LampsLeft` and `LampsRight` have type `int`, but should only hold values 0 or 1 to indicate states *on* or *off*. A straightforward system property that could be verified would thus be that `LampsLeft` and `LampsRight` indeed are only assigned 0 or 1, which could be expressed by the following LTL specification:

$$\mathbf{G}(0 \leq \text{LampsLeft} \leq 1 \wedge 0 \leq \text{LampsRight} \leq 1)$$

1895 A thorough introduction with more details is given in the RTT-MBT user  
 1896 manual [Ver13].

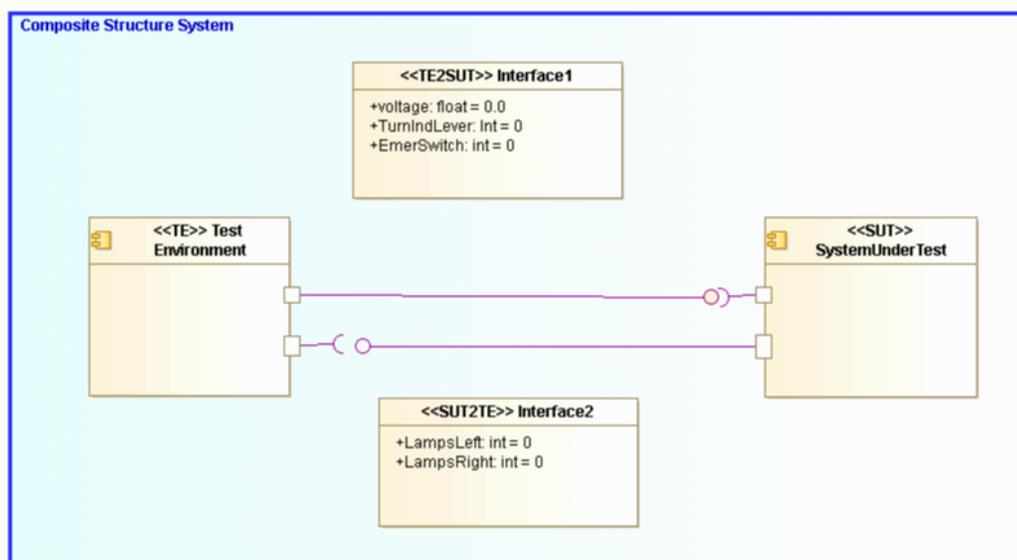


Figure 105: Simple model that highlights interfaces between the environment and the system-under-test.

## 1897 C Underlying Principles

1898 The INTO-CPS tool chain facilitates the design and validation of CPSs  
1899 through its implementation of results from a number of underlying principles.  
1900 These principles are co-simulation, design space exploration, model-based  
1901 test automation and code generation. This appendix provides an introduc-  
1902 tion to these concepts.

### 1903 C.1 Co-simulation

1904 Co-simulation refers to the simultaneous simulation of individual models  
1905 which together make up a larger system of interest, for the purpose of ob-  
1906 taining a simulation of the larger system. A co-simulation is performed by a  
1907 co-simulation orchestration engine. This engine is responsible for initializing  
1908 the individual simulations as needed; for selecting correct time step sizes such  
1909 that each constituent model can be simulated successfully for that duration,  
1910 thus preventing time drift between the constituent simulations; for asking  
1911 each individual simulation to perform a simulation step; and for synchron-  
1912 izing information between models as needed after each step. The result of  
1913 one such round of simulations is a single simulation step for the complete  
1914 multi-model of the system of interest.

1915 As an example, consider a very abstract model of a nuclear power plant. This  
1916 consists of a nuclear reactor core, a controller for the reactor, a water and  
1917 steam distribution system, a steam-driven turbine and a standard electrical  
1918 generator. All these individual components can be modelled separately and  
1919 simulated, but when composed into a model of a nuclear power plant, the  
1920 outputs of some become the inputs of others. In a co-simulation, outputs  
1921 are matched to inputs and each component is simulated one step at a time  
1922 in such a way that when each model has performed its simulation step, the  
1923 overall result is a simulation step of the complete power plant model. Once  
1924 the correct information is exchanged between the constituent models, the  
1925 process repeats.

### 1926 C.2 Design Space Exploration

1927 During the process of developing a CPS, either starting from a completely  
1928 blank canvas or constructing a new system from models of existing compo-  
1929 nents, the architects will encounter many design decisions that shape the

1930 final product. The activity of investigating and gathering data about the  
1931 merits of the different choices available is termed Design Space Exploration.  
1932 Some of the choices the designer will face could be described as being the  
1933 selection of parameters for specific components of the design, such as the  
1934 exact position of a sensor, the diameter of wheels or the parameters affecting  
1935 a control algorithm. Such parameters are variable to some degree and the  
1936 selection of their value will affect the values of objectives by which a design  
1937 will be measured. In these cases it is desirable to explore the different values  
1938 each parameter may take and also different combinations of these parameter  
1939 values if there are more than one parameter, to find a set of designs that best  
1940 meets its objectives. However, since the size of the design space is the prod-  
1941 uct of the number of parameters and the number of values each may adopt,  
1942 it is often impractical to consider performing simulations of all parameter  
1943 combinations or to manually assess each design.

1944 The purpose of an automated DSE tool is to help manage the exploration  
1945 of the design space, and it separates this problem into three distinct parts:  
1946 the search algorithm, obtaining objective values and ranking the designs  
1947 according to those objectives. The simplest of all search algorithms is the  
1948 exhaustive search, and this algorithm will methodically move through each  
1949 design, performing a simulation using each and every one. This is termed  
1950 an open loop method, as the simulation results are not considered by the  
1951 algorithm at all. Other algorithms, such as a genetic search, where an initial  
1952 set of randomly generated individuals are bred to produce increasingly good  
1953 results, are closed loop methods. This means that the choice of next design  
1954 to be simulated is driven by the results of previous simulations.

1955 Once a simulation has been performed, there are two steps required to close  
1956 the loop. The first is to analyze the raw results output by the simulation to  
1957 determine the value for each of the objectives by which the simulations are  
1958 to be judged. Such objective values could simply be the maximum power  
1959 consumed by a component or the total distance traveled by an object, but  
1960 they could also be more complex measures, such as the proportion of time  
1961 a device was operating in the correct mode given some conditions. As well  
1962 as numerical objectives, there can also be constraints on the system that  
1963 are either passed or failed. Such constraints could be numeric, such as the  
1964 maximum power that a substation must never exceed, or they could be based  
1965 on temporal logic to check that undesirable events do not occur, such as all  
1966 the lights at a road junction not being green at the same time.

1967 The final step in a closed loop is to rank the designs according to how well  
1968 each performs. The ranking may be trivial, such as in a search for a design

1969 that minimizes the total amount of energy used, or it may be more complex  
1970 if there are multiple objectives to optimize and trade off. Such ranking  
1971 functions can take the form of an equation that returns a score for each  
1972 design, where the designs with the highest/lowest scores are considered the  
1973 best. Alternatively, if the relationship between the desired objectives is not  
1974 well understood, then a Pareto approach can be taken to ranking, where  
1975 designs are allocated to ranks of designs that are indistinguishable from each  
1976 other, in that each represents an optimum, but there exist different tradeoffs  
1977 between the objective values.

### 1978 C.3 Model-Based Test Automation

1979 The core fragment of test automation activities is a model of the desired  
1980 system behaviour, which can be expressed in SysML. This test model in-  
1981 duces a transition relation, which describes a collection of execution paths  
1982 through the system, where a path is considered a sequence of timed data  
1983 vectors (containing internal data, inputs and outputs). The purpose of a test  
1984 automation tool is to extract a subset of these paths from the test model  
1985 and turn these paths into test cases, respectively test procedures. The test  
1986 procedures then compare the behaviour of the actual system-under-test to  
1987 the path, and produce warnings once discrepancies are observed.

### 1988 C.4 Code Generation

1989 Code generation refers to the translation of a modelling language to a com-  
1990 mon programming language. Code generation is commonly employed in con-  
1991 trol engineering, where a controller is modelled and validated using a tool  
1992 such as 20-sim, and finally translated into source code to be compiled for  
1993 some embedded execution platform, which is its final destination.

1994 The relationship that must be maintained between the source model and  
1995 translated program must be one of refinement, in the sense that the trans-  
1996 lated program must not do anything that is not captured by the original  
1997 model. This must be considered when translating models written in high-  
1998 level specification languages, such as VDM. The purpose of such languages  
1999 is to allow the specification of several equivalent implementations. When  
2000 a model written in such a language is translated to code, one such imple-  
2001 mentation is essentially chosen. In the process, any non-determinism in the  
2002 specification, the specification technique that allows a choice of implemen-

2003 tations, must be resolved. Usually this choice is made very simple by re-  
2004 stricting the modelling language to an executable subset, such that no such  
2005 non-determinism is allowed in the model. This restricts the choice of imple-  
2006 mentations to very few, often one, which is the one into which the model is  
2007 translated via code generation.