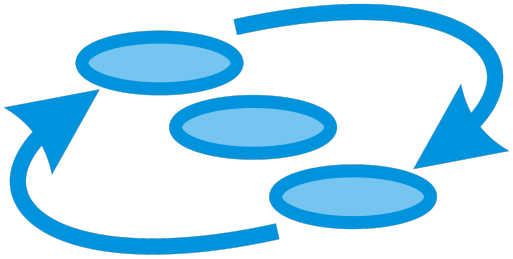
Grant Agreement: 644047

INtegrated TOol chain for model-based design of CPSs



Distributed testing and simulation network

Technical Note Number: D5.1b

Version: 0.3

Date: 2015

Public Document

http://into-cps.au.dk

Contributors:

Jörg Brauer, VSI
Oliver Möller, VSI
Adrian Pop, LIU

Editors:

Oliver Möller, VSI

Reviewers:

François Hantry, CLE
Andrey Sadovykh, ST
Claes Dühring Jaeger, AI

Consortium:

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

Document History

Ver	Date	Author	Description
0.1	2015-11-02	Oliver Möller	Initial document version
0.2	2015-11-16	Adrian Pop	Added OpenModelica description
0.3	2015-12-14	Oliver Möller	Addressed comments from reviewers: extended introduction; added reference to INTO-CPS traceability requirements; added conclusion

Abstract

This deliverable contains the conceptual description of the distributed testing and simulation framework. Modelio, OpenModelica, 20-sim or Overture can be used as a model editor to specify the test model of the system under test. The RT-Tester (in connection with the model-based extension RTT-MBT) is used as a configuration mechanism and as execution back-end that allows to separate stimulation, checking, and simulation tasks into separate FMUs. Requirements can be traced throughout the workflow within this tool chain.

Contents

1 Introduction

Based on FMI 2.0, a distributed testing and simulation framework together with a matching tool chain can be defined. Starting with a timed behavioural model (defined using a tool such as OpenModelica, Overture, 20-sim or Modelio) of the system under test (SUT), we can identify and separate test and simulation objectives.

For sake of clarity, we only focus on OpenModelica in this document. The tool supports several stages of the design process, most prominently the description of (behavioural) system requirements and a behavioural description of the system. This description can be thought of as a timed state chart, where inputs influence control flow and timing behaviour. The requirements are then annotations to states and/or transition, which confirm the proper realisation. Other classes of requirements may exist (e.g., concerning the capability of the target hardware), but are not addressed by testing; only aspects that correspond to observable behaviour can be tested.

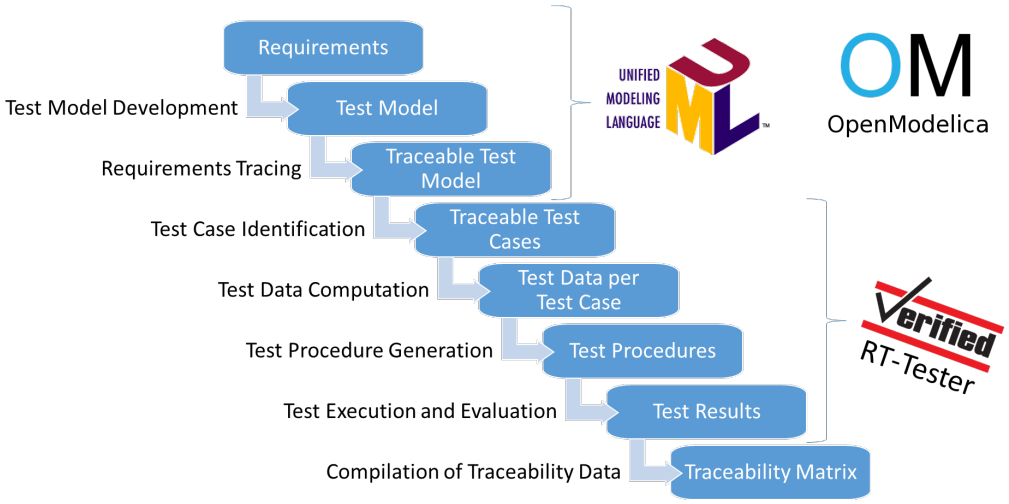


Figure 1: Work flow in connection with tool OpenModelica and RT-Tester.

Figure 1 sketches the work flow in the tool chain and explains the transition from the modelling part (OpenModelica) to the test operations (RT-Tester).

If the modelled design has progressed to a stage where a behavioural analysis gives meaningful insights, it is handed over to the test tool part via model export. The test design then defines a specific environmental interaction with a (prototypical) SUT implementation. If the implementation is correct, it

shows the same reactions to inputs as the modelled behavioural description. If needed, parts of the SUT can be simulated (based on a model description); this is also helpful to focus the analysis (by tests) on specific sub components. Testing includes the selection of data paths that shall be exercised in one run, since this is necessarily a finite experiment. This selection is also referred to as a *test goal*, and can be described in terms of model elements or sets of requirements that should be included. During a complete test project, situations related to all of the behavioural requirements are exercised.

For a specific test goal, a solver component (with RTT-MBT) derives a path through the model and generates corresponding SUT stimuli. The expected SUT outputs are transformed into a checker component, which allows to compare the observed SUT output with the ones predicted by the model. Additionally, it is possible to configure and generate simulation tasks by the same mechanism. It is important to stress that these two components — the checker component and the stimulation component — are independent of each other. A real-time capable test tool (RT-Tester) is used as execution back-end both for stimulation and for checking. This contains powerful mechanisms to define, organise, and trace requirements throughout a test project.

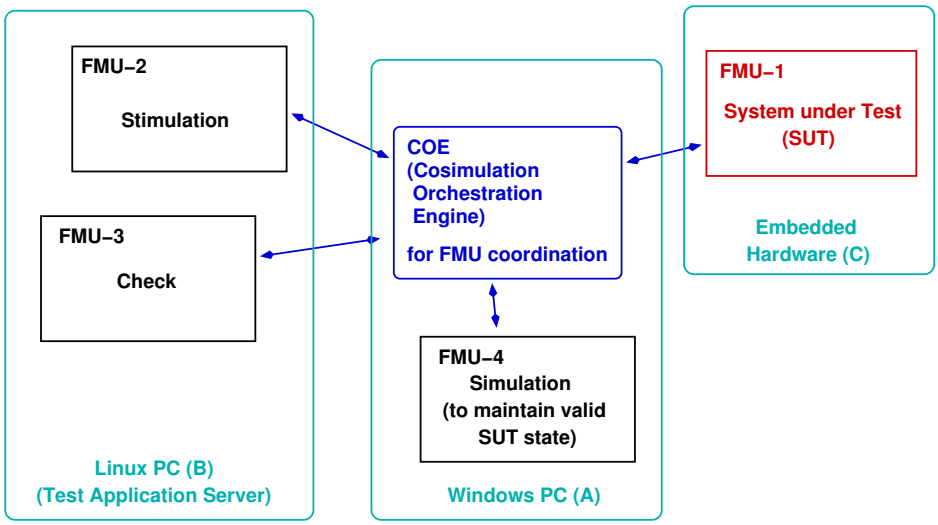


Figure 2: Distributed testing network.

Figure 2 indicates how the test execution works. This is distributed in the sense that not all actors have to be physically present on the same machine. For example, a Windows PC (A) could host the *Cosimulation Orchestration Engine (COE)*, which connects the relevant FMUs. On the same machine,

a Simulation component (FMU-4) can be run, e.g., simulating a part of the environment that does not contribute to the test objective, but is required in order to keep the SUT in a valid state. A Linux PC (B) can be used for the testing part, which involves both providing SUT inputs (Stimulation) and comparing the observed reactions with the expected outputs (Check). Finally, the SUT is here located on an embedded hardware (i.e., running in a target environment) and FMU-1 wraps the inputs and outputs in order to connect with the test environment.

2 Related Work

A substantial amount of literature is available for model-based testing and cyber-physical systems, for specific pointers see, e.g., [Pel13] and [CBM+13]. Since this work addresses several key issues (modelling, distribution, tracing), it would not be helpful to start citing long lists of literature here. Rather, it makes sense to relate this work to recent research projects with similar aim and scope.

In this respect, the following projects can be considered to be closely related.

ADVANCE[1] addresses analysis of connected CPSs, but focuses on formal verification rather than on testing.

AMADEOS[2] is also concerned with the time aspects of system of systems, which relates to the timed nature of the tool chain described here.

MODRIO[3] addresses the requirement modelling aspect, but not to the extend that it is confirmed by tests.

COMPASS[4] and **PTOLEMY**[5] both address the distributed aspects with mixture of models of computation, without casting this into the FMI 2 co-simulation standard.

Further (less closely) related work can be found in [Con14, Appendix B].

[1] `http://www.advance-ict.eu/`
[2] `http://amadeos.imag.fr`
[3] `https://itea3.org/project/modrio.html`
[4] `http://www.compass-research.eu/`
[5] `http://ptolemy.berkeley.edu/publications/index.htm`

3 Overview of Testing and Simulation Interfaces

3.1 RT-Tester

Testing is the process of running a system in a real or simulated environment in order to compare its actual to its expected behaviour. For many stages of this process, tool support is essential in order to increase the number of situations that can be exercised in a limited amount of time.

The RT-Tester[6] is a test automation tool for test generation, test execution and real-time test evaluation [Ver15a]. This core tool can be supplemented by a model-based testing extension (RTT-MBT) that derives execution sequences and checks from a state-chart like system description [Ver15b]. Both can be operated via the same GUI (RTTUI). For brevity, core + model-based extension + GUI are simply referred to as "RT-Tester" in the following.

3.1.1 RT-Tester Operations

The starting point for testing and simulation interfaces for the RT-Tester is the conceptual distinction of system under test (SUT) and environment, see Figure 3. Inputs to the SUT are stimulations. Outputs of the SUT can be used for testing, in the sense that the observed SUT reactions (to stimuli) can be compared to the expected behaviour. In addition, SUT outputs may expose internal states or contain diagnostic data that may not be directly relevant for testing.

In the context of model-based testing, the test system serves as the environment (to the SUT) in the sense that it steers the stimulation of the SUT. The objective of test generation is then twofold:

1. Provide stimuli to the SUT that "force" it to exhibit a desired behaviour, that is, the behaviour that has been specified as input to the test generation process.
2. Compare the SUT outputs with the expected values for the appropriate situation.

For both objectives it is required to have a specification on how the SUT is supposed to react to inputs. This is captured by a *behavioural model* of the

[6] `https://www.verified.de/products`

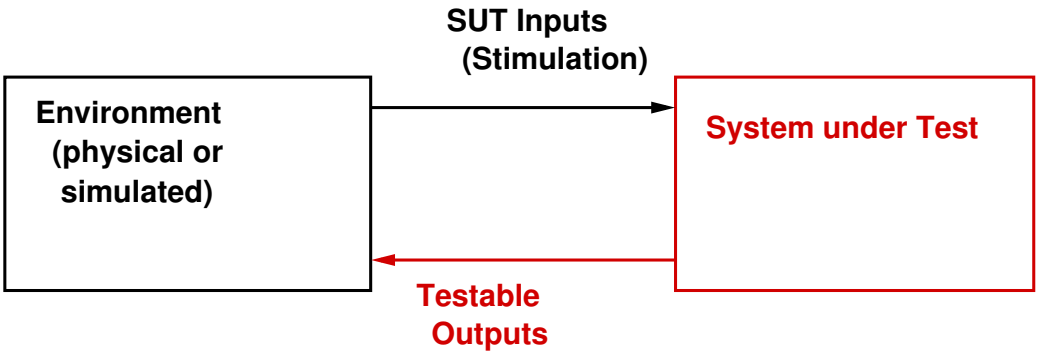


Figure 3: High-Level interface between environment and SUT.

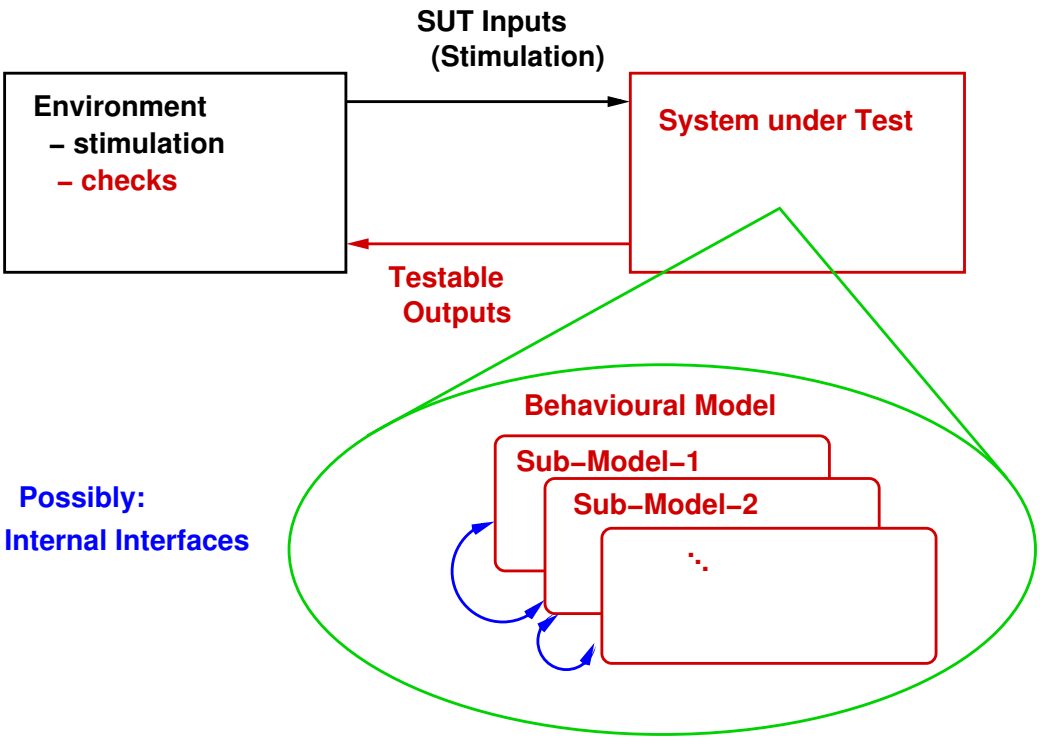


Figure 4: The SUT component is complemented by a behavioural description.

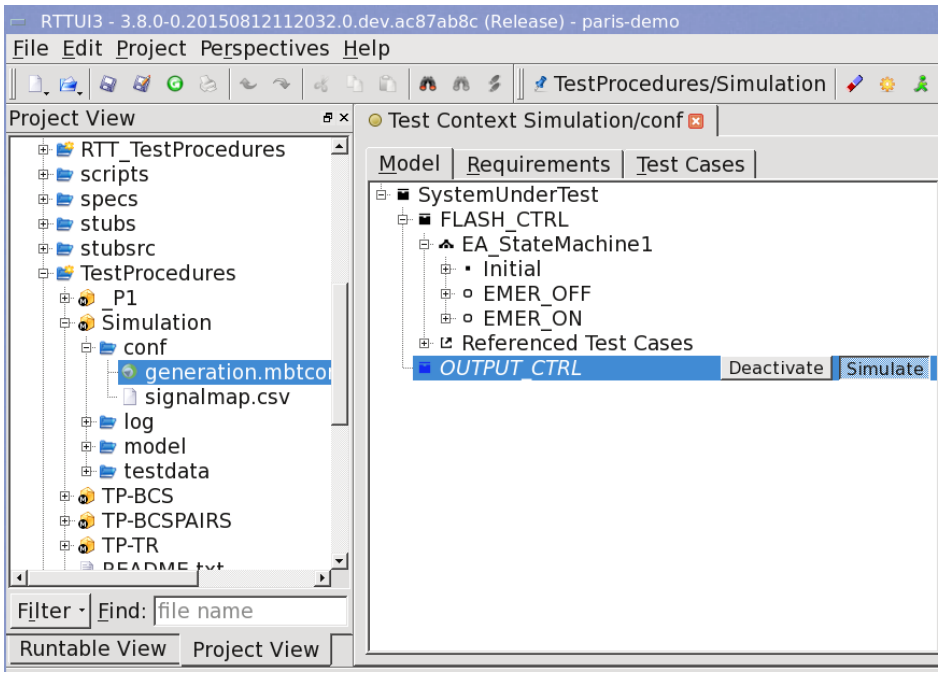


Figure 5: Selecting parts of the SUT to create a simulation.

SUT, see Figure 4. In general, the behavioural model can be understood as a network of (possibly hierarchical) timed state machines. The design may give rise to internal interfaces between parallel components.

The RT-Tester Model-Based Test case generator (RTT-MBT) creates tests by instantiating the environment/SUT interface. This means that all SUT inputs have to be provided in a timely manner, while all outputs of the SUT need to be evaluated. The environment side is thus filled in by test logic that both steers the SUT into certain "interesting" situations and compares the outputs to expected observable behaviour.

In addition, it is possible to generate *simulations* for certain aspects of the SUT which are currently not relevant for testing. This allows parts of the SUT to be developed independently for example.

Both can be done via the RTTUI, see Figure 5.

3.1.2 Integration in the INTO-CPS Tool Chain

The primary purpose of RT-Tester in the INTO-CPS context is to provide means to perform *test case and requirement tracing* for given test models.

This requires two phases of user interaction.

a) The *test model* has to be constructed and user-validated for design compliance. For this the OpenModelica tool (see section 3.2) is the primary contributor, since it allows exploration of the constructed model before going forward with the next development steps. The *System Design Model* serves as test model in this context, since it captures the behavioural description. The RT-Tester works on the model export.

b) Since a test can only demonstrate one incarnation of SUT behaviour, it must be possible to derive sequences of inputs based on this test model. This is realised by the definition of *test goals*, that capture depth and nature of the expected model coverage.

The test model already (implicitly) contains the *SUT requirements* by mere structure. The test tool back-end is then used to identify test cases and derive a tracing to the SUT requirements. The test case results (also known as *verdicts*) are then obtained by comparing the observed SUT output with the outputs predicted by the test model.

3.1.3 INTO-CPS Requirements (RT-Tester)

From the high level requirements in the INTO-CPS requirements report D7.3 [LPH+15] the RT-Tester tool chain must satisfy the following with respect to testing:

- Requirement **0025** - RT-Tester must provide an INTO-CPS FMI tool wrapper that is compliant with the COE.
- Requirement **0026** - RT-Tester must be able to automatically generate test cases and test data for a co-simulation configuration.
- Requirement **0027** - RT-Tester must be able to control and influence the behaviour of the co-simulation.
- Requirement **0028** - RT-Tester must be able to automatically validate the result of a co-simulation against the desired behaviour.

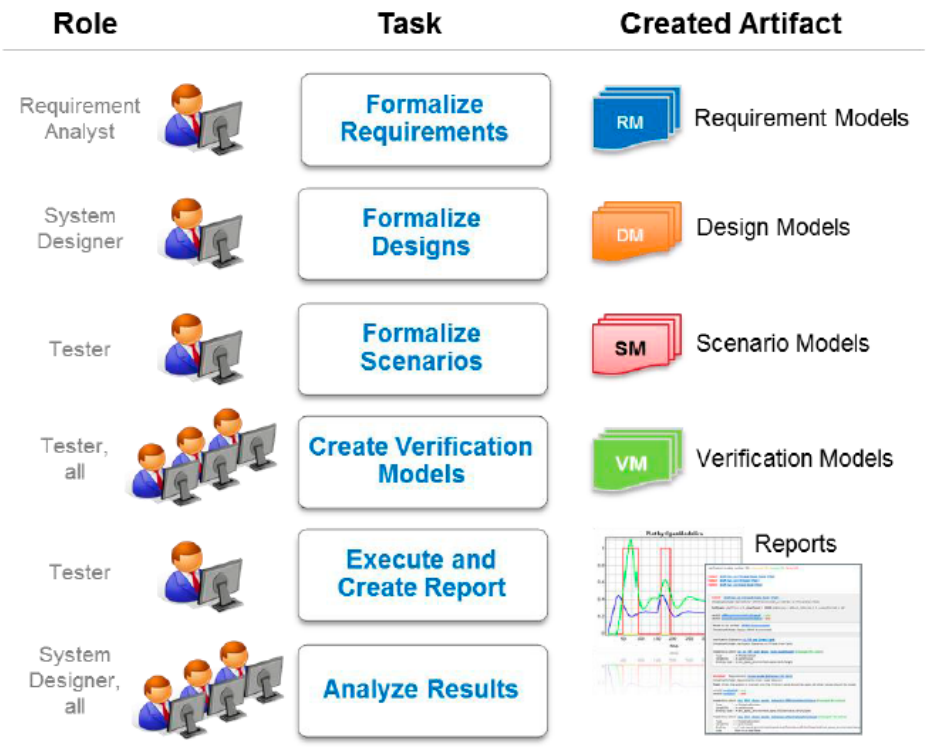


Figure 6: vVDR Overview.

3.2 OpenModelica

The goal of OpenModelica[7] team in the INTO-CPS project with regards to testing is assessing, adapting, applying and extending the previously developed ModelicaML framework [Sch13] for testing and verification of requirements and models.

3.2.1 The virtual Verification of Designs against Requirements method

The proposed method in ModelicaML framework is virtual Verification of Designs against Requirements (vVDR) and is depicted in Figure 6.

It is crucial to separate the experimental description from the system model. To analyse designs under different conditions, deterministic scenarios will need to be modelled. In vVDR the minimum set of modelling artifacts that

[7] https://www.openmodelica.org/

is required for design verification are:

- **Requirement Model** (RM): Each requirement is represented by a model that indicates requirement violation at any simulated time instant.
- **System Design Model** (DM): A particular design alternative or version is represented by a model to be used for verification against requirements.
- **Scenario Model** (SM): A scenario defines the course of actions to be simulated. It emulates the user or external events that will be experienced by the system.
- **Verification Model** (VM): Requirements, design alternatives, and scenarios can be combined in different ways. Such combinations are called verification models. They are used to perform design verification and to report on requirement violations.

Figure 7 shows the flow of information. Requirement models are the output of the formalise requirements activity and are the input to the formalise designs activity. Design and requirement models are the input to the formalise scenarios activity that outputs scenario models. All models are the input to the create verification models activity, which outputs verification models to be used for analysis in the execute and create report step.

In a scenario-based approach, a verification model will comprise one system design alternative that is to be verified against a set of requirements by running one verification scenario as illustrated in Figure 8.

Figure 9 shows the verification session report (both in the GUI tool and the HTML report) generated after executing verification models and post-processing simulation results.

3.2.2 Applying vVDR method in the INTO-CPS project

In the INTO-CPS context the modelling artifacts (RM, DM, SM, VM) of the vVDR method are all FMUs. The requirements can be modelled in SysML (Modelio) or Modelica (OpenModelica) and FMUs can be generated from them. The system under test SUT (or DM) is the behaviour model for the system to be tested and is provided by one of the tools Modelio, OpenModelica, 20-sim or Overture also in FMU form. The scenario models

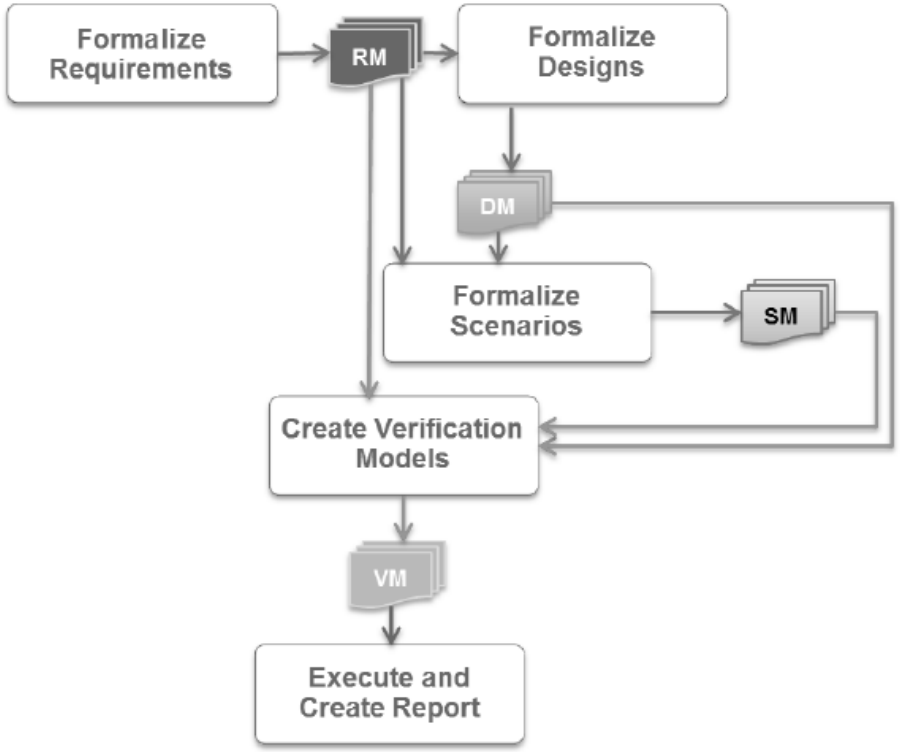


Figure 7: Information Flow in vVDR .

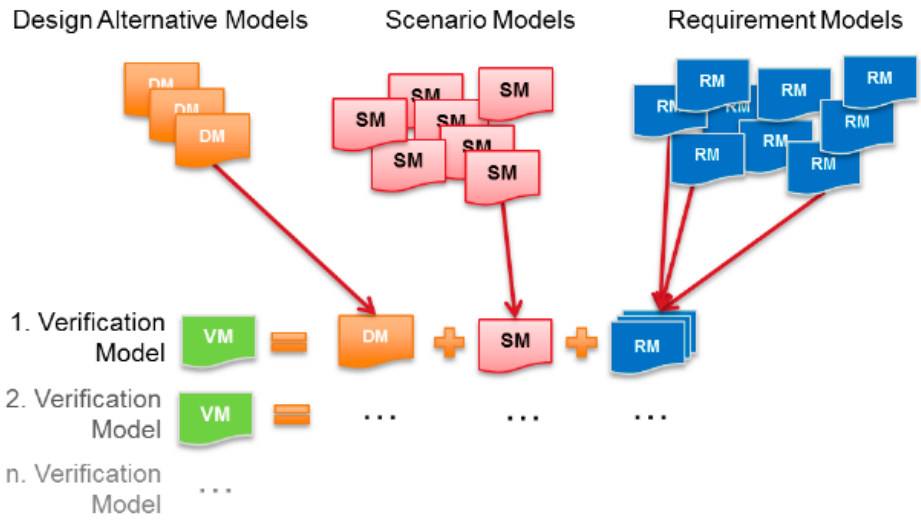


Figure 8: Concept of verification models.

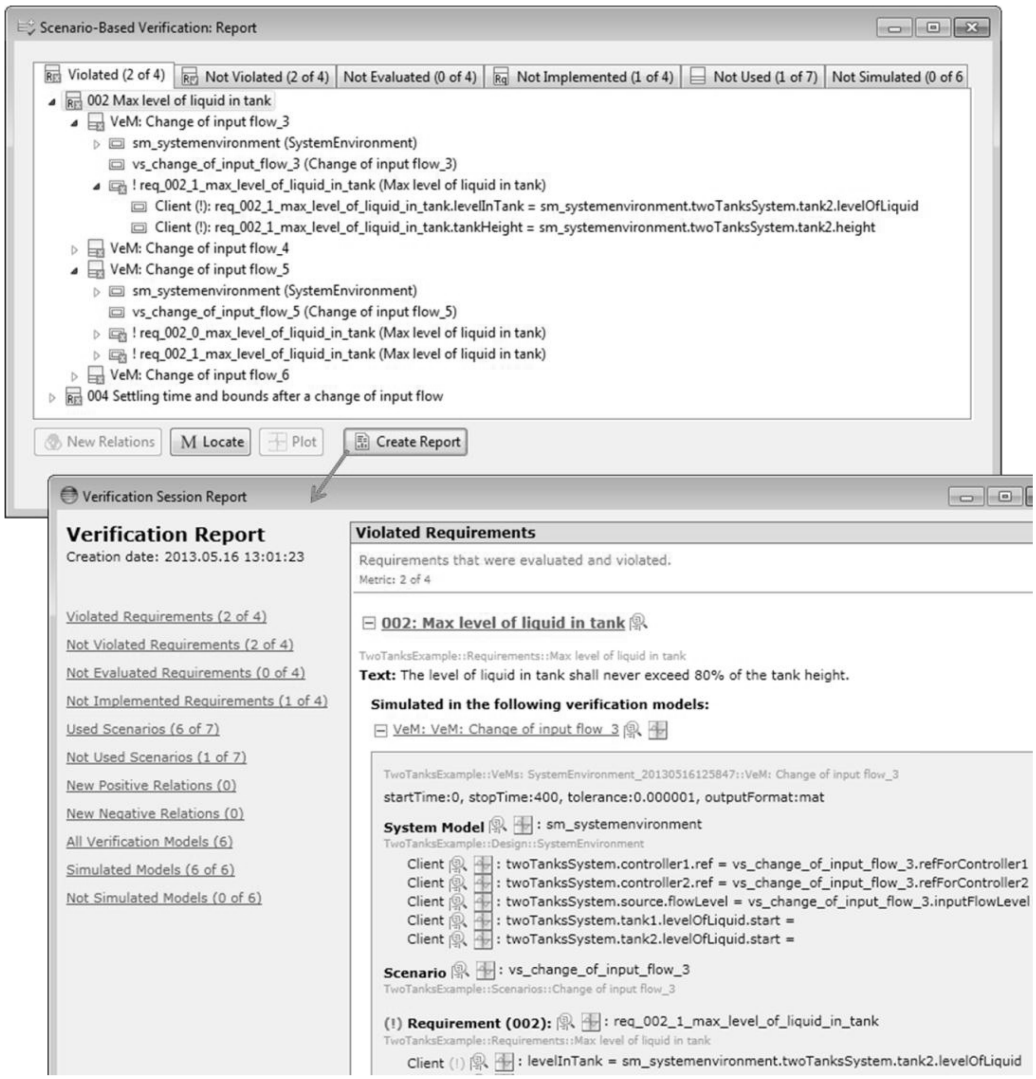


Figure 9: Verification session report GUI and HTML.

and verification models could be designed in SysML (Modelio) and exported as FMUs.

Then the verification models as FMUs can be ran on the COE to verify requirements.

3.2.3 INTO-CPS Requirements (OpenModelica)

From the high level requirements from the INTO-CPS requirements report D7.3 [LPH+15] the OpenModelica tool must satisfy:

- Requirement **0029** - OpenModelica must be able to read the CT model Modelio (SysML) requirements and generate Modelica code or FMUs for validation of requirements.
- Requirement **0030** - OpenModelica must be able to automatically validate the requirements via co-simulation on the COE.

4 Design of Test- and Simulation-Integration with COE

The key functionality of the COE is to provide a means for communication between different components (named FMUs), all of which provide an interface according to FMI 2.0. The exact configuration of the co-simulation environment, however, is controlled by the INTO-CPS App, which utilises an interface to Modelio so as to obtain FMU connection links.

4.1 RT-Tester FM-Interface Design Principles

After having designed a test model describing the SUT behaviour, the objective of test and simulation integration is to export the components as appropriate FMUs that can easily be integrated into a co-simulation environment. In essence, the test and simulation generation facilities must be able to generate FMUs that exhibit appropriate interfaces. In a UML/SysML modelling tool that can interact with RTT-MBT via XMI (such as Modelio), the inputs and outputs are modelled as explicit interfaces. The FMU generation functionality of RTT-MBT then simply transforms these interface definitions in UML/SysML according to the FMI 2.0 standard.

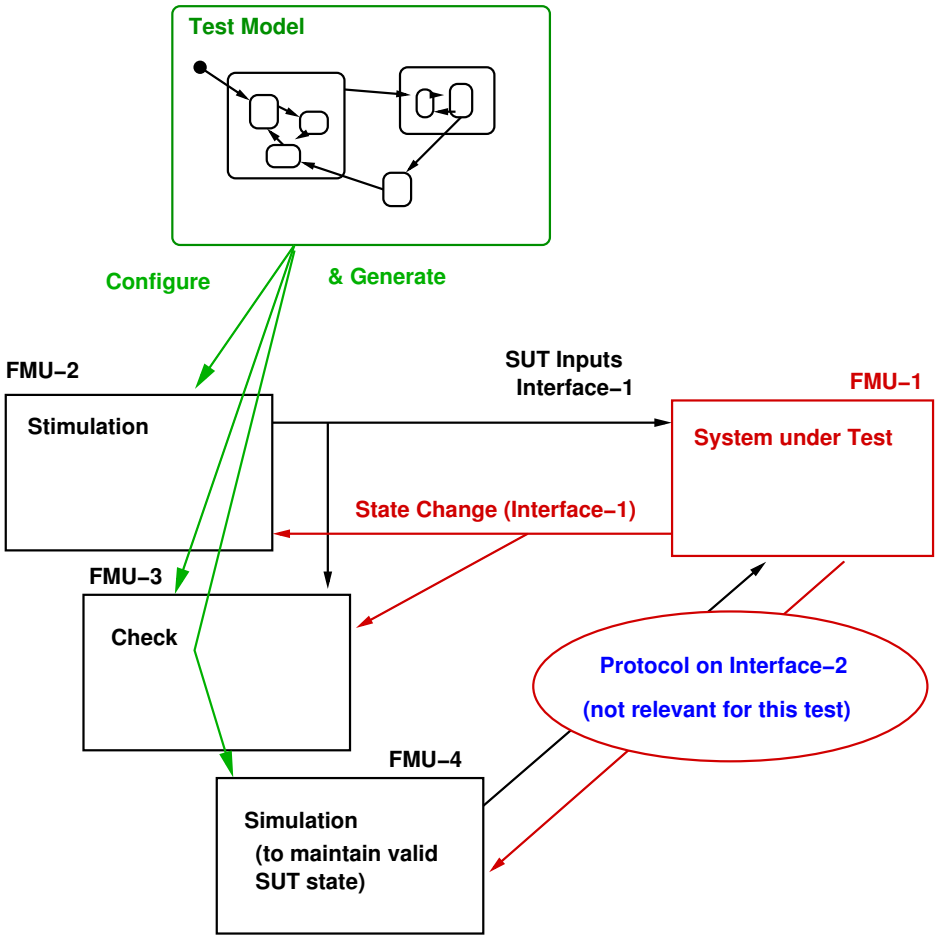


Figure 10: Functional distribution of test and simulation tasks.

Conceptually, the SUT should be treated as one FMU that only allows to stimulate and observe the high-level interface (black box testing). Here the model is only used to identify the inputs and outputs, compare FMU-1 in Figure 10. To this point, it is open whether FMU-1 corresponds to a real hardware execution or, e.g., a software simulation.

From the test model now a *stimulation* of the SUT inputs is derived, based on some test objective (more on this is found in subsequent section 4.2). This stimulation needs to read the SUT outputs "State Change on Interface-1" in general, because it has to act synchronised with the SUT. If synchronisation via time is sufficient, then this input can be omitted. In Figure 10, this stimulation is placed in FMU-2.

The *check* operations can be separated off to FMU-3, which naturally needs to be aware of all SUT outputs. It also needs to read the SUT inputs, since correctness of the SUT behaviour requires evaluation of the precondition.

The (optional) *simulation* part is exemplified by FMU-4, which is not concerned (at all) with the test logic, but interacts with the SUT on an isolated Interface-2. This can be necessary to keep the SUT in a "normal operation" state, e.g., by following a protocol-based data exchange that exchanges heart-beat data. It can be an advantage to have such an actor outside the simulation/check logic.

The RTTUI allows the user to select parts of the test model and generate behavioural simulations for these. The simulations then behave exactly as the test model prescribes.

4.2 RT-Tester Requirements-Tracing Design Principles

A requirement is usually identified by the following aspects.

- a (unique) identifier
- a precondition (situation where it applies)
- a set of *inputs* to the system
- a set of expected *outputs* of the system

A model of the SUT (Figure 10) can be annotated by such requirements. For example, in a certain model state (precondition) the system should act on

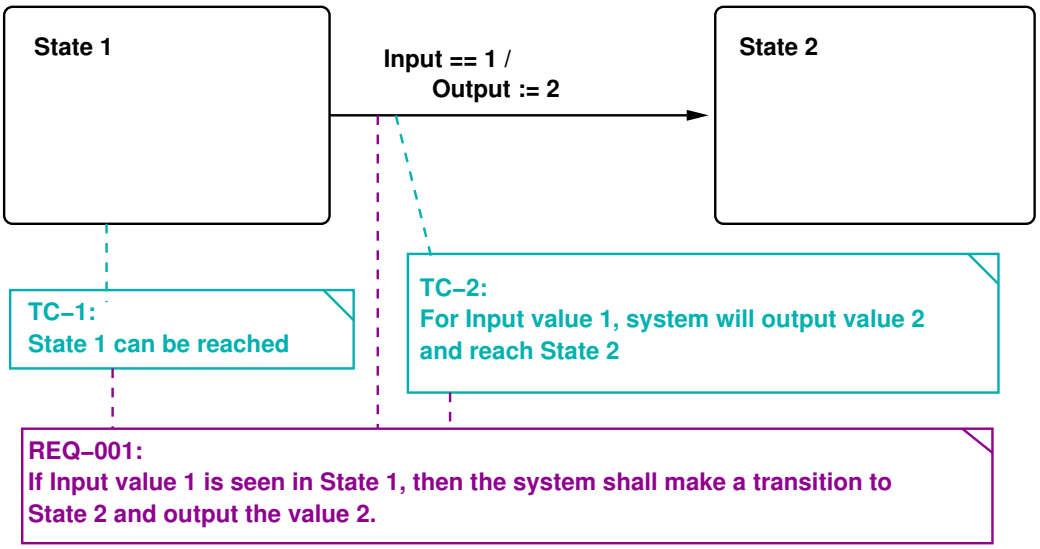


Figure 11: Model annotated with test cases TC-1, TC-2 and requirement REQ-001.

an input (stimulation) by changing the model state and generating a certain output (testable output).

Often a requirement demands several pairs of inputs and outputs. Therefore it is usually broken down into a set of *test cases*, each of which describes a situation that can unambiguously expressed by model elements. A requirement is covered, if for all aspects the related test cases have been exercised. One test case can be related to more than one requirement. Figure 11 gives a simple example for this, more details can be found in [Ver15b].

A model annotated with requirements serves as a *test model*. Based on this, *test procedures* can be configured by selecting a number of requirements (resp. test cases) that shall be covered during one execution.

The RTTUI allows to drag- and drop test cases into a set of desired goals, see Figure 12. In addition it is possible to activate test strategies, e.g., "cover all basic control states". It is also possible to specify LTL formulas that shall hold in the execution. The conjunction of these properties defines the set of test goals that shall be covered by a single test procedure.

Based on the model, this results in a set of timed inputs and corresponding checks of the SUT outputs (compare [Ver15b]).

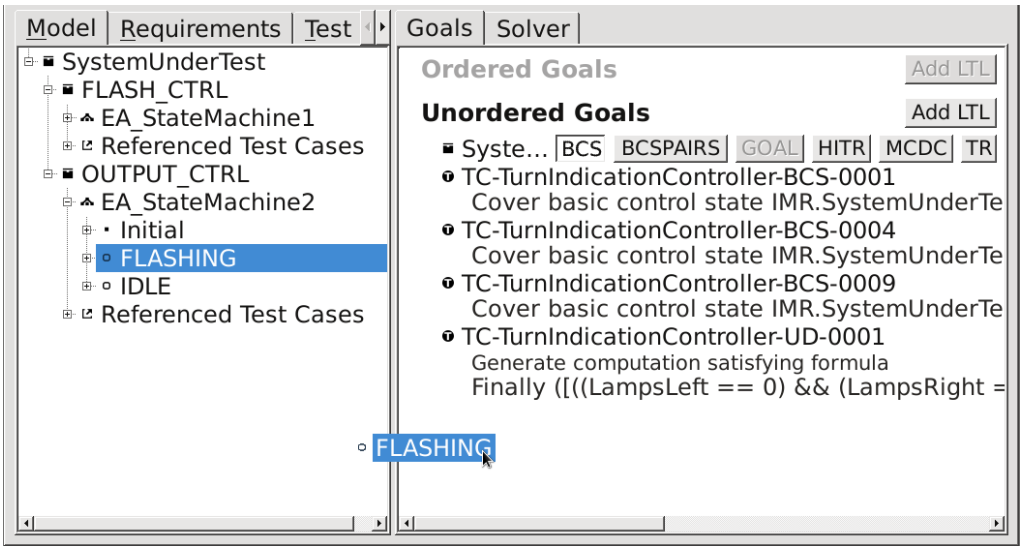


Figure 12: Configuring a set of test-cases that shall be covered in on execution, here by drag-and-drop of all states associated with "FLASHING".

If this is put together in a FMU execution driven by the COE, for each test case the associated result (i.e., the verdict) is recorded. This result can be either PASS or FAIL. Further details are elaborated in [Ver15a].

Importantly, one test case may be executed in more than one test procedure. The sum of the test case verdicts is computed on a project (or: sub-project) basis. The result (or verdict) of a requirement is then derived by the sum of associated test cases; it is only PASS if all associated test cases are PASS. Likewise, it is FAIL if at least one associated test case result has the verdict FAIL.

The status browsing of requirements can be done graphically in the RT-TUI, Figure 13 gives an impression on how this is organised. Appropriate exports can be generated to other requirement management tools such as DOORS.

4.2.1 INTO-CPS Requirements (Traceability)

From the high level requirements from the INTO-CPS requirements report D7.3 [LPH+15] the tool chain must satisfy the following with respect to traceability:

- Requirement **0015** - It must be possible to trace which FMU instances

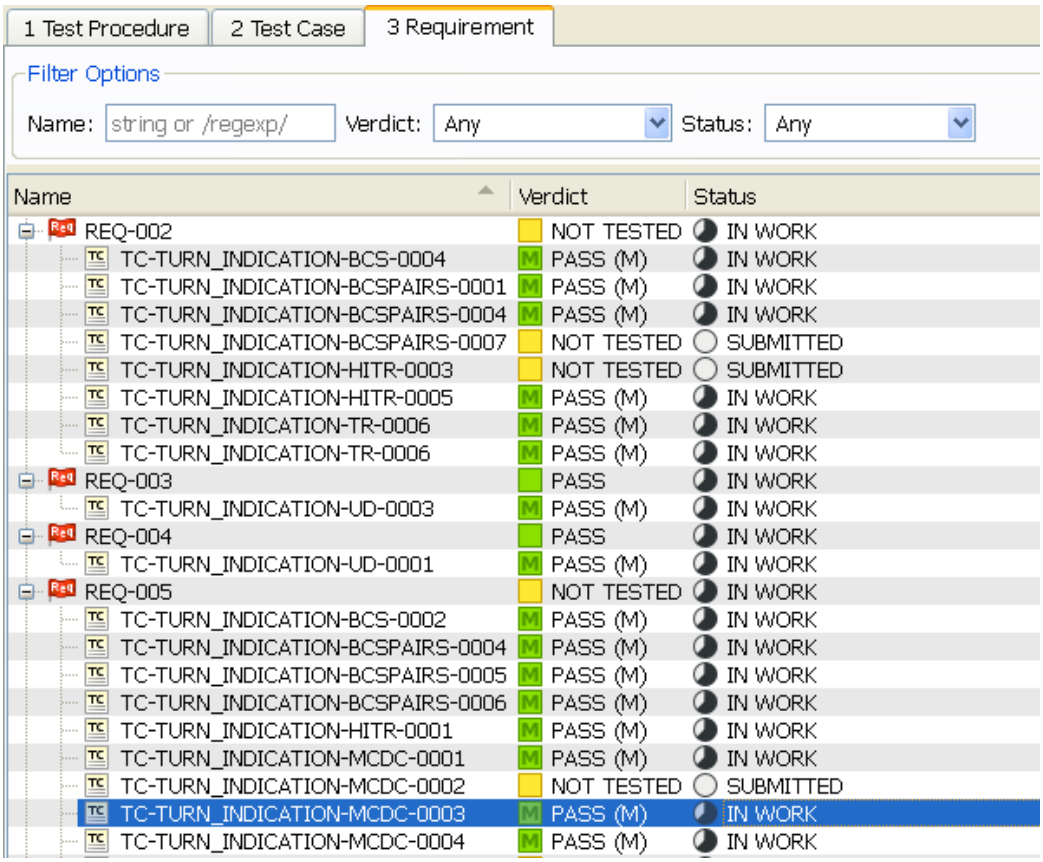


Figure 13: Requirements coverage after test execution.

contribute to the satisfaction of which requirements

- Requirement **0017** - It must be possible to trace test cases to test/simulation results.

4.3 OpenModelica

The integration of ModelicaML vVDR method in the INTO-CPS development and testing framework is not yet defined. The integration will be available at M24 of the project.

5 Conclusion

During year 1 of INTO-CPS, the prototypical tool chain has been set up to the point where model-based descriptions can be translated into FMUs. This holds both for the model simulation side and the test side, where the behavioural description serves to derive the required stimulation and the expected SUT behaviour.

Concerning the traceability, the basic idea has been sketched: Requirements are associated with dynamic model elements like locations or transitions. Evidence of fulfilment is then collected in terms of test executions that reach or exercise these model elements. An atomic element corresponds to one test case. Each test execution then gives partial evidence of correctness in terms of test case verdicts; this is mapped to the associated requirements in an accumulation step. During year 2, this part has to be implemented, tried and tested.

References

[CBM+13] M. V. Cengarle, S. Bensalem, J. McDermid, R. Passerone, A. Sangiovanni-Vincentelli, and M. Törngren. Characteristics, capabilities, potential applications of Cyber-Physical Systems: a preliminary analysis. Project Deliverable D2.1, EU Framework 7 Project: Cyber-Physical European Roadmap & Strategy (CyPhERS), November 2013.

[Con14] INTO-CPS Consortium. Grant agreement for INtegrated TOol chain for model-based design of CPSs (INTO-CPS). Grant agreement number 644047, European Commission, December 2014.

[LPH+15] Peter Gorm Larsen, Ken Pierce, Francois Hantry, Joey W. Coleman, Sune Wolff, Kenneth Lausdahl, Marcel Groothuis, Adrian Pop, Miran Hasanagić, Jörg Brauer, Etienne Brosse, Carl Gamble, Simon Foster, and Jim Woodcock. Requirements Report year 1. Technical report, INTO-CPS Deliverable, D7.3, December 2015.

[Pel13] Jan Peleska. Industrial-strength model-based testing - state of the art and current challenges. In Alexander K. Petrenko and Holger Schlingloff, editors, Proceedings Eighth Workshop on *Model-Based Testing,* Rome, Italy, 17th March 2013, volume 111 of *Electronic Proceedings in Theoretical Computer Science*, pages 3–28. Open Publishing Association, 2013.

[Sch13] Wladimir Schamai. *Model-Based Verification of Dynamic System Behavior against Requirements: Method, Language, and Tool.* PhD thesis, Linkoping University, Department of Computer and Information Science. Linkoping University, The Institute of Technology., 2013. http://liu.diva-portal.org/smash/get/diva2:654890/FULLTEXT01.pdf.

[Ver15a] Verified Systems International GmbH, Bremen, Germany. *RT-Tester 6.0: User Manual*, 2015. `https://www.verified.de/products/rt-tester/`, Doc. Id. Verified-INT-014-2003.

[Ver15b] Verified Systems International GmbH, Bremen, Germany. *RT-Tester Model-Based Test Case and Test Data Generator – RTT-MBT: User Manual*, 2015. `https://www.verified.de/products/model-based-testing/`, Doc. Id. Verified-INT-003-2012.

A List of Acronyms

AST	Abstract Syntax Tree
AU	Aarhus University
COE	Co-simulation Orchestration Engine
CPS	Cyber-Physical Systems
FMI	Functional Mockup Interface
FMI-Co	Functional Mockup Interface – for Co-simulation
FMI-ME	Functional Mockup Interface – Model Exchange
FMU	Functional Mockup Unit
GUI	Graphical User Interface
HiL	Hardware-in-the-Loop
HMI	Human Machine Interface
HTML	HyperText Markup Language
HW	Hardware
ICT	Information Communication Technology
IDE	Integrated Design Environment
LIU	Linköping University
LTL	Linear Timed Logic
M&S	Modelling and Simulation
MBD	Model Based Design
MBT	Model Based Testing
MiL	Model-in-the-Loop
OS	Operating System
RTT	RT-Tester
RTTUI	RT-Tester graphical User Interface
SiL	Software-in-the Loop
SUT	System Under Test
SysML	Systems Modelling Language
TA	Test Automation
TRL	Technology Readiness Level
TWT	TWT GmbH Science & Innovation
UY	University of York
VSI	Verified Systems International
WP	Work Package
XMI	XML Metadata Interchange
XML	Extensible Markup Language