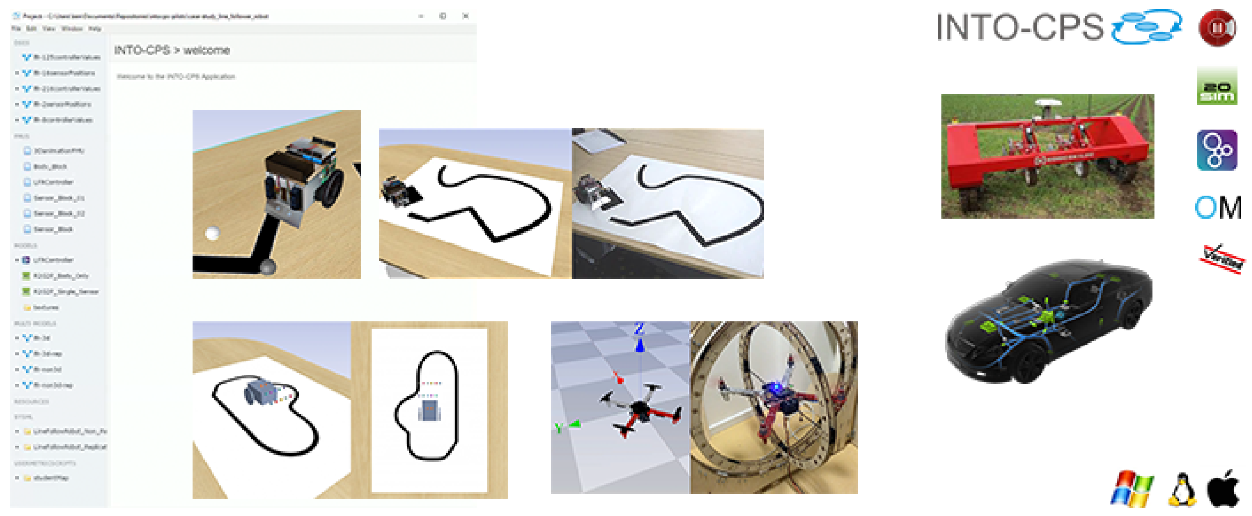
INTO-CPS

Oct 21, 2020

Content

chat on gitter

INTO-CPS is a collection of tools developed to aid the development of Cyber-Physical Systems (CPSs). Typical for such systems is a large number of complex components of different nature, such as software controllers, mechanical parts and electrical circuits. Simulating such system requires special algorithms due to the different formalisms used to model the individual components. This is possible using a simulation technique known as co-simulation, enabled by the use of a model exchange standard called Functional Mock-up Interface (FMI) standard.



Specifically, INTO-CPS provides all the necessary tools for this such as:

- Modelling the individual components referred to as a Functional Mock-up Unit (FMU)
- Performing a co-simulation
- Evaluating the impact of different design decisions
- Validating and testing FMUs
- Coupling the simulation with real-life prototypes

A full list of the tools and how to obtain them can be found in *Tools* and additional information is available in the following documents:

`The INTO-CPS Manifesto`
`The INTO-CPS User Manual`
`The INTO-CPS Examples Compendium`
`The INTO-CPS Method Guidelines`

For a demonstration of the tools please check the INTO-CPS Association's YouTube.

The *Tutorials* demonstrates how the different tools can be used together to model and simulate a various types of systems.

In order to use the tools effectively it may be worthwhile to read some theory on the technologies involved. This information can be found in the *Guides* section, where topics such as co-simulation and FMI are covered.

CHAPTER 1

Tools

The INTO-CPS toolchain consists of several tools for developing Cyber-Physical Systems. These tools provide functionality needed to enable FMI based co-simulation namely as *Creating new FMUs*, *verify* and *Performing Co-Simulations* the simulation of FMUs. To keep the infrastructure as lightweight and flexible as possible the projects are hosted independently in their own repositories and provide standalone documentation.

A list of the tools and, grouped by the functionality they provide, can be found below:

1.1 Performing Co-Simulations

1.1.1 INTO-CPS Desktop Application

The INTO-CPS Desktop Application provides a graphical user interface which can be used to configure and orchestrate co-simulation scenarios. Install instruction are found in the application's documentation.

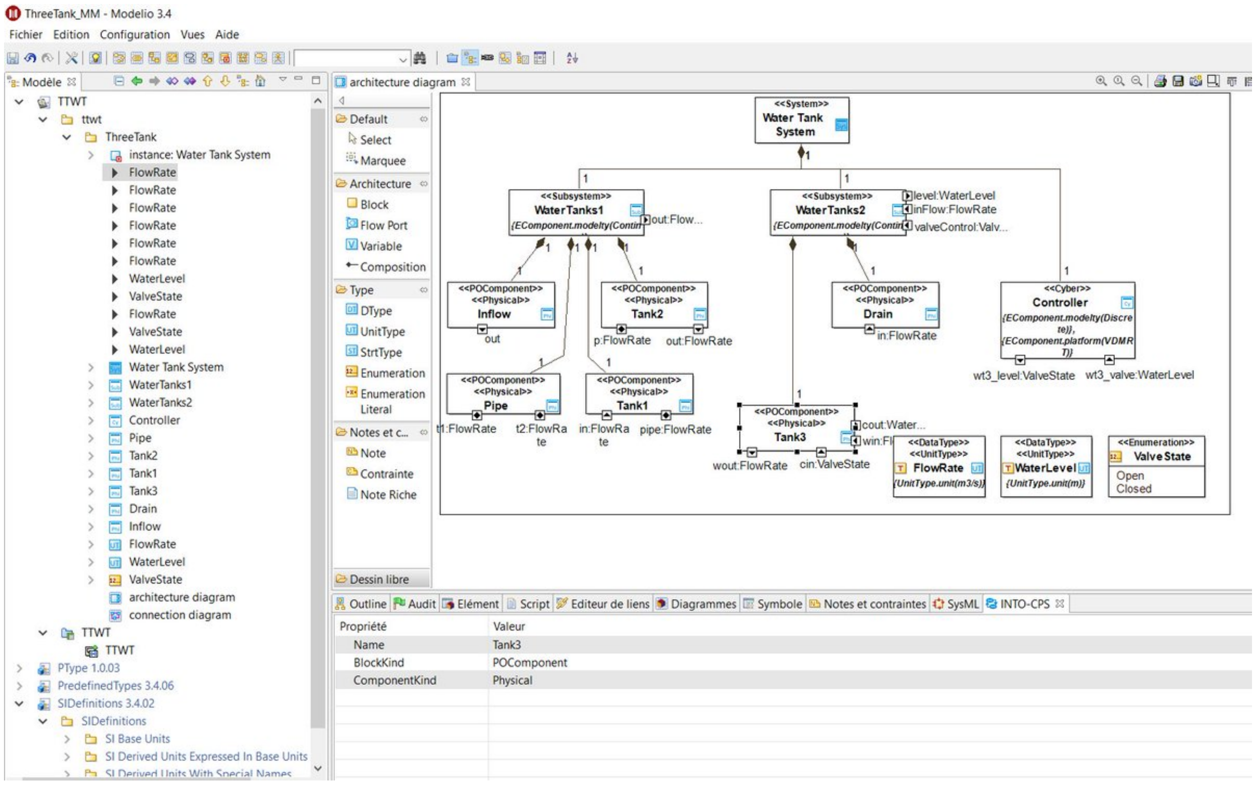
1.1.2 INTO-CPS Cloud Application

The INTO-CPS Cloud Application is a port of the Desktop Application to run in a cloud environment.

1.1.3 Modelio

Modelio is a combined UML/BPMN modeler supporting a wide range of models and diagrams. Its main features are:

- SysML support integrated with UML and BPMN
- XMI import and export
- Scripting language support (Jython)
- Extensibility: Modelio can be extended for any language, methodology or modeling technique just by adding modules. You can either use existing modules or else develop your own.


ThreeTank_MM - Modelio 3.4
Fichier Edition Configuration Vues Aide
Modèle
TTWT
ttwt
ThreeTank
instance: Water Tank System
FlowRate
WaterLevel
ValveState
Water Tank System
WaterTanks1
WaterTanks2
Controller
Pipe
Tank2
Tank1
Tank3
Drain
Inflow
architecture diagram
connection diagram
PType 1.0.03
PredefinedTypes 3.4.06
SIDefinitions 3.4.02
SIDefinitions
SI Base Units
SI Derived Units Expressed In Base Units
SI Derived Units With Special Names
Default
Select
Marquee
Architecture
Block
Flow Port
Variable
Composition
Type
DType
UnitType
StrtType
Enumeration
Enumeration Literal
Notes et c...
Note
Contrainte
Note Riche
Dessin libre
<<System>> Water Tank System
<<Subsystem>> WaterTanks1 {EComponent.modelty(Contin
out:Flow...
<<Subsystem>> WaterTanks2 {EComponent.modelty(Contin
level:WaterLevel
inFlow:FlowRate
valveControl:Valv...
<<POComponent>> <<Physical>> Inflow
out
<<POComponent>> <<Physical>> Tank2
p:FlowRate
out:FlowRate
<<POComponent>> <<Physical>> Drain
in:FlowRate
<<Cyber>> Controller {EComponent.modelty(Discrete)}, {EComponent.platform(VDMRT)}
wt3_level:ValveState
wt3_valve:WaterLevel
<<POComponent>> <<Physical>> Pipe
t1:FlowRate
t2:FlowRate
<<POComponent>> <<Physical>> Tank1
in:FlowRate
pipe:FlowRate
<<POComponent>> <<Physical>> Tank3
cout:Water...
win:Fl
wout:FlowRate
cin:ValveState
<<DataType>> <<UnitType>> FlowRate {UnitType.unit(m3/s)}
<<DataType>> <<UnitType>> WaterLevel {UnitType.unit(m)}
<<Enumeration>> ValveState
Open
Closed
Outline Audit Elément Script Editeur de liens Diagrammes Symbole Notes et contraintes SysML INTO-CPS

Propriété	Valeur
Name	Tank3
BlockKind	POComponent
ComponentKind	Physical

In addition, to its general purpose use, a special SysML profile allows Modelio to be used to setup co-simulation scenarios that can be executed within the INTO-CPS Desktop Application. For information on how to use this integration see Desktop Application Modelio integration

License

Modelio is **open source** software. Most of the source code is under the **GNU GPL** license. The module runtime, used to develop extensions to Modelio, is under the **Apache license** providing a very large degree of freedom to anyone wishing to reuse and embed the code.

Full details on Modelio licensing conditions can be found here.

Download and Installation

Modelio's current version is 3.4.0.

Binary distribution archives for Linux and Windows are available on the download page of the Modelio community site.

Installation requirements and instructions can be found in our Quick start guide on the community site.

Modelio version contents can be found in the [[Version history]].

Installing INTO-CPS support

A specific project have been created into Modelio.org forge http://forge.modelio.org/projects/intocps.

This latter will help you with:

1. A dedicated wiki http://forge.modelio.org/projects/into-cps-user-manual-english/wiki

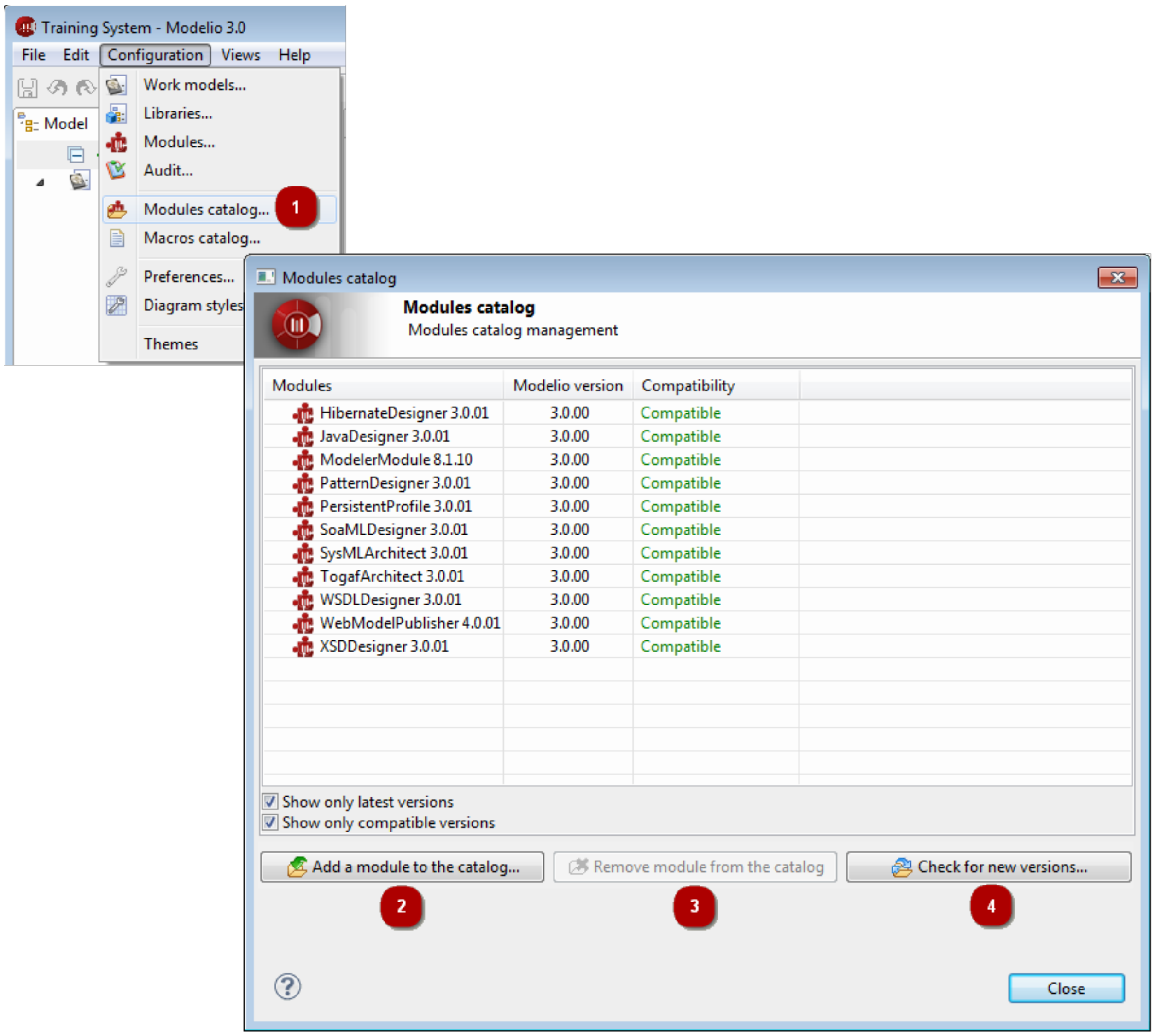
2. Lattest version of INTO-CPS extension http://forge.modelio.org/projects/intocps-modelio34/files
3. Issues reporting http://forge.modelio.org/projects/intocps/issues/new

To be able to use INTO-CPS extension into Modelio you have to first install the INTO-CPS module which is available here:

http://forge.modelio.org/projects/intocps-modelio34/files.

and then add it to the Modelio module catalog.

Add INTO-CPS Extension


Training System - Modelio 3.0
File
Edit
Configuration
Views
Help
Model
Work models...
Libraries...
Modules...
Audit...
Modules catalog...
1
Macros catalog...
Preferences...
Diagram styles
Themes
Modules catalog
Modules catalog
Modules catalog management
Show only latest versions
Show only compatible versions
Add a module to the catalog...
Remove module from the catalog
Check for new versions...
2
3
4
Close

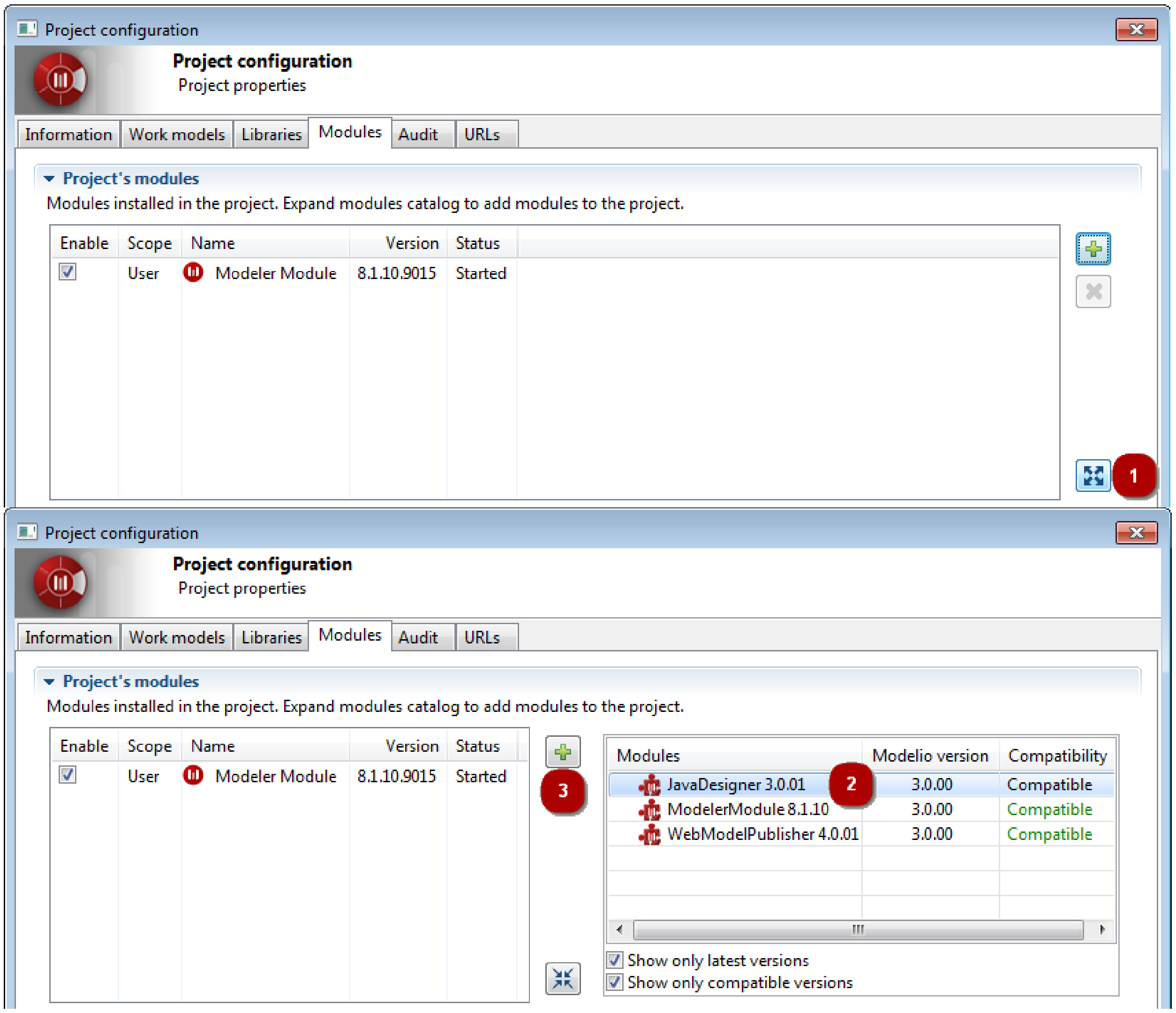
Modules	Modelio version	Compatibility
HibernateDesigner 3.0.01	3.0.00	Compatible
JavaDesigner 3.0.01	3.0.00	Compatible
ModelerModule 8.1.10	3.0.00	Compatible
PatternDesigner 3.0.01	3.0.00	Compatible
PersistentProfile 3.0.01	3.0.00	Compatible
SoaMLDesigner 3.0.01	3.0.00	Compatible
SysMLArchitect 3.0.01	3.0.00	Compatible
TogafArchitect 3.0.01	3.0.00	Compatible
WSDLDesigner 3.0.01	3.0.00	Compatible
WebModelPublisher 4.0.01	3.0.00	Compatible
XSDDesigner 3.0.01	3.0.00	Compatible

Key:

1. Run the **Configuration** **Modules catalog...** command.
2. To add a module, click on **Add a module to the catalog...** and use the file browser to select the modules (*.jmdac files).

3. To remove a module, select the module in question and click on the **Remove module from the catalog** button.
4. To download new versions of modules into the catalog, click on **Check for new versions...**.

Installing modules in a project


Project configuration

Project configuration
Project properties

Information | Work models | Libraries | Modules | Audit | URLs

Project's modules
Modules installed in the project. Expand modules catalog to add modules to the project.

Enable	Scope	Name	Version	Status
✓	User	Modeler Module	8.1.10.9015	Started

Modules	Modelio version	Compatibility
JavaDesigner 3.0.01	3.0.00	Compatible
ModelerModule 8.1.10	3.0.00	Compatible
WebModelPublisher 4.0.01	3.0.00	Compatible

✓ Show only latest versions
✓ Show only compatible versions

Installing a module in a project

Steps:

1. Click on [ ] to expand the Modules catalog.
2. In the Modules catalog, select the module you want to install.
3. Click on [ ] to install the module in the project..

Documentation

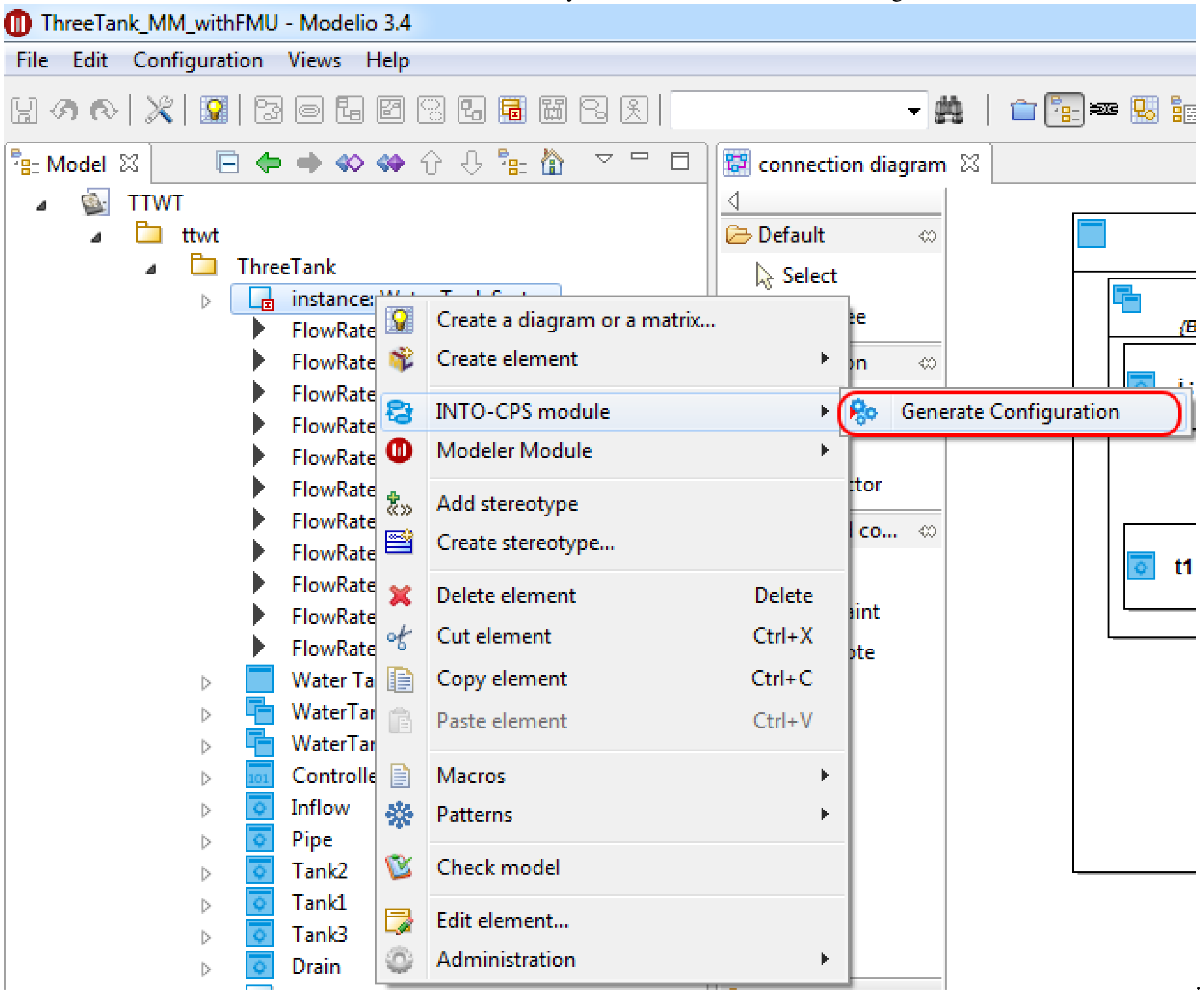
For end users

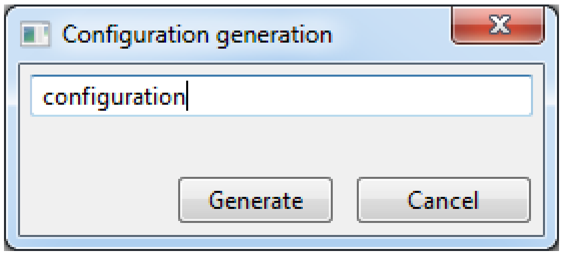
- User's manual *(wiki)*
- Installing a module in to Modelio *(wiki)*
- Configuring Modelio project modules *(wiki)*

Connection Export

- How to export the configuration of a BlockInstance.

To generate a configuration from a BlockInstance, select the desired blockInstance, right click on it and in the INTO-CPS entry choose Generate configuration, as shown in



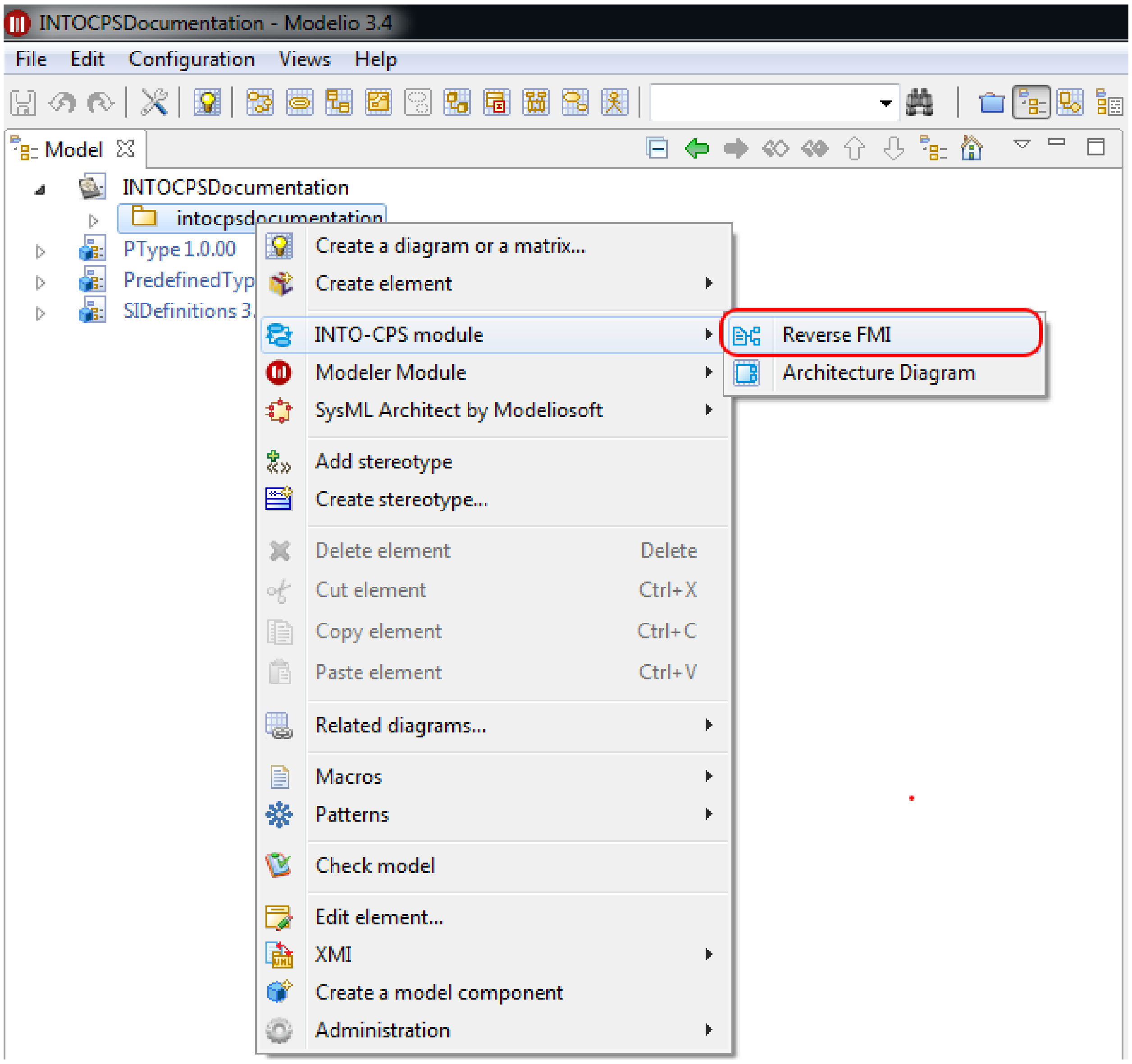
Choose a relevant name  and click on Generate.

FMU Import/Export

Two commands have been implemented in order to link SysML modelling and FMI definition i.e. the modeldescription.xml file defined inside the FMI 2.0 Specification.

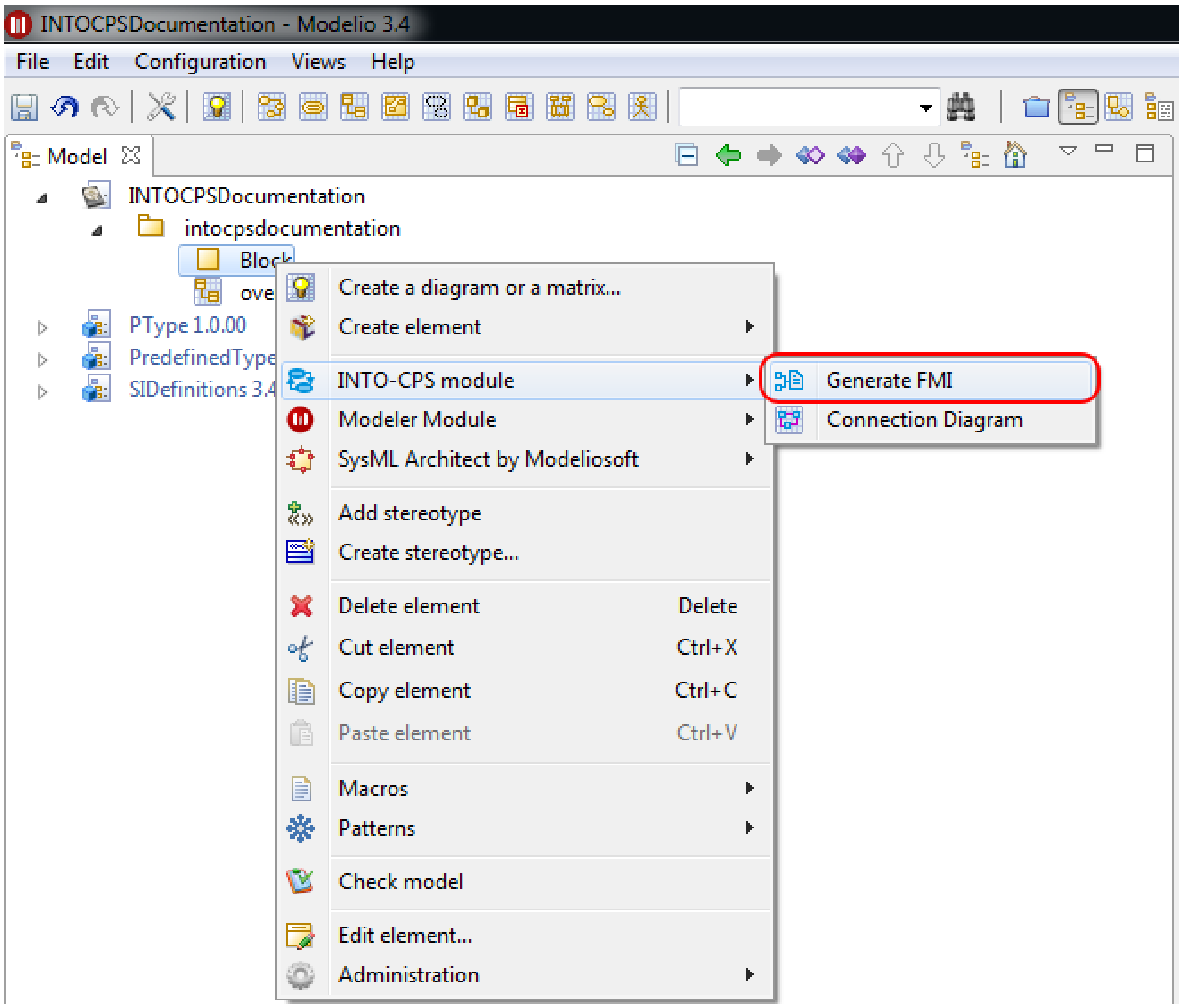
FMI Import

- This command imports the *modelDescription.xml* document describing the interface of a FMI and created an corresponding SysML Block under Modelio.
- Importing is easily done via the right click content menu, under a Package, as shown on the screenshot below.



FMI Export

- This only exports the *modelDescription.xml* document describing the interface of a FMI and not its behavior.
- Exporting is easily done via the right click content menu, under a SysML Block, as shown on the screenshot below.



1.1.4 Maestro

Maestro is a tool for orchestrating co-simulation of FMUs. It features both a command line interface and a web interface. The web interface is used internally by the INTO-CPS Desktop Application, but it can also be downloaded and used seperately.

All development efforts related to Maestro is available at https://github.com/INTO-CPS-Association/maestro.
Maestro1 Documentation is available here: https://into-cps-maestro.readthedocs.io/en/docs/
Maestro2 is currently under development and available at the branch 2.0.0-alpha with documentation here: https://into-cps-maestro.readthedocs.io/en/latest/.

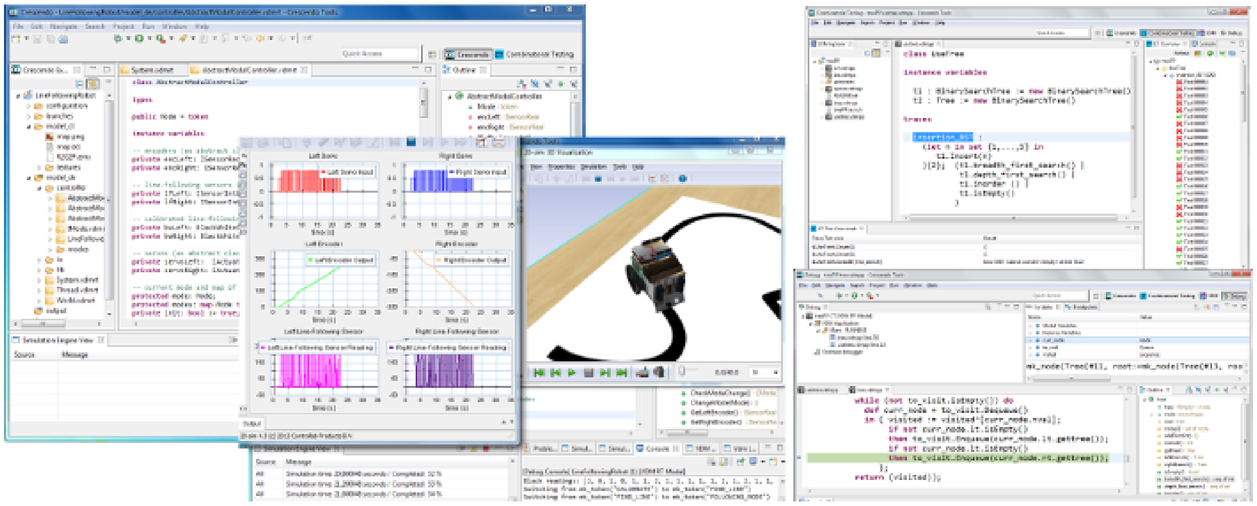
1.2 Creating new FMUs

Several commercial and open-source tools exist, that are enables the creation of FMUs. An comprehensive list of these can be found in the tools section on the FMI-standard's website.

Below is a list of tools that are part of INTO-CPS that supports FMI export.

1.2.1 Overture

The Overture community supports the modelling method The Vienna Development Method (VDM) which is a set of modelling techniques that have a long and successful history in both research and industrial application in the development of computer-based systems. The Overture Tool is an open-source integrated development environment (IDE) for developing and analysing VDM models. The tool suite is written entirely in Java and built on top of the Eclipse platform.



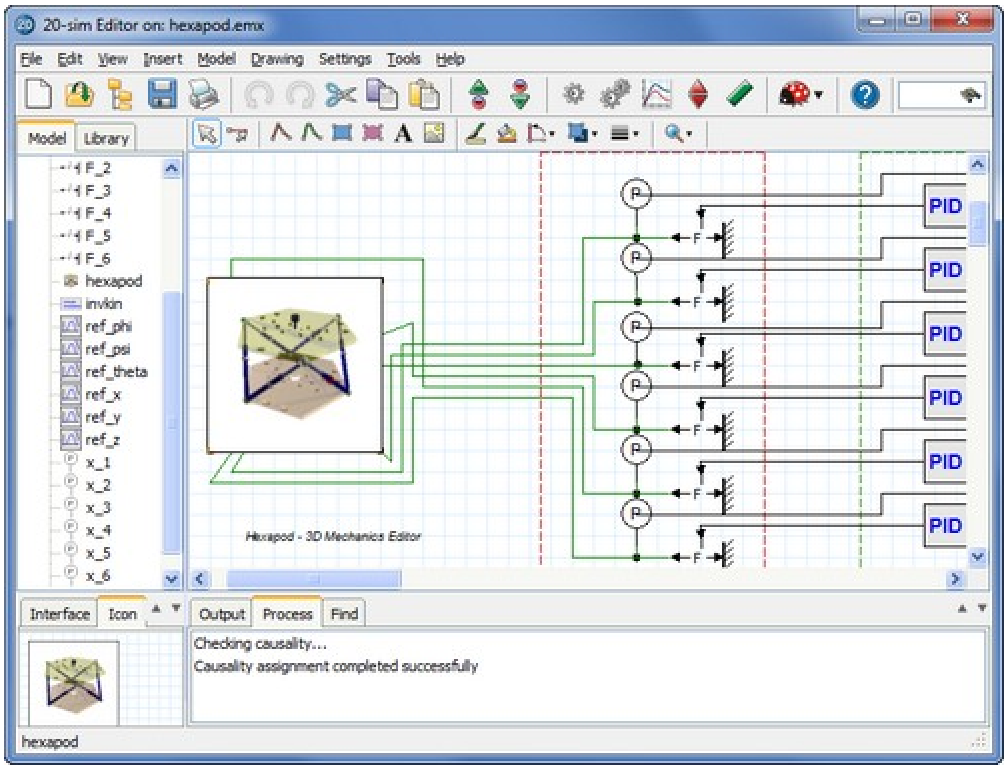
By installing a plugin, Overture can export its models as plain C-code and as standalone FMUs that can be imported into a FMI compatible simulation tool such as the INTO-CPS Desktop Application.

1.2.2 PyFMU

PyFMU is a command line program that enables the rapid development of FMUs using Python.

1.2.3 20-sim

20-sim is a modeling and simulation program for mechatronic systems. With 20-sim you can enter model graphically, similar to drawing an engineering scheme. With these models you can simulate and analyze the behavior of multi-domain dynamic systems and create control systems. You can even generate C-code and run this code on hardware for rapid prototyping and HIL-simulation.



Checking causality...
Causality assignment completed successfully

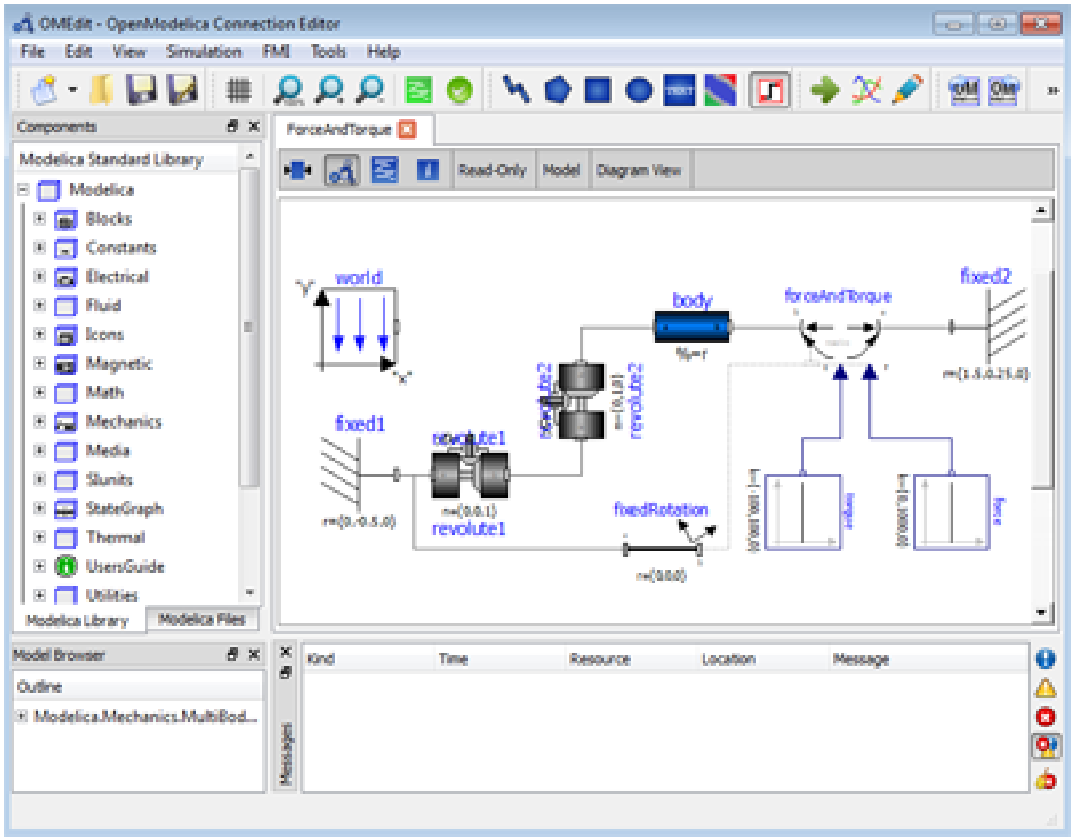
FMI support for INTO-CPS is provided using a code generation template.

Note: For 20-sim 4.5, this template can be downloaded from our GitHub repository and the installation instructions can be found in the included README file. Starting with 20-sim 4.6, the template is available out of the box.

Instructions on how to set up automated compilation of the .fmu are given under the section External dependencies in the above README file.

1.2.4 OpenModelica

OpenModelica is an open-source Modelica-based modeling and simulation environment intended for industrial and academic usage.



The short-term goal is to develop an efficient interactive computational environment for the Modelica language, as well as a rather complete implementation of the language. It turns out that with support of appropriate tools and libraries, Modelica is very well suited as a computational language for development and execution of both low level and high level numerical algorithms, e.g. for control system design, solving nonlinear equation systems, or to develop optimization algorithms that are applied to complex applications.

The longer-term goal is to have a complete reference implementation of the Modelica language, including simulation of equation based models and additional facilities in the programming environment, as well as convenient facilities for research and experimentation in language design or other research activities. However, our goal is not to reach the level of performance and quality provided by current commercial Modelica environments that can handle large models requiring advanced analysis and optimization by the Modelica compiler.

Documentation

The OpenModelica users guide as pdf or epub or as html. The OpenModelica API documentation. More documentation can be found here.

Export an FMU from OpenModelica

To export an FMU from a Modelica model one can use a exportFMU.mos script:

One can run the exportFMU.mos script via the omc[.exe] executable.
On Windows:
```
%OPENMODELICAHOME%\bin\omc exportFMU.mos
```

On Linux (if OpenModelica is installed via apt-get):

```
omc exportFMU.mos
```

On MacOS X:

```
omc exportFMU.mos
```

Documentation of the function translateModelFMU.

Link to OpenModelica latest binaries

The latest release binaries for Windows can be found here. The latest nightly-builds binaries for Windows can be found here.

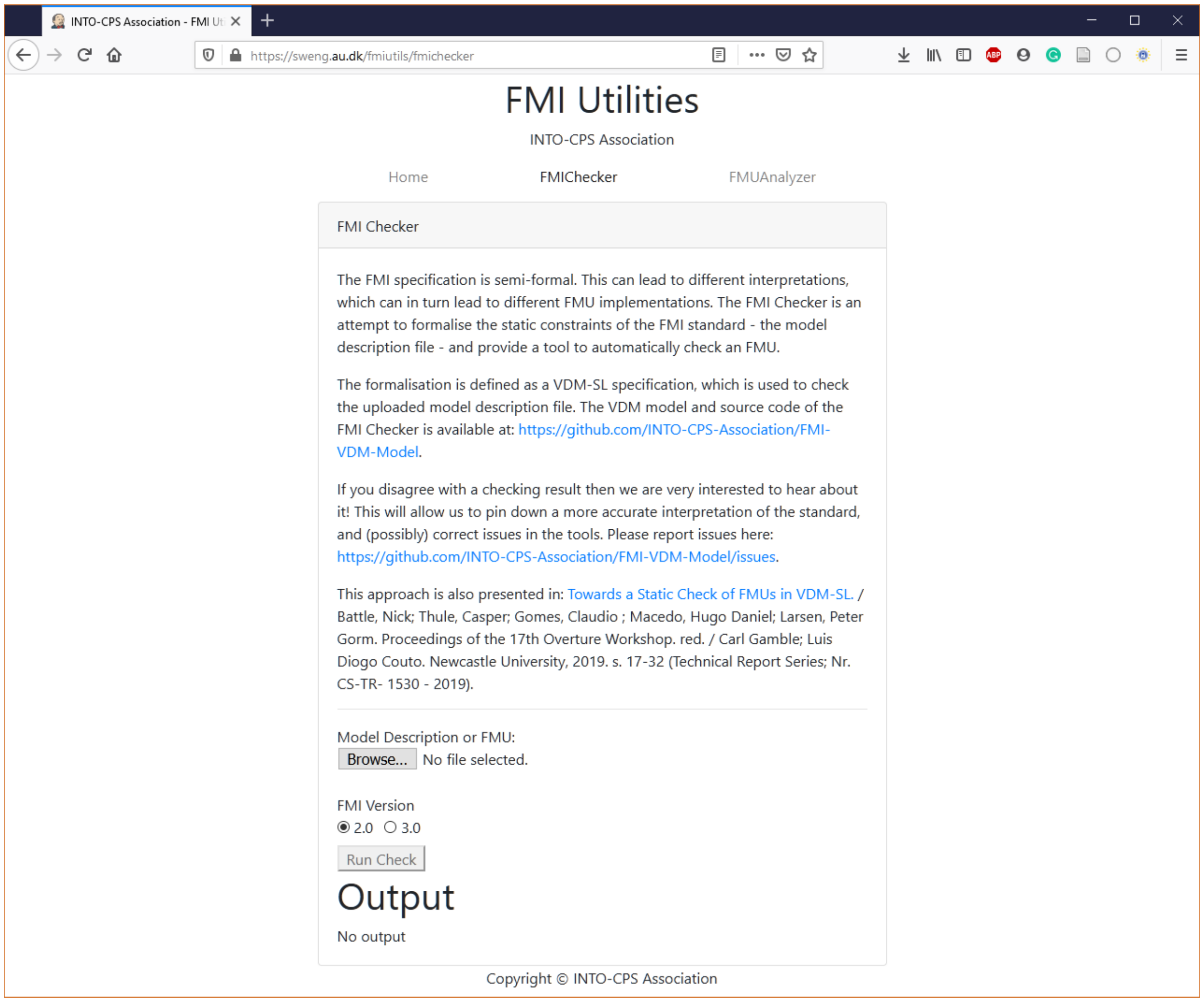
We also support Linux and Mac OS installations.

For more information see OpenModelica users documentation.

1.3 Verifying compliance of FMUs

To maximize compatibility between an FMU and existing simulation tools it is import to ensure strict compliance to the FMI standard. Unfortunately, the current situation is less that ideal, since a large portion of FMUs encountered do not implement the standard correctly.

1.3.1 FMIChecker



INTO-CPS Association - FMI Ut
https://sweng.au.dk/fmiutils/fmichecker

FMI Utilities

INTO-CPS Association

Home FMIChecker FMUAnalyzer

FMI Checker

The FMI specification is semi-formal. This can lead to different interpretations, which can in turn lead to different FMU implementations. The FMI Checker is an attempt to formalise the static constraints of the FMI standard - the model description file - and provide a tool to automatically check an FMU.

The formalisation is defined as a VDM-SL specification, which is used to check the uploaded model description file. The VDM model and source code of the FMI Checker is available at: https://github.com/INTO-CPS-Association/FMI-VDM-Model.

If you disagree with a checking result then we are very interested to hear about it! This will allow us to pin down a more accurate interpretation of the standard, and (possibly) correct issues in the tools. Please report issues here: https://github.com/INTO-CPS-Association/FMI-VDM-Model/issues.

This approach is also presented in: Towards a Static Check of FMUs in VDM-SL. / Battle, Nick; Thule, Casper; Gomes, Claudio ; Macedo, Hugo Daniel; Larsen, Peter Gorm. Proceedings of the 17th Overture Workshop. red. / Carl Gamble; Luis Diogo Couto. Newcastle University, 2019. s. 17-32 (Technical Report Series; Nr. CS-TR- 1530 - 2019).

Model Description or FMU:
Browse... No file selected.

FMI Version
[✓ 2.0] 3.0

Run Check

Output

No output

Copyright © INTO-CPS Association

The FMIChecker is a command line tool that provides thorough static check of FMUs. Install instructions can be found on the GitHub Repository. Alternatively, the tool can be accessed at this webpage, eliminating the need to install the tool.

1.3.2 RT-Tester

Core

RT-Tester is a test automation tool for automatic test generation, test execution and real-time test evaluation. Key features include a strong C/C++-based test script language, high performance multi-threading, and hard real-time capability. The tool has been successfully applied in avionics, rail automation, and automotive test projects.

More information is available on the product web site here.

Model Based Extension (RTT-MBT)

RT-Tester Model Based Test Case and Test Data Generator (RTT-MBT) supports model-based testing (MBT), that is, automated generation of test cases, test data, and test procedures from UML/SysML models. A number of common modelling tools can be used as front-end for this. The derived test procedures use the RT-Tester Core as a back-end, allowing the system under test to be provided on real hardware, software only, or even just simulation to aid test model development. RTT-MBT includes requirement tracing from test models down to test executions and allows for powerful status reporting in large scale testing projects. More information is available on the product web site here.

FMI/FMU support for RT-Tester / RTT-MBT

The RTT-MBT component is catering for FMI/FMU by a specialised feature release that allows to cast a test procedures to an FMI2-compliant FMU. That FMU is in input to the COE.

Links to Documentation

The links to documentation of different tools are:

- Maestro CLI Interface
- INTO-CPS Library
- Python FMUs
- Maestro v2

Downloads

In order to operate properly, it is necessary to first obtain the prerequisite tools Python-2.7 and gcc-4.9_v2. An installer collection is available for Windows via GitHub from here.

RT-Tester Core, RTT-MBT, RTTUI3, and prepared example projects can be downloaded and installed by one click (on Windows) using the VSI_bundle_v0_1_1.exe file. This file can be downloaded via GitHub from here.

Updates of the installers will be available at the same URL & GitHub repository.

1.4 Interfacing

1.4.1 RabbitMQ FMU

RabbitMQ FMU provides a way to bring external data into an FMI co-simulation.

CHAPTER 2

Tutorials

The tutorials are available at https://github.com/INTO-CPS-Association/training

2.1 Case Studies

There are several case studies in the INTO-CPS Project. Besides the links below, please see the file `The INTO-CPS Examples Compendium`.

2.1.1 INTO-CPS FMI JNI

It is possible to write Functional Mockup Units (FMUs) in Java. This repository illustrates the use of Java Native Interface (JNI) for providing Functional Mockup Interface (FMI) to FMUs written in Java.

2.1.2 Incubator

The Incubator case study is currently under development. It will be added in a future update.

2.1.3 Single Tank

2.1.4 Three Tank

2.1.5 UAV Swarm

2.1.6 Fan Coil Unit (FCU)

2.1.7 Line Follower Robot

2.1.8 Ether

2.1.9 Turn Indication

2.1.10 Mass Spring Damper

CHAPTER 3

Guides

If you are interesting in publishing a guide related to concepts relevant to INTO-CPS, then please see *Contributors Guide*.

If you would like to take a more practical approach to learning these topics is by following the tutorials found in the *Tutorials* section.

Warning: Currently the concepts simulation, co-simulation, design space exploration, Functional Mock-up Interface and Functional Mock-up Unit are explained in `The INTO-CPS Method Guidelines`

CHAPTER 4

Contributors Guide

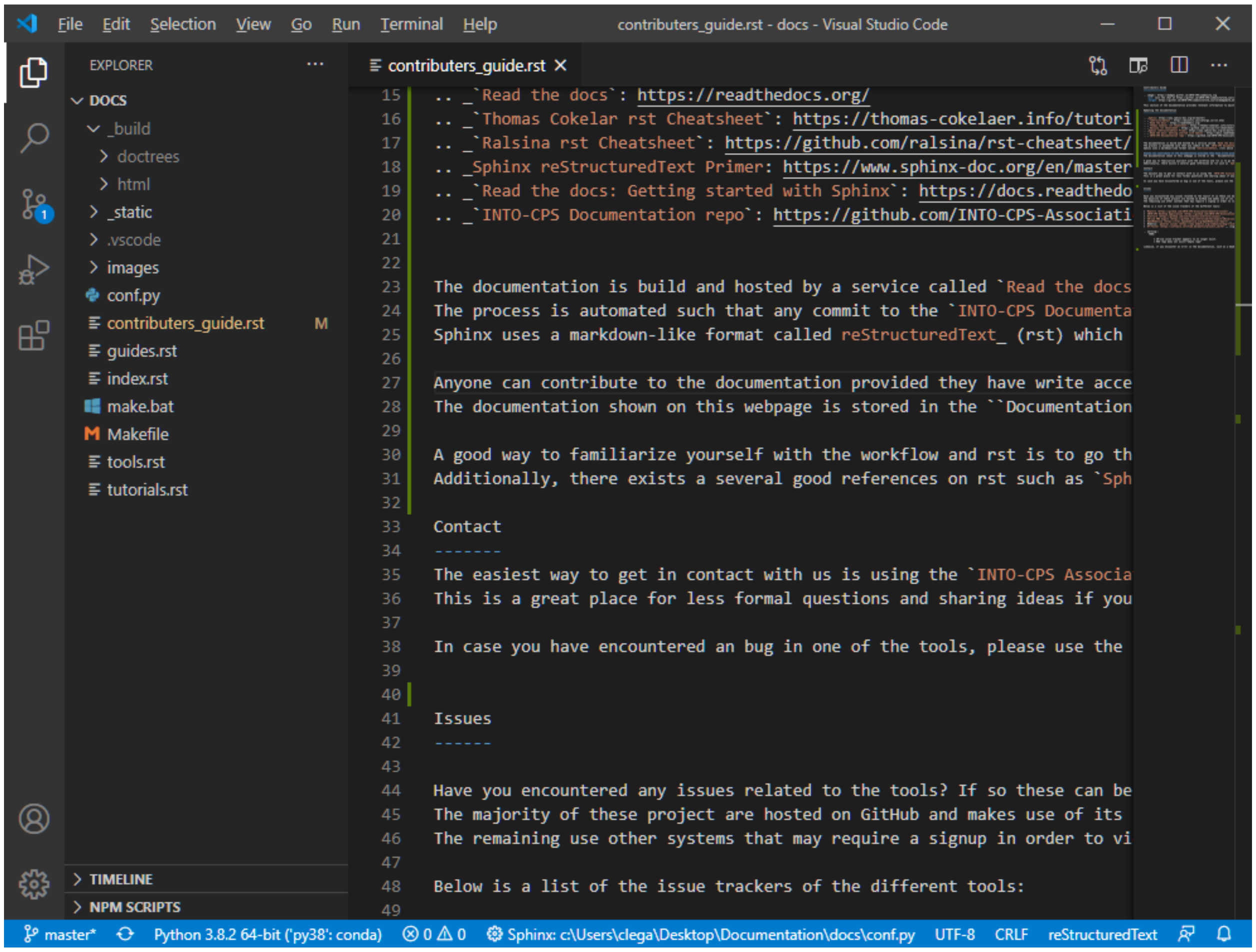
chat on gitter

This section of the documentation provides relevant information to maintainers and contributors of the project. If you are interested in becoming a contributor, please see https://into-cps.org/contact/.

4.1 Updating the documentation

The documentation is build and hosted by a service called Read the docs (rtd) which uses Sphinx to build html, pdf and epub documentation. The process is automated such that any commit to the INTO-CPS Documentation repo causes the documentation to be updated. Sphinx uses a markdown-like format called reStructuredText (rst) which is suitable for creating hierarchical document structures contains cross-references.

Anyone can contribute to the documentation provided they have write access to INTO-CPS Documentation repo. The documentation shown on this webpage is stored in the `Documentation/docs` directory as shown below. Of particular interest is the `index.rst` and `conf.py` file, the former being the entry point of the documentation and the latter being the configuration used by Sphinx.



```
.. _`Read the docs`: https://readthedocs.org/
.. _`Thomas Cokelar rst Cheatsheet`: https://thomas-cokelaer.info/tutori
.. _`Ralsina rst Cheatsheet`: https://github.com/ralsina/rst-cheatsheet/
.. _Sphinx reStructuredText Primer: https://www.sphinx-doc.org/en/master
.. _`Read the docs: Getting started with Sphinx`: https://docs.readthedo
.. _`INTO-CPS Documentation repo`: https://github.com/INTO-CPS-Associati


The documentation is build and hosted by a service called `Read the docs
The process is automated such that any commit to the `INTO-CPS Documenta
Sphinx uses a markdown-like format called reStructuredText_ (rst) which

Anyone can contribute to the documentation provided they have write acce
The documentation shown on this webpage is stored in the ``Documentation

A good way to familiarize yourself with the workflow and rst is to go th
Additionally, there exists a several good references on rst such as `Sph

Contact
-------
The easiest way to get in contact with us is using the `INTO-CPS Associa
This is a great place for less formal questions and sharing ideas if you

In case you have encountered an bug in one of the tools, please use the



Issues
------

Have you encountered any issues related to the tools? If so these can be
The majority of these project are hosted on GitHub and makes use of its
The remaining use other systems that may require a signup in order to vi

Below is a list of the issue trackers of the different tools:

```

```
master*   Python 3.8.2 64-bit ('py38': conda)   ⊗ 0 ⚠ 0   Sphinx: c:\Users\clega\Desktop\Documentation\docs\conf.py   UTF-8   CRLF   reStructuredText
```

The documentation can be build using the `make` command in the `docs` folder.

```
make html # builds html docs
make latexpdf # builds pdf
make linkcheck # verifies links are correct
```

A good way to familiarize yourself with the workflow and rst is to go through Read the docs: Getting started with Sphinx. Additionally, there exists a several good references on rst such as Sphinx reStructuredText Primer, Thomas Cokelar rst Cheatsheet , and Ralsina rst Cheatsheet .

4.2 Contact

The easiest way to get in contact with us is using the INTO-CPS Association Gitter channel or https://into-cps.org/contact/. This is a great place for less formal questions and sharing ideas if you want to contribute.

In case you have encountered an bug in one of the tools, please use the appropriate issue tracker as listed in *Issues*.

4.3 Issues

Have you encountered any issues related to the tools? If so these can be submitted to the bug tracker of the respective projects. The majority of these project are hosted on GitHub and makes use of its built-in issue tracker. The remaining use other systems that may require a sign-up in order to view and submit issues.

Below is a list of the issue trackers of the different tools:

- Maestro
- INTO-CPS Desktop Application
- Overture
- 20-sim FMU export
- OpenModelica
- Modelio: general issues and INTO-CPS related issues
- RT-Tester (signup required)

Warning: TODO:

- 20-sim issue tracker appears to no longer exist.
- DSE repo does not exist/"empty repo"

Likewise, if you encounter an error in the documentation, such as a dead link, or you have a suggestion for how to improve the documentation, please submit an issue to the documentation issue tracker