Grant Agreement: 644047

INtegrated TOol chain for model-based design of CPSs

Demonstration of Integrated Co-Simulation and Testing

Deliverable Number: D5.3b

Version: 1.0

Date: 2017

Public Document

http://into-cps.au.dk

Contributors:

Jörg Brauer, VSI
Luis Diogo Couto, UTRC
Marcel Groothuis, CLP
Miran Hasanagić, AU
Kangfeng Ye, UY

Editors:

Jörg Brauer, VSI

Reviewers:

Julien Ouy, CLE
Etienne Brosse, ST
Frederik Foldager, AI
Peter Gorm Larsen, AU

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	24-08-2017	Jörg Brauer	Outline
0.2	06-11-2017	Miran Hasanagić	Initial version of related work
0.3	28-11-2017	Jörg Brauer	Ready for review
0.4	05-12-2017	Jörg Brauer	Integrated review by Frederik Foldager
1.0	08-12-2017	Jörg Brauer	Reworked after internal reviews

Abstract

This deliverable discusses the details of a hardware-in-the-loop setup, which connects a fan coil unit controller running on an embedded device with model-based tests. All relevant steps, that is, the generation of FMUs, the generation of tests, and the execution of the tests, are performed using tools and techniques developed within the course of the INTO-CPS project. The deliverable further contains an outlook on how industrial-scale testing could benefit from the applied techniques.

Contents

1 Introduction

One of the significant advantages of system development based on co-simulation is *decoupling* of the system components. Whilst the overall system design may be coupled, the different components that constitute the overall system are coupled only through their external interfaces, which provide access to the inputs and outputs of the system. This conceptual form of decoupling is advantageous in many regards, for example:

- Components are interchangeable so that it is possible to execute fully implemented components together with simulations of component models. It is often the case that different components have reached different implementation levels. The ability to execute and evaluate the implementation of some components on the integration level, even though the overall system may be far away from being complete, is invaluable in industrial practice.
- The communication layer, which handles the interchange of information on the external interfaces of the system components, allows a spatial separation of the different testing components. For example, an avionics cabin controller installed in Germany could interact with sensors located in France, as long as the communication layer provides the required infrastructure.
- Additional components can easily be plugged into the system. For example, test oracles can directly be connected to the overall system without influencing the behavior of the system at all, because the only connection to the co-simulated system is in reading its interfaces.

The FMI 2.0 standard [Blo14] among some others, has had some immense success in recent years as it provides a framework for the interaction between system components of different kind. In particular, it defines the interfaces of the system-wide communication mechanisms, without detailing how these should be implemented.

This document describes an instantiation of the FMI 2.0 standard in the INTO-CPS project using different components for HiL testing, and discusses the interplay of various components and tools. The Co-Simulation Orchestration Engine (COE) provides the signal distribution and coordination between two components, an implementation of a fan-coil unit (FCU) running on a Raspberry 3 embedded device, and a test driver generated from a test model using RTT-MBT [PVL11, Pel13] running on a PC.

In theory, HiL testing of embedded systems is simple. In practice, however, the task turns out to be challenging due to the interaction between different components via external interfaces, and often also timing issues, which often lead to unpredictable test results. Approaching HiL testing via a co-simulation setup provides a separation of concerns, which may lead to more stable HiL testing of independent components. A single FMU becomes responsible for the interaction with the embedded device, whereas the communication between the different FMUs is provided via the COE. Each FMU is responsible only for communication with one embedded device, but there is no direct dependency on communication with other hardware devices. The situation is depicted in Fig. 1 for one test driver and one embedded device, but the setup may of course be extended to an arbitrary number of test drivers and devices, which still are only connected via FMI 2.0.

The remainder of this document is laid out as follows. First, Sect. 2 presents details of the overall setup, followed by a description of which kind of behaviors were actually simulated and tested in Sect. 3. The deliverable then concludes with a presentation of related work in Sect. 4 and a discussion in Sect. 5.

2 Co-Simulation Setup

This section discusses the overall setup used for HiL co-simulation of the FMU in the sense that it describes both, the model used to generate the implementation of the FCU running on a Raspberry Pi 3 device (cp. Sect. 2.1), and the test model, from which test stimulations were derived (cp. Sect. 2.2). The overall experiment setup is depicted in Fig. 1. A standard computer — the host — is running the COE, the test driver FMU, and a FMU based on 20-sim 4C which maintains the connection between the host and the embedded device.

2.1 FCU Controller Implementation

This section describes the FCU controller model used for HiL co-simulation. The controller is a CT Proportional Integral (PI) controller in the back calculation anti-windup scheme as shown in Fig. 2. Two implementations of the controller have been developed, one using Simulink and one using Dymola. The controller is identical in both versions. The Simulink version, however, was developed for co-simulation in the building case study, whereas

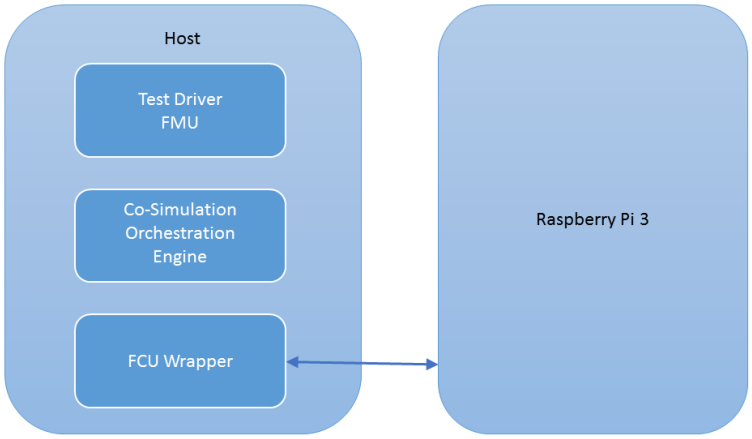

Figure 1: Overall setup of the co-simulation environment.

the Dymola version is used for HiL co-simulation. For background information about the building case study and additional details about the purpose and functionality of the FCU and its controller, refer to Deliverable D1.3d [CAB+17].

The FCU implementation is wrapped into FMUs and the following FMI ports are available for the FCU model to control the plant according to the user preferences:

`RAT_sp`: This input port enables setting the desired temperature for the room (the set-point). It should be connected to an FMU representing user preferences.

`Room_Temp`: This input port abstracts a temperature sensor and enables setting the measured temperature in the room. It is connected to the plant.

`VSD_Fan`: This output port indicates the speed at which the FCU fan should spin. It is connected to the plant.

`cool_valve`: This output port indicates the opening of the FCU coil valve. It is connected to the plant.

The model used for HiL co-simulation was re-developed in Dymola from the existing model. In order to use HiL co-simulation, the model must be

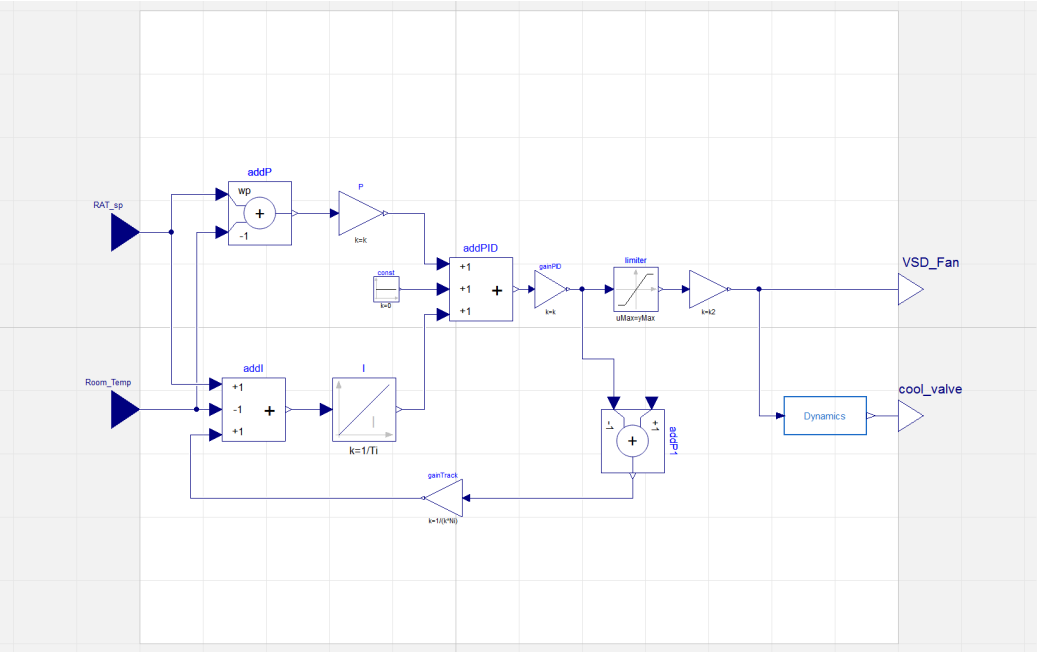

Figure 2: Dymola model of FCU controller.

exported as a source code FMU. The exporter for the Simulink model — we use *FMI Toolbox* for MATLAB/Simulink with the *FMI Coder* Addon from *Modelon*[1] — does not support exporting of source code FMUs, only binaries. As such, we use Dymola, whose FMU export capabilities support exporting source code FMUs. Once generated, the FMU is imported into 20-sim 4C via the 20SimParser that parses the source code FMU and creates a valid 20-sim 4C project. Once configured, 20-Sim 4C is able to cross compile the source code for the ARM architecture, which is required to push the code onto a Raspberry Pi 3 board (see Fig. 3) as a periodic task. The board is equipped with Raspbian Linux OS and the Xenomai 2.6.5 framework to enable real-time capabilities.

To enable full HiL capabilities, it is necessary to generate a toolwrapper FMU that acts as interface between the co-simulation FMU and the FMU running on the Raspberry Pi. This enables executing the controller code on an actual hardware platform while maintaining the plant (and its thermal physics models) running in a simulation.

[1] `http://www.modelon.com/products/fmi-tools/fmi-toolbox-for-matlabsimulink/`

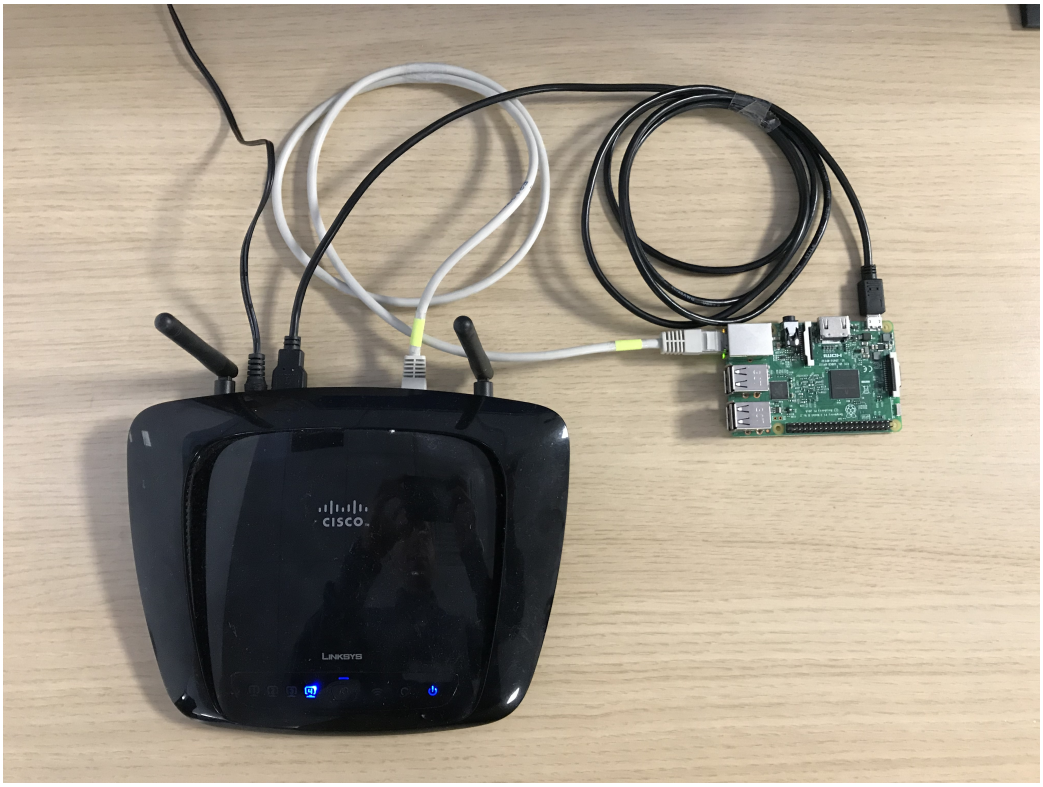

Figure 3: The Raspberry Pi 3 board used for HiL co-simulation.

2.2 FCU Test Model

This test model is developed for HiL co-simulation and corresponds to the FCU controller in Dymola as shown in Fig. 2. It is a PI controller and working in the cooling mode. From this aspect, the controller as well as the test model in this document are much different from the controller and the test model in the FCU pilot study which is given in Sect. 4 of the deliverable D3.6 [MGP+17].

In this test model, test automation is applied to the FCU controller. Note, however, that the plant is not included in the test model. Therefore, the system under test (SUT) of this test model is the PI controller, which is originally implemented in Dymola as a continuous-time PI controller. In order to model it in RT-Tester, its specification is discretized and implemented as a state machine.

Subsequently, we present the test model in SysML, and test input simulation by a state machine diagram.

2.2.1 Test Model

Similar to other test models, this SysML model consists of a SUT and a test environment (TE) as shown in Fig. 4. These components are used to represent the different building blocks of model-based testing: The SUT component represents the specified behavior of the controller, whereas the TE component describes environmental constraints. These components are represented by the blocks `SystemUnderTest` and `TestEnvironment`. Both components have two ports with `Observables` and `Stimuli` interfaces, respectively. The connection diagram between them is illustrated in Fig. 5.

2.2.2 `SystemUnderTest` Component

In Fig. 5, `Stimuli` represents the inputs to `SystemUnderTest`, namely `RAT_sp` and `Room_Temp`. Likewise, `Observables` denotes the outputs from `SystemUnderTest`, namely `cool_valve` and `VSD_Fan`. The meaning of these signals is described in Sect. 2.1. The `SystemUnderTest` consists only of a `Controller` component. The architecture structure diagram of `SystemUnderTest` is shown in Fig. 6. The state machine diagram of the

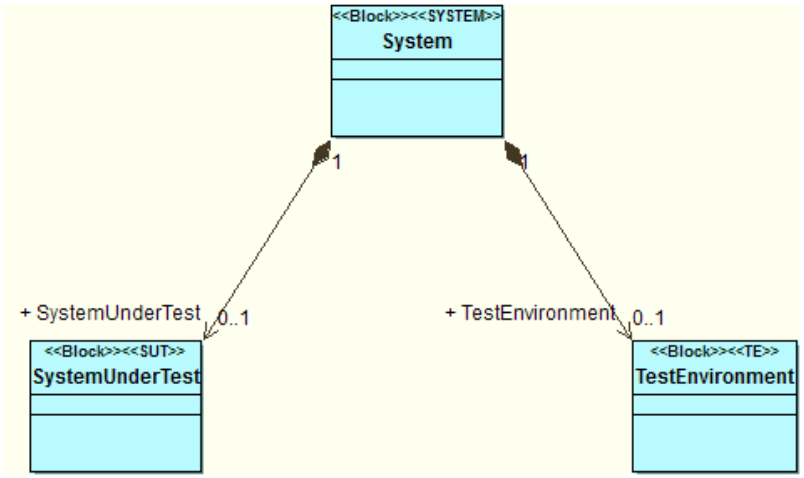

Figure 4: Architecture structure diagram of the FCU test model

FCU controller, implemented as the state machine `fcuCTRL` is illustrated in Fig. 7.

In the diagram,

- `Setup` is used to initialize constant variables;
- After `Setup`, the state machine resides in the `Waiting` state most of the time;
- Then every one second, it starts to calculate PID variables again:
  - At first, the inputs `RAT` and `RATSP` are copied to local variables to make sure all subsequent computations refer to same values of `RAT` and `RATSP`.
  - Then, `uP`, `uI`, and other local variables are computed for future use.
  - Next, the multiply of the gain `K` and the summation of `uP` and `uI` is calculated and assigned to `y0`.
  - Subsequently, a limiter is applied to `y0` to get the final output `y`.
  - Finally, `y0` and `y` are used to compute the gain track `uG`, and `y` is used to compute the output `cool_valve` and `VSD_Fan` respectively in `PostUpdate`.

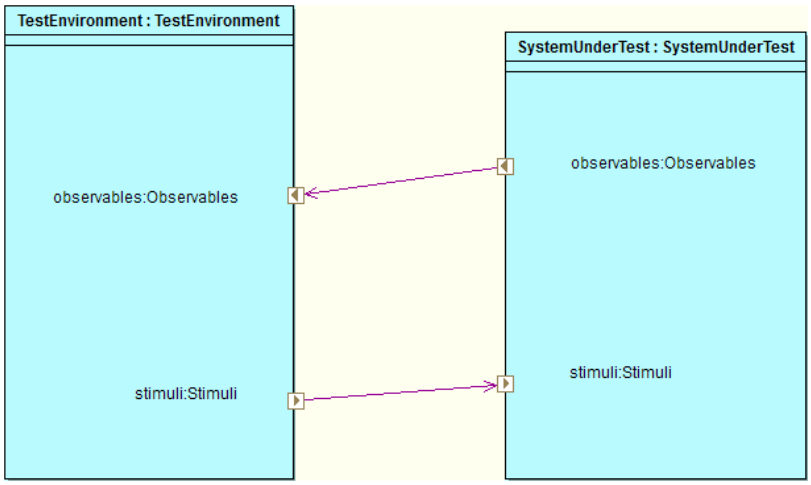

Figure 5: Connection diagram of the FCU test model

Figure 6: Architecture structure diagram of the `SystemUnderTest` component

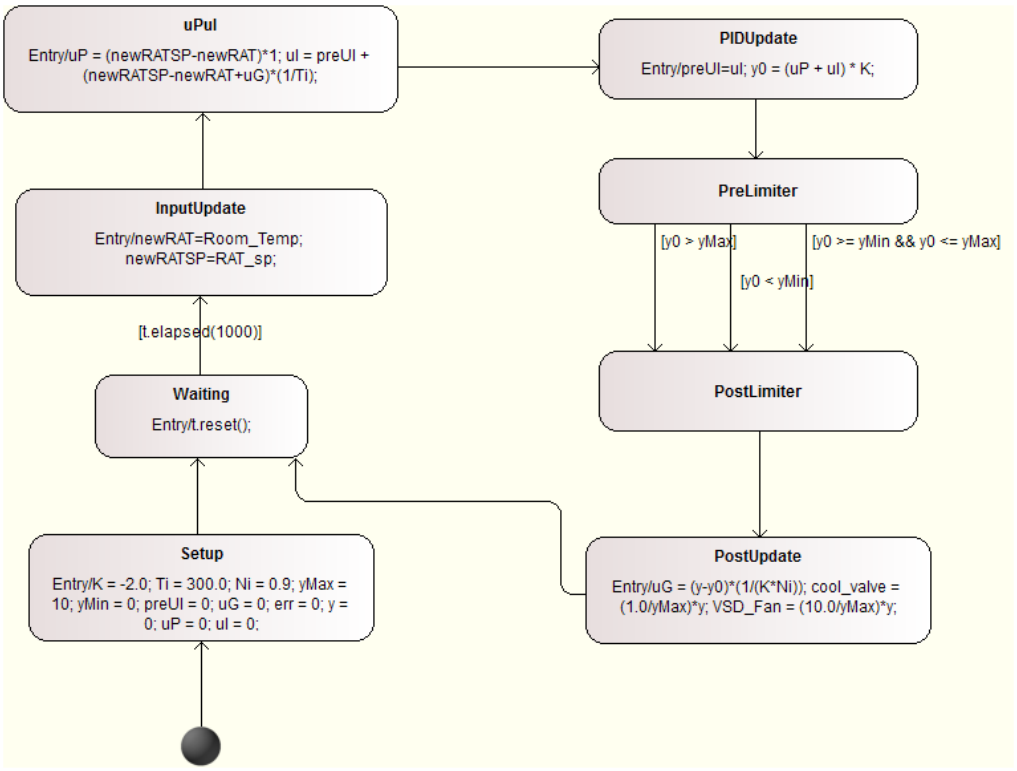

Figure 7: State machine diagram of `fcuCTRL`

2.2.3 `TestEnvironment` Component

In the beginning, the automatically generated test cases from a test model using RTT-MBT did not match up with our expectations because the environment description was too coarse, which resulted in unrealistic behavior of the environment. Frequent discontinuities in temperature changes, which usually evolve continuously, are an exemplar of such a unrealistic behavior. In order to provide a reasonable input to co-simulation, it is possible to encapsulate a list of real data in an FMU using tools such as 20-sim or OpenModelica. However, RTT-MBT cannot automatically include this data in the test case generation process, which is based on SMT solving. However, RTT-MBT allows to use a state machine to specify the input sequence in `TestEnvironment` of the test model.

We used this mechanism to specify the input sequence by defining a state machine in a block `TESim` of the `TestEnvironment` component. The architecture structure diagram of `TestEnvironment` is given in Fig. 8.

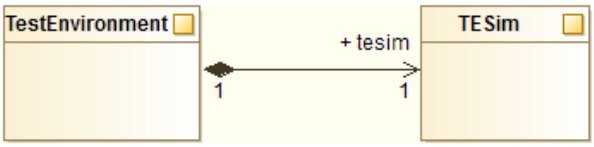

Figure 8: Architecture structure diagram of `TestEnvironment` component

The state machine diagram of `TESim` is shown in Fig. 9. Initially, `Room_Temp` is set to 30°C and `RAT_sp` to 22°C. Then, every minute `Room_Temp` is decreased by 0.4°C till it is 1°C below `RAT_sp`. This will take fifty minutes. After that, every ten minutes `RAT_sp` is decreased by 1°C and `Room_Temp` is increased by 2.5°C to make `Room_Temp` still slightly higher than `RAT_sp`. Therefore, the outputs of `cool_valve` and `VSD_Fan` are activated.

The specified input sequence by the state machine diagram and expected outputs are illustrated in Fig. 10, which depicts both, the inputs (`RAT_sp` and `Room_Temp`) and the outputs (`VSD_Fan` and `cool_valve`). Since the FCU is working in the cooling mode, `Room_Temp` is set to be higher than `RAT_sp` in order to activate the valve and the fan.

With this test input, test automation could evaluate how the SUT and the test model response to it, including the opening of the valve and the speed of the fan, and compare the difference from their behaviours.

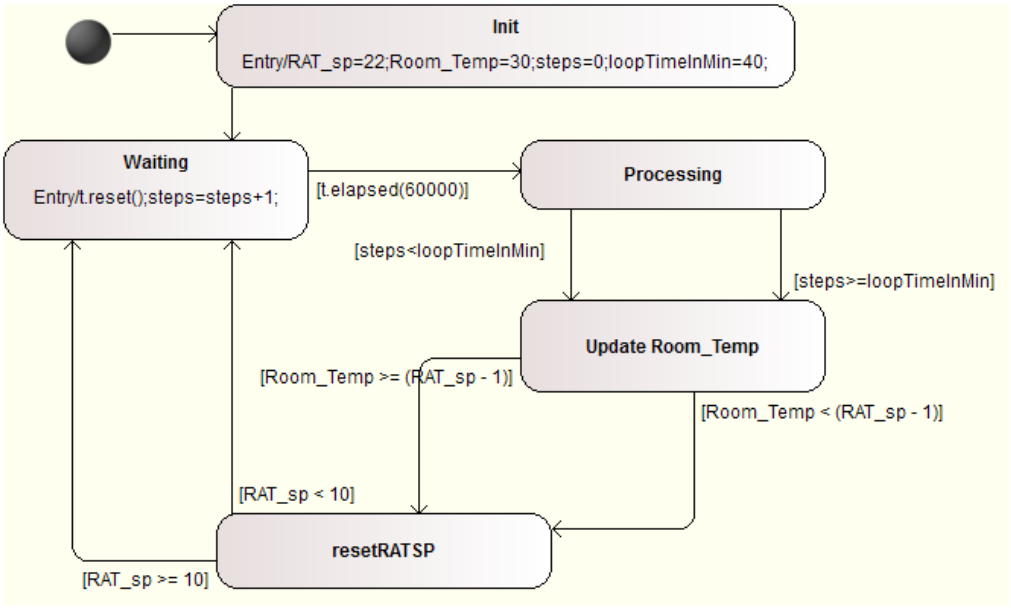

Figure 9: State machine diagram of `TESim` component

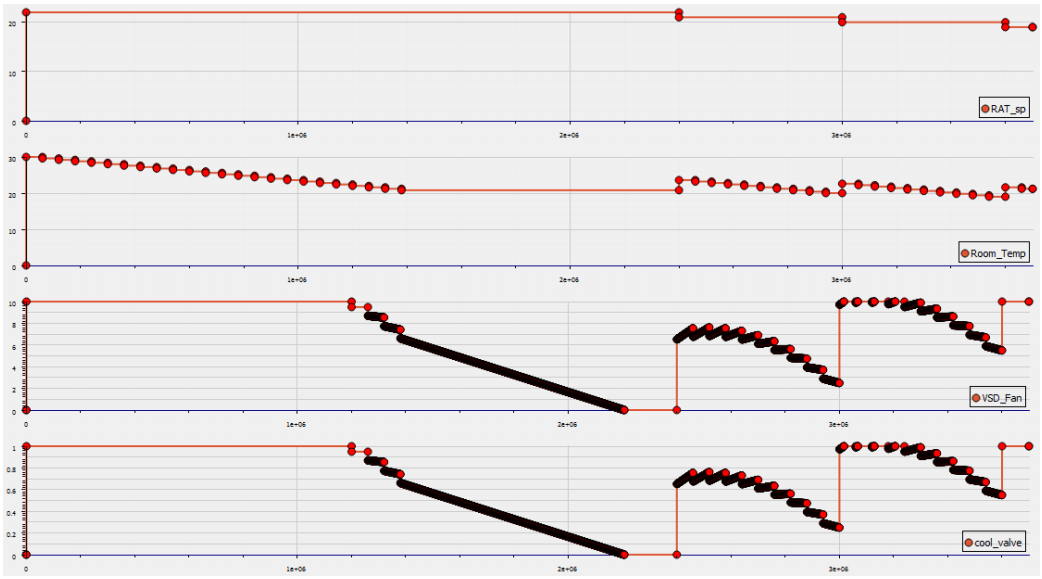

Figure 10: Sequence of input values for the room air temperature set point (`RAT_sp`), the room air temperature (`Room_Temp`), and the speed of the FCU fan (`VSD_Fan`) and the opening of the fan coil valve (`cool_valve`) generated from the test model using RTT-MBT. See Sect. 2.1 for a description of the units of the signals.

3 Simulation Details

The approach described in this document applies HiL testing as described in Sect. 2.2 to the SUT described in Sect. 2.1, which is a different scenario compared to what is described in [MGP+17]. The expected behavior of the SUT from the perspective of the generated test model is depicted in Fig. 10 and has already been discussed in the previous section.

By way of contrast, this section focusses on the execution results, how the reactions of the real FCU running on the Raspberry deviate from the expected behavior, and how this affects the testing verdicts. It comes to no surprise that the behavior of the FCU controller based on the Dymola running on the Raspberry Pi, which is shown in Fig. 11, deviates from the expected behavior. Even though both the implementation model and the test model are based on the same specification, they have been implemented by different engineers using different modeling formalism. Whereas the Dymola-based implementation is based on a continuous-time model, the test model is based on a discrete-event abstraction expressed as a collection of state machines. It is therefore interesting to observe that the overall tests have passed and no failures have been detected. However, there are still deviations that warrant discussion.

Clearly, the shapes of the concrete signal flows for the output signals `VSD_Fan` and `cool_valve` appear very similar to the expected outputs given in Fig. 10. There are some significant deviations though. For example, the expected value of `VSD_Fan` in the middle of the graph drops to 0, whereas the FCU actually sets it to a value in the order of `0.8` units. For an analysis, a test engineer then has to examine the log files, which are automatically generated by the RT-Tester test system, and evaluate in detail whether the deviations in the actual behavior from the expected behavior actually indicates potential defects in the SUT. An excerpt of such a log file, which stores the timestamps at which certain signal values have been observed, is given in Fig. 12. It may then be necessary to update the test configuration so that only smaller signal latencies and tolerances are allowed.

4 Related Work

HiL simulation and testing has been applied within a variety of distinct use-cases and areas, where simulation as well as test validation is involved, supporting rapid prototyping as well as — ideally — improved cost effectiveness.

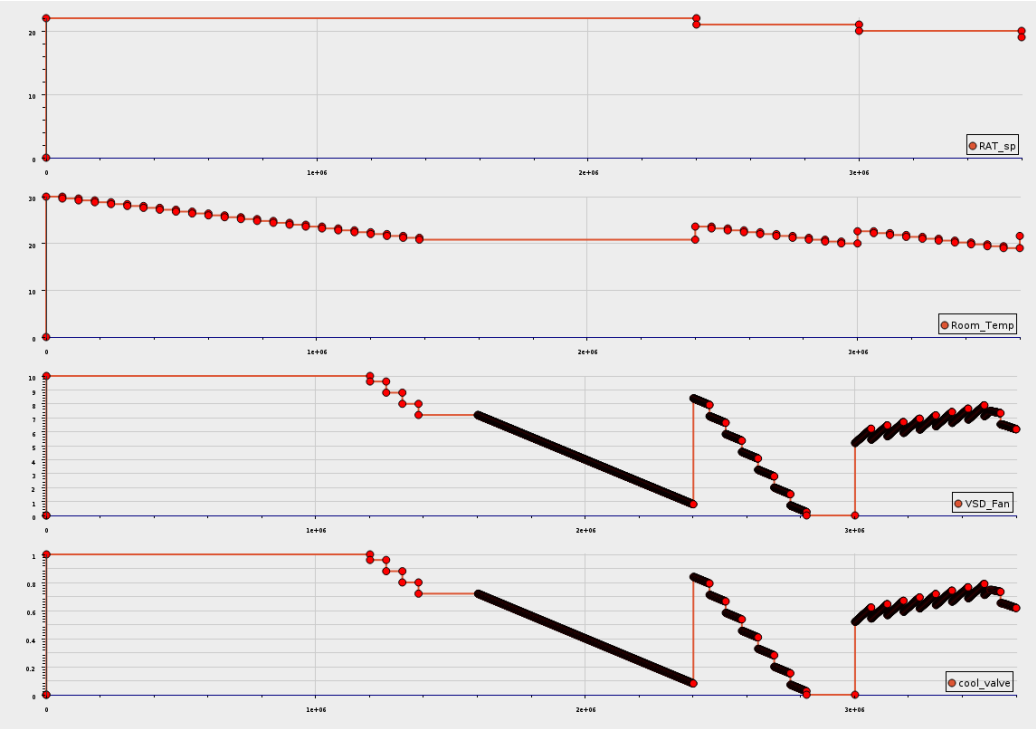

Figure 11: Sequence of interface values observed from executing the FCU on the Raspberry Pi against the generated test data, which shows how the interface values evolve over time. See Sect. 2.1 for a description of the units of the signals.

```
timestep 1716000 {
 "VSD_Fan" 6.282869
}

timestep 1717000 {
 "VSD_Fan" 6.274869,
 "cool_valve" 0.627487
}

timestep 1718000 {
 "VSD_Fan" 6.266869
}
```

Figure 12: Excerpt of the signal log generated from the concrete test execution

Below we highlight the different areas as well as general HiL technologies together with tools, which support such testing during prototyping.

Areas of HiL research include automotive [GPDSV06, FFHS06a], Unmanned-Aerial-Vehicles (UAVs) [CCLD09], wind energy systems [LSS+06], electric drives [Bou08, SKP12]. In [FFHS06b] they discuss key facets of enabling such HiL simulation during development, where they in particular consider how a control validation tool is integrated as part of a system development approach. The facets these authors highlight include hardware/software integration, proper modelling as well as sensor and actuator fidelity. Similarly, in [Han96] the authors approach HiL simulation as part of a specific tool set, and the authors in [ISS99] present HiL simulation as part of design and testing of engine control systems. Hence in all of the above work cited, a combination of co-simulation (HiL in this case) together with test cases plays a key enabling technology as a means of fast prototyping towards a realisation of a cyber-physical system.

Since it can be important to have HiL capabilities integrated within the same tool chain, commercial tools such as Matlab/Simulink[2], dSpace[3] and RT-Tester[4] support such efforts within their tool. In general, a drawback for HiL simulation can be that models cannot be reused between different tool chains unless they standardise and agree upon model simulation semantics between the different involved tools. This is, however, one of the goals of the FMI 2.0 standard [Blo14], which supports such facets of reusability between models within a HiL setting. Hence using our INTO-CPS co-simulation FMI engine, where similar usage of FMI is applied compare to works such as [BMK+16, BCWS11, CS12], we demonstrate integration of co-simulation together with testing capabilities within a HiL simulation setup.

An attempt to standardize the interface between tools and models for HiL scenarios is made by the ITEA project *Advanced Co-Simulation Open System Architecture* (ACOSAR)[5]. Similar to the FMI standard, an *Advanced Co-Simulation Interface* (ACI) is being defined, which contains a protocol for exchanging messages between *Advanced Co-Simulation Units* (ACUs), and a communication API. State-of-the-art for different domains, simulation scenarios (discrete-event, continuous, hybrid, real-time), languages, tools or

[2]See `https://se.mathworks.com/help/physmod/simscape/ug/hardware-in-the-loop-simulation-workflow.html`

[3]See `https://www.dspace.com/en/inc/home/products/systems/ecutest.cfm`

[4]See `https://www.verified.de`

[5]See `http://www.acosar.eu/`

standards was collected in an ACOSAR deliverable document [LRV+16]. Release of the initial public version of the ACI standard is planned for mid 2018. Similar to FMI, the main motivation for developing the ACI standard is the reuse of interface definitions, substitution of tools with and models with real test benches, and exchangeability of communication protocols.

5 Concluding Discussion & Future Work

This deliverable shows that the INTO-CPS tool-chain is feasible for the challenging task of HiL simulation and testing, which is generally challenging due to complex (and sometimes unstable) interfaces to embedded devices and rather unpredictable timings. Most importantly, we have shown that it is possible to combine the different tools from the INTO-CPS project to straightforwardly perform HiL simulation and testing, simply by exchanging the FMUs that communicate with each other via the COE.

In other deliverables of the INTO-CPS project, it has already been shown that the INTO-CPS approach allows simulation of different components from different tools and modelling formalisms. This deliverable thus provides a missing piece by showing the feasibility of the framework to the combination of HiL testing/simulation and purely simulated components.

Co-simulation and testing of systems whose components are encapsulated via FMUs [PBL+17] provides compelling directions for future work. Industrial-scale test benches are often suffering from the fact that some tasks have to be executed at a very high frequency, but producing little output, whereas other tasks only need to be performed sporadically or with a much lower period. A real-world example of such a scenario is the detection of analogue audio connections between handset phones involved in a system. Technically, the detection can be achieved via real-time Fourier analysis. While there is significant amounts of analogue input data and computation time involved, the output data is changing only very infrequently; connections between handsets are not built up and closed in the order of milliseconds, but rather seconds or even minutes.

Industrial test benches are often composed of different *nodes*, where some of these nodes are dedicated only to the computationally expensive tasks. It turns out to be possible to wrap the test and simulation functionality running on these high-performance nodes in FMUs containing a COE themselves. One high-performance node then executes one single top-level FMU, which

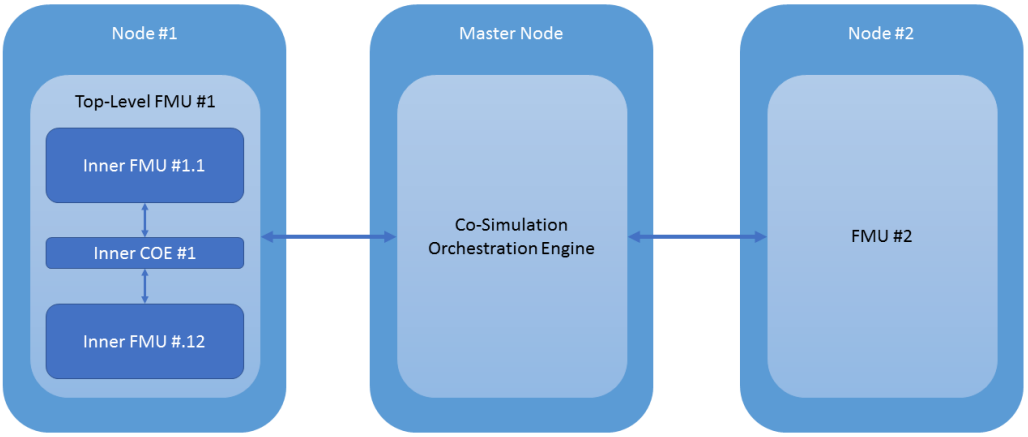

Figure 13: Possible FMI-based setup for the integration of high-frequency and low-frequency FMUs in test benches.

internally executes a COE with several inner FMUs at a high frequency. Other nodes in the test bench may be similarly structured, but the exchange of information between the different top-level FMUs is performed at a much lower frequency. This situation is depicted in Fig. 13. Node #1 is internally running a COE encapsulated in an FMU (Top-Level FMU #1), which is communicating with FMU #2 running on Node #2 via a COE running on a master node. Inner COE #1 communicates internally with Inner FMU #1 and Inner FMU #2 at a high frequency, whereas the external communication with FMU #2 takes place at a much lower frequency using the COE running on the master node. The application of such architectures for HiL co-simulation and testing appears extremely promising for industrial-scale test benches, and also supports the separation of concerns.

References

[BCWS11] Jens Bastian, Christoph Clauss, Susann Wolf, and Peter Schneider. Master for Co-Simulation Using FMI. In *8th International Modelica Conference*, 2011.

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

[BMK+16] Ròbert Lajos Bücs, Luis Murillo, Ekaterina Korotcenko, Gaurav Dugge, Rainer Leupers, Gerd Ascheid, Andreas Ropers, Markus Wedler, and Andreas Hoffmann. Virtual hardware-in-the-loop co-simulation for multi-domain automotive systems via the functional mock-up interface. In *Languages, Design Methods, and Tools for Electronic System Design*, pages 3–28. Springer, 2016.

[Bou08] Alain Bouscayrol. Different types of hardware-in-the-loop simulation for electric drives. In *Industrial Electronics, 2008. ISIE 2008. IEEE International Symposium on*, pages 2146–2151. IEEE, 2008.

[CAB+17] Luis Diogo Couto, Pasquale Antonante, Stylianos Basagiannis, Sara Falleni, Hassan Ridouane, Hajer Saada, and Erica Zavaglio. Building Case Study 3, (Confidential). Technical report, INTO-CPS Confidential Deliverable, D1.3d, December 2017.

[CCLD09] Guowei Cai, Ben M Chen, Tong H Lee, and Miaobo Dong. Design and implementation of a hardware-in-the-loop simulation system for small-scale uav helicopters. *Mechatronics*, 19(7):1057–1066, 2009.

[CS12] Patrick Chombart and Dassault Systèmes. Multidisciplinary modeling and simulation speeds development of automotive systems and software. *ITEA2 innovation report*, 2012.

[FFHS06a] Hosam K Fathy, Zoran S Filipi, Jonathan Hagena, and Jeffrey L Stein. Review of hardware-in-the-loop simulation and its prospects in the automotive area. *Ann Arbor*, 1001:48109–2125, 2006.

[FFHS06b] Hosam K. Fathy, Zoran S. Filipi, Jonathan Hagena, and Jeffrey L. Stein. Review of Hardware-in-the-Loop Simulation and its Prospects in the Automotive Area. In *Proc. SPIE 6228,*

Modeling and Simulation for Military Applications, May 2006. http://dx.doi.org/10.1117/12.667794.

[GPDSV06] Olaf Gietelink, Jeroen Ploeg, Bart De Schutter, and Michel Verhaegen. Development of advanced driver assistance systems with vehicle hardware-in-the-loop simulations. *Vehicle System Dynamics*, 44(7):569–590, 2006.

[Han96] Herbert Hanselmann. Hardware-in-the-loop simulation testing and its integration into a cacsd toolset. In *Computer-Aided Control System Design, 1996., Proceedings of the 1996 IEEE International Symposium on*, pages 152–156. IEEE, 1996.

[ISS99] R Isermann, J Schaffnit, and S Sinsel. Hardware-in-the-loop simulation for the design and testing of engine-control systems. *Control Engineering Practice*, 7(5):643–653, 1999.

[LRV+16] Leonid Lichtenstein, Florian Ries, Michael Völker, Jos Höll, Christian König, Josef Zehetner, Oliver Kotte, Isidro Coral, Lars Mikelsons, Nicolas Amringer, Steffen Beringer, Janek Jochheim, Stefan Walter, Corina Mitrohin, Natarajan Nagarajan, Torsten Blochwitz, Desheng Fu, Timo Haid, Jean-Marie Quelin, Rene Savelsberg, Serge Klein, Pacome Magnin, Bruno Lacabanne, Victor Schreiber, Martin Krammer, Nadja Marko, Martin Benedikt, Stefan Thonhofer, Georg Stettinger, Markus Tranninger, and Thies Filler. Literature Review in the fields of Standards, Projects, Industry and Science. Technical report, Acosar Public Deliverable, D1.1, Juli 2016.

[LSS+06] Hui Li, Michael Steurer, KL Shi, Steve Woodruff, and Da Zhang. Development of a unified design, test, and research platform for wind energy systems based on hardware-in-the-loop real-time simulation. *IEEE Transactions on Industrial Electronics*, 53(4):1144–1151, 2006.

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

[PBL+17] Adrian Pop, Victor Bandur, Kenneth Lausdahl, Marcel Groothuis, and Tom Bokhove. Final Integration of Simulators in the INTO-CPS Platform. Technical report, INTO-CPS Deliverable, D4.3b, December 2017.

[Pel13] Jan Peleska. Industrial-Strength Model-Based Testing - State of the Art and Current Challenges. *Electronic Proceedings in Theoretical Computer Science*, abs/1303.1006:3–28, 2013.

[PVL11] Jan Peleska, Elena Vorobev, and Florian Lapschies. Automated Test Case Generation with SMT-Solving and Abstract Interpretation. In Mihaela Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, editors, *Nasa Formal Methods, Third International Symposium, NFM 2011*, pages 298–312, Pasadena, CA, USA, April 2011. NASA, Springer LNCS 6617.

[SKP12] Thomas Schulte, Axel Kiffe, and Frank Puschmann. Hil simulation of power electronics and electric drives for automotive applications. In *Electronics*, volume 12, pages 130–135, December 2012.

A List of Acronyms

20-sim	Software package for modelling and simulation of dynamic systems
ACA	Automatic Co-model Analysis
AST	Abstract Syntax Tree
AU	Aarhus University
BDD	Binary Decision Diagram
BMC	Bounded Model Checking
CLE	ClearSy
CLP	Controllab Products B.V.
COE	Co-simulation Orchestration Engine
CPS	Cyber-Physical Systems
CT	Continuous-Time
DE	Discrete Event
DESTECS	Design Support and Tooling for Embedded Control Software
DSE	Design Space Exploration
FMI	Functional Mockup Interface
FMI-Co	Functional Mockup Interface – for Co-simulation
FMI-ME	Functional Mockup Interface – Model Exchange
FMU	Functional Mockup Unit
HiL	Hardware-in-the-Loop
LTL	Linear Temporal Logic
MC	Model Checking
RTT-MBT	RT-Tester Model Based Test Case Generator
SAT	SATisfiable Boolean formula, a symbolic representation of terms that can/should evaluate to *true*
SMT	Satisfiability Modulo Theories, i.e., a SAT formula interpreted over a logical theory (here, this describes a system design)
ST	Softeam
SysML	Systems Modelling Language
TWT	TWT GmbH Science & Innovation
UML	Unified Modelling Language
UNEW	University of Newcastle upon Tyne
UTRC	United Technologies Research Center
UY	University of York
VSI	Verifdied Systems International
WP	Work Package
XML	Extensible Markup Language