INtegrated TOol chain for model-based design of CPSs

The INTO-CPS Manifesto

Version: 0.01

Date: October, 2018

The INTO-CPS Association

http://into-cps.org

Contributors:

Peter Gorm Larsen, Aarhus University
John Fitzgerald, University of Newcastle
Jim Woodcock, University of York
Christian König, TWT
Stylianos Basagiannis, UTRC
Etienne Brosse, Softeam
Cláudio Gomes, University of Antwerp
José Cabral, Fortiss
Hugo Macedo, Aarhus University
Andrey Sadovykh, Softeam

Editors:

Peter Gorm Larsen, Aarhus University

Document History

Ver	Date	Author	Description
0.01	23-04-2018	Peter Gorm Larsen	Initial version.

Abstract

John Fitzgerald

Contents

1 Introduction **7**

2 Challenges in Engineering CPSs **9**

3 INTO-CPS in a Nutshell **10**
3.1 How INTO-CPS works 11
3.2 Case studies 13
3.3 The INTO-CPS foundations 15
3.4 The INTO-CPS methods and guidelines 15

4 The INTO-CPS Foundations **17**

5 The INTO-CPS Method and Guidelines **18**

6 The INTO-CPS Tool Chain **19**
6.1 Modelio 19
6.2 Modeling tools 20
6.3 RT Tester 21
6.4 3D animation 21
6.5 The INTO-CPS Application 21

7 The INTO-CPS Industrial Case Studies **22**

8 Related Work **23**

9 Future Directions **24**

A List of Acronyms **27**

B Underlying Principles **29**
B.1 Co-simulation 29
B.2 Design Space Exploration 30
B.3 Model-Based Test Automation 31
B.4 Code Generation 31

C Background on the Individual Tools **33**
C.1 Modelio 33
C.2 Overture 34
C.3 20-sim 36
C.4 OpenModelica 37
C.5 RT-Tester 38

C.6 4DIAC 40
C.7 AutoFOCUS-3 40

1 Introduction

Systems composed of closely coupled computing and physical elements are becoming increasingly important in the modern world. For society and citizens, the potential of such smart systems to deliver more efficient, sustainable and resilient services is enormous. Such Cyber-Physical Systems (CPSs) are characterised by a complex architecture and a design process involving several diverse science and engineering disciplines. In the interface between disciplines, different formalisms and technical cultures meet, and the traditional approaches for designing systems vary significantly among the relevant fields. The developer of a CPS faces a large design space that is hard to cover with hardware prototypes due to the high cost of their implementation. Common workflows to assist CPS engineers, and the necessary tools, are currently lacking.

The design of CPSs involves the usage of results obtained using a combination of different formalisms serving different engineering disciplines. Compounding the variety of formalisms, the formalisms are usually associated with different tools. Either as different versions, or supported by different entities.

In this manifesto we put forward the idea of combining different formalisms and respective supporting tools by means of a tool chain. A tool chain allows the practitioner to easily combine the results from the various fields during the design process. Our view is to keep the best of each of the tools and integrating them by building an environment automating the tool chain interaction, accessibility, and unifying the entry point to it.

In such unified environment of several tools, users keep their usual patterns of interaction with their familiar tool or tools available in the tool chain. Instead of having to learn a new formalism...

But, on the other hand, it is hard to master all the components of the tool chain. The mastering of tool chains involves the management of the whole set of models and inputs/results for each of the tools. In addition, the conversion of results, traceability of changes and keeping track of the user interactions need also to be taken into account. Moreover, given the evolving nature of each of the tools, the task of managing the tool chain while ensuring the dependencies of each of the tools and their interoperability becomes too complex.

Practice shows users are more receptive to a push-button approach when it is time to combine their results with the other tools.

The INTO-CPS builds upon a frontend application which was developed following the unified entry point idea. The INTO-CPS application allows the user to fetch the different tools, checkout model files from repositories, orchestrate the several tools run, and organise the results/interactions. This reduces the previously mentioned mastering complexity demands to a push-button effort.

2 Challenges in Engineering CPSs

John Fitzgerald

3 INTO-CPS in a Nutshell

To address the challenges presented above in Section 2, INTO-CPS project has created an integrated "tool chain" for comprehensive model-based design of CPSs. The tool chain supports multidisciplinary, collaborative modelling of CPSs from requirements, through simulation of multiple heterogeneous models that represent the physical elements as well as the computational parts of the system, down to realisation in hardware and software, enabling traceability at all stages of the development as outlined in figure **??**.

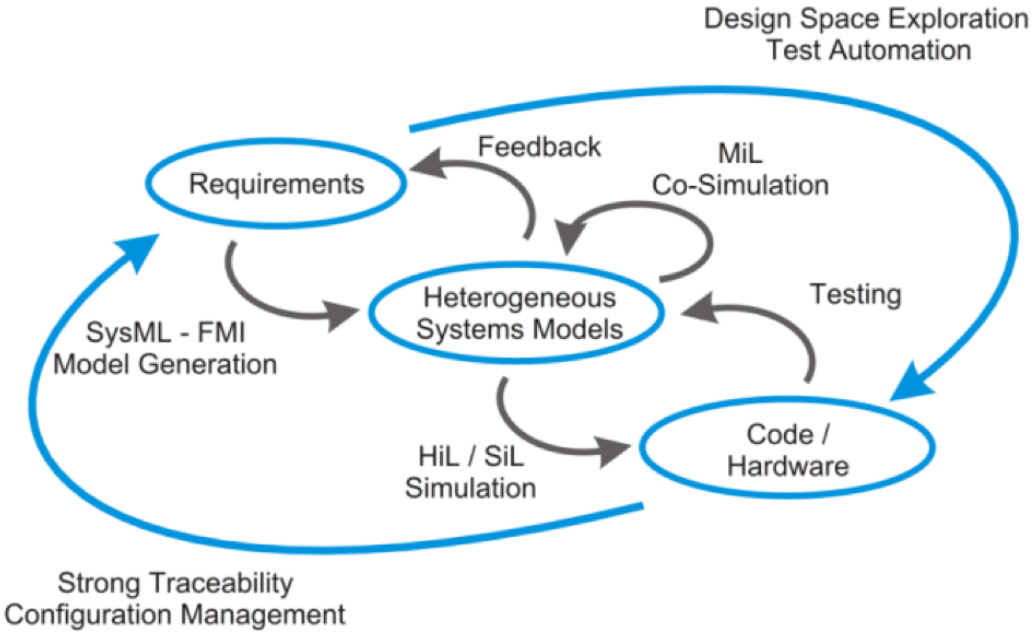

Figure 1: Connections in the INTO-CPS tool chain.

The goals of the INTO-CPS project have been to:

1. Build an open, well-founded tool chain for multidisciplinary model-based design of CPS that covers the full development life cycle of CPS.
2. Provide a sound semantic basis for the tool chain.
3. Provide practical methods in the form of guidelines and patterns that support the tool chain.
4. Demonstrate the effectiveness of the methods and tools in an industrial setting in a variety of application domains.

5. Form an INTO-CPS Association to ensure that results extend beyond the life of the project.

3.1 How INTO-CPS works

The INTO-CPS project had a consortium consisting of 11 partners (four universities, seven companies) who contribute with complementary knowledge, baseline technologies and applications. The baseline technologies support systems modelling (Modelio), modelling and simulation of physical systems (OpenModelica, 20-sim), discrete-event modelling and simulation (Overture), Co-Simulation (Crescendo, TWT Co-Simulation engine) and test automation (RT-Tester). These baseline technologies enable both descriptions of Discrete Event (DE) models as well as Continuous-Time (CT) models. Any number of such constituent models may be combined in a hybrid setting using the INTO-CPS technology. Advancing over technologies commonly used today in industry, INTO-CPS provides an open tool chain that enables the following:

1. Providing a faster route to market for CPS products where control aspects depend upon the development of physical elements (e.g. mechanical parts) that typically take a long time to be developed.
2. Avoiding vendor lock-in by having an open tool chain that can be extended and used in different ways. Although it is well-founded it is based on pragmatic principles where a trade-off between accuracy and speed of analysis is enabled.
3. Including capabilities for exploring large design spaces efficiently so that ?optimal? solutions can be found given the parameters that are important for the user, both on the cyber and the physical side.
4. Limiting the necessity for large amounts of expensive physical tests in order to provide the necessary evidence for the dependability of the CPS.
5. Enabling traceability of all project artefacts produced by different tools using an open traceability standard.

A Co-simulation Orchestration Engine (COE) has been built on the baseline technologies and in accordance with requirements driven by the industry case studies outlined below. This engine combines previous experience from TWT?s Co-Simulation engine and the Crescendo tool developed in the project Design Support and Tooling for Embedded Control Software

(DESTECS). The goals for the co-simulation orchestration engine include, among others, optimised scalability and performance, and data exchange between the different models facilitated by the Functional Mockup Interface (FMI). Interfaces to further tools will be provided so that the requirements and the different artefacts will be fully exploited. An INTO-CPS Application acting as a common front-end to the INTO-CPS tool chain has been produced using web-based technologies. This enables stakeholders without detailed knowledge on the different modelling technologies to experiment with alternative candidate designs and use systematic ways to either explore a large design space or systematically test heterogeneous models. The INTO-CPS Application, the COE and its most important connections are shown in Figure 2.

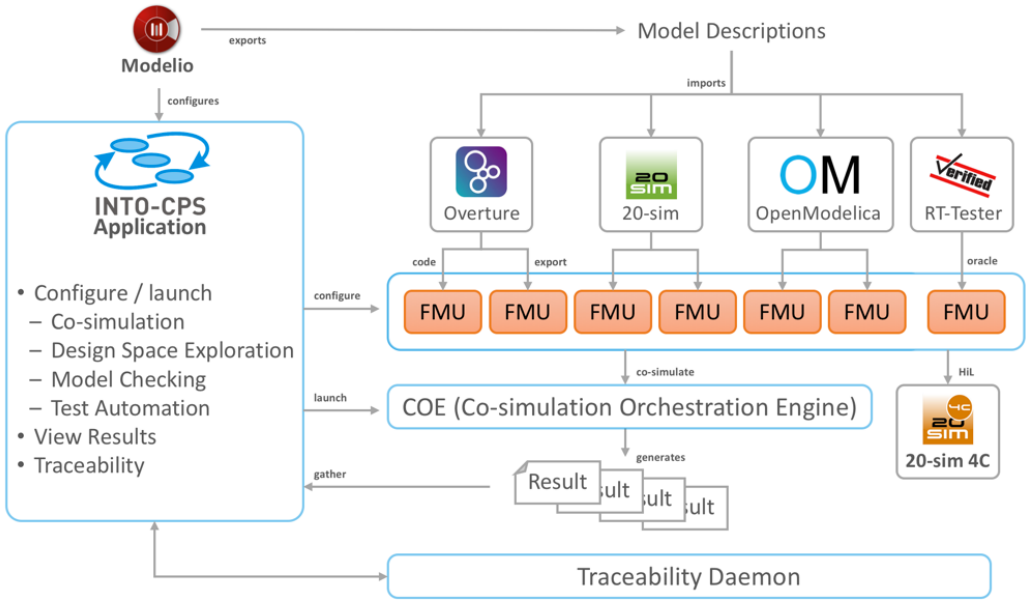

Figure 2: Overview of the INTO-CPS tool chain.

The COE connects multiple diverse models, each in the encapsulated form of a Functional Mock-up Unit (FMU) or running in its native modelling environment, to an overall system model. An algorithm for Design Space Exploration (DSE) enables sweeps through ranges of design parameters, performing co-simulations on each. System robustness can be evaluated by using Test Automation (TA) tools that can manipulate the simulation. Links to models can be kept in a database to allow for versioning and traceability. The entire co-simulation is controlled from a user interface. The final year of the project has seen a significant investment in promoting the co-simulation capabilities, and http://fmi-standard.org/tools/ has been set up as a core

webpage for all tools supporting FMI-based co-simulations.

During Year 3, the COE has been extended with capabilities both for distributed co-simulations and nested co-simulations. A graphical front-end for the entire tool chain called the INTO-CPS Application has been extended considerably. This has been developed further as a desktop application, using web technologies to enable a smoother transition to delivering this as an on-line service should this be desirable in the future . Building on the work of previous years, a 3D FMU support unit has been improved significantly, and has been used for multiple models, including those employed in the industrial case studies.

Year 3 have seen advances in many other areas of the technology, including new features for Design Space Exploration (DSE), Test Automation (TA), Model Checking (MC), traceability and Code Generation (CG). All of these features have been integrated in the INTO-CPS Application. The final version of the DSE support has been augmented with a capability to deploy DSE on cloud services, making DSE a technology that can be used by businesses without large-scale compute resources of their own. Our SysML profile, designed to ease the high-level development of multi-models, has been extended with features to support disciplined approaches to DSE. The TA and MC capabilities already incorporated in the INTO-CPS Application have been improved during Year 3. Finally, the CG features are incorporated in the different baseline modelling, and simulation tools have been improved in terms of performance and integration. The full deployment of the CG features in a distributed and embedded setting turned out to require more resources than originally envisaged. Most importantly, the support for traceability between different CPS artefacts has been developed during Year 3, including extensions of all the baseline tools with Open Source Life Cycle (OSLC) support. It also turned out that more effort was needed for the full traceability support compared to what was originally envisaged.

3.2 Case studies

Inside the INTO-CPS project four industry-led case studies from different application domains have allowed us to evaluate the final INTO-CPS tool chain:

In the Railways case study led by CLE, an innovative distributed interlocking solution has been developed, where signalling safety rules take both the logic and the physical conditions into account with a higher degree of indepen-

dence than normally. The challenge was to find the right trade-off between the efficiency of an interlocking system (availability of routes, trains' delays and cost of interlocking system) and safety (collision avoidance, derailment prevention, availability and efficiency of the emergency system).

The Agriculture case study led by AI concerns both an automated control system for an agricultural robot as well as an autonomously operating lawn mower. The robot, which provides more efficient removal of weeds in the field while operating safely but with minimal human interaction, was completed after Year 2 of the project. During Year 3 the goal was to experiment with the extent to which it would be possible to reuse model elements from Year 2 in the development of an autonomous control for the lawn mower. The challenge in both cases was to simulate the behaviour of physical components (such as mechanical loading on certain elements) together with controls of the automated system even before the physical mechanical components are available. These controls access local data (e.g. sensors) and external data (e.g. GPS). Model-based design allowed for accelerated time-to-market and virtual verification while reducing the need for multiple physical prototypes. The Year 3 case study also demonstrated that reuse of ideas was possible in the sense that using the INTO-CPS technology development is faster compared to the traditional alternative.

In the Building Automation case study led by UTRC, CPSs for control of Heating, Ventilation and Air-conditioning (HVAC) have been developed. These CPSs need to be adaptable to components of various manufacturers and different building patterns and the corresponding requirements. The challenge here is also to manage the complexity of the overall system in a way so the co-simulations are sufficiently scalable. The various parts that influence an HVAC system have been modelled and simulated, e.g. the fan-coil unit that distributes air, the buildings and rooms as well as the controllers of the fan-coil units. During Year 3 the team has been focusing on evaluating the scalability of the INTO-CPS co-orchestration engine to HVAC models of increased complexity. In the Year 3 use case, the chiller model has been introduced in order to challenge the efficiency and execution of co-simulations using INTO-CPS technology since we have experienced that even commercial tools often fail to deliver the required performance here. In addition, UTRC has run an extensive evaluation of all INTO-CPS features including DSE, test automation, Hardware-in-the-Loop (HiL) and 3D co-simulation.

In the Automotive case study led by TWT, a range optimisation assistant for electric vehicles is being developed. In order to maximise the range without compromising other qualities such as comfort or speed, a comprehensive as-

sessment of the vehicle and its environment is necessary. To achieve this goal, all relevant parts of the system have been modelled, e.g. battery, drive train, topography, traffic, weather and cabin thermal control. These constituent models are created in native industrial tools, such as Matlab, and coupled using the INTO-CPS tool suite.

In Year 3, the final INTO-CPS technologies have been used in all four case studies to get experience with these. In order to properly compare the INTO-CPS technology under development with existing modelling and simulation tools, some of the industrial case studies have, on purpose, developed some of their constituent models using such legacy tools (AI has used Gazebo, UTRC has used Dymola and TWT has used Matlab) in order to experiment with the FMUs exported from them in connection with the COE and the rest of the INTO-CPS tool chain. Generally speaking, the results have been quite positive. All four industrial case studies have been extended in different ways (compared to Year 2) and as documented in D1.3a [INTO-CPS-D1.3] progress has been achieved in the assessment of the industrial needs for all four industrial case studies.

3.3 The INTO-CPS foundations

The development of tools and methods in INTO-CPS is based on a sound semantic description of co-simulation. Our tools use VDM-RT as the discrete-event language and Modelica as the continuous-time language. The framework for co-simulation is based on FMI. In WP2, we have formalised and mechanised both languages in this framework using Isabelle/UTP. We have a semantics of the relevant parts of SysML that can be used with FMI. These foundations allow for the formal checking of the validity of analysis and co-simulation results. In Year 3, as well as completing the work on this formalisation, we have established a close integration with the industrial case studies from WP1 (in particular, the railways and FCU applications) and supported the development of the INTO-CPS tool chain in WP4 and WP5. We have shown how to use the foundational tools, with both theorem proving and model checking, to add value to the INTO-CPS tool chain.

3.4 The INTO-CPS methods and guidelines

Lowering the barriers to multidisciplinary model-based engineering of CPSs demands methods that permit the deployment of tools in industry processes,

embodied in guidelines that reflect experience gained using such methods. We present our modelling methods as guidelines for applying the INTO-CPS tool chain in real industry contexts, with a strong focus on supporting systematic DSE, our form of tradespace analysis, and on managing the traceability of design artefacts. All of the methods and guidelines materials have been made ready for subsequent use by the INTO-CPS Association.

In order to ease practical deployment of INTO-CPS technology, a SysML profile has been developed to enable designers to move more readily from abstract system models to the structure of heterogeneous co-models. Thus, it has been extended with the ability to help engineers describe explicitly the parameters, objectives and ranking involved in the DSE process, and to allow sweeps to be made both over parameters and operating scenarios. We have developed a Traceability Information Model (TIM) that supports the needs of heterogeneous CPS engineering teams. In defining permissible relations between artefacts and activities, we have drawn on two sources: Open Services for Lifecycle Collaboration (OSLC) and W3C PROV supported traceability links.

Our guidelines have been implemented in training materials and pilot studies which have been made publicly available and can readily be imported into the INTO-CPS Application, making it easy for newcomers to experiment with the INTO-CPS tools and methods. The pilots provide coverage of all INTO-CPS simulation technologies (VDM-RT, 20-sim and OpenModelica), have architectural models in SysML using the INTO-SysML profile, may be co-simulated with the INTO-CPS Application, can perform DSE, use code generation and have support for test automation.

4 The INTO-CPS Foundations

Jim Woodcock

5 The INTO-CPS Method and Guidelines

John Fitzgerald

6 The INTO-CPS Tool Chain

Christian König with support from Etienne Brosse

make sure that this section fits to the other sections, incl. references

indicate maturity of the different tool features, in particular wrt the new ones

This section discusses the interconnectivity of the different tools, and how the tools fit into the workflows and tasks that are covered by INTO-CPS. In particular, this section focuses on the features that were added during the INTO-CPS project, and in the framework of the INTO-CPS association. This section does *not* describe all the tools in detail (here, the reader is referred to the different manuals, and to the User Manual (ref to the INTO-CPS tool manual).

An overview of the different tools that form the tool-chain of the INTO-CPS association, is given in Figure 3.

[text on the overall tool-chain, refer to workflows, probably from previous section]

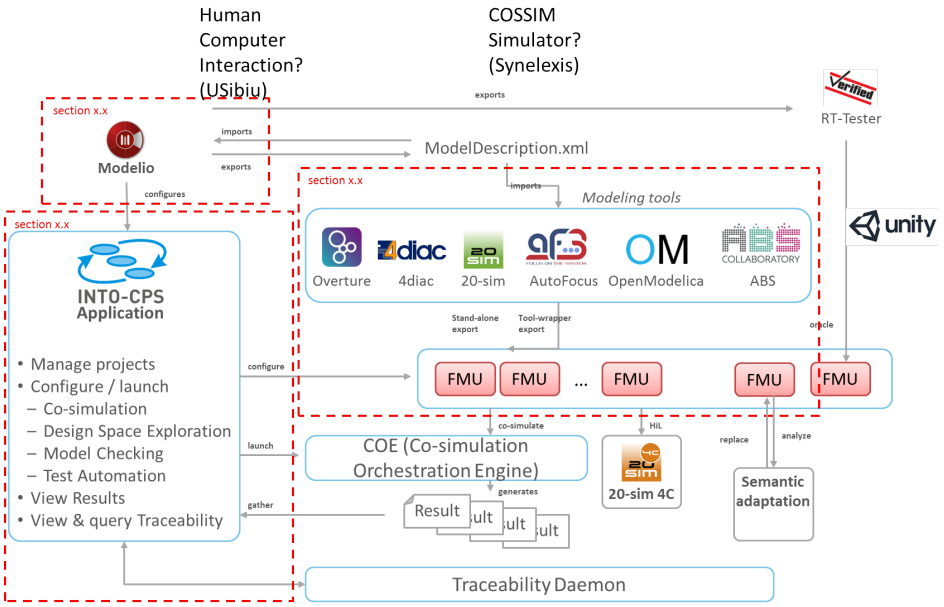

Figure 3: Overview of the different tools and their arrangement in a tool-chain.

6.1 Modelio

Modelio is an open-source modeling environment for various formalisms with many interfaces to import and export models. In the context of INTO-CPS, the support for SysML modelling is of primary importance, while Modelio can be extended with a range of modules to enable more modelling languages. In the terminology of the methods guidelines (e.g. [**?**]), Modelio is a tool for the *architectural modeling* and for *requirements management*.

During the INTO-CPS project, a SysML profile was created, which is available as a module for Modelio 3.4 and 3.6 (see `http://forge.modelio.org/projects/intocps`). This INTO-CPS SysML profile extends Modelio with several functionalities that described in detail elsewhere (refs). Here, only those parts of the INTO-CPS SysML profile are discussed that add features for interconnectivity in the tool-chain.

To support the FMI multi-modelling approach, ModelDescription.xml files can now be imported into, and exported from a SysML Architectural Modeling diagram. Importing ModelDescription.xml files creates a SysML block with the corresponding flow ports and attributes, exporting them allows import in other modeling tools, such as those described below.

The Connections Diagram describes the signal flow between the different SysML blocks, which can each correspond to one FMU. Using the new INTO-CPS SysML profile, the Connections Diagram can be exported to an intermediary JSON format, which can then be imported by the INTO-CPS Application, to create a new Multi-Model.

DSE modeling configuration
Requirements management
Traceability

6.2 Modeling tools

Most modelling tools support the same functions. A ModelDescription.xml file (e.g. one that is automatically created from Modelio, see previous section) can be imported to create a skeleton model with the input and output signals and exposed parameters.

MD.xml import
FMU export

Tool	MD.xml import	FMU export (stand alone)	FMU export (tool wrapper)	Traceability
20-sim	yes	no	yes	yes
OpenModelica	yes	yes	no	yes
Overture	yes	yes	no	yes
4diac	N.A.	N.A.	N.A.	N.A
AutoFocus 3	N.A.	N.A.	N.A.	
Cossim	N.A.	N.A.	N.A.	
ABS	N.A.	N.A.	N.A.	

Table 1: Functionalities of the modeling tools

Traceability support

6.3 RT Tester

Import of xsd files from Modelio
FMU export of test oracles

6.4 3D animation

6.5 The INTO-CPS Application

Integration of the different artifacts
configuration and execution of Co-Simulation
Front-end for Test-automation and model checking
Viewer for traceability

7 The INTO-CPS Industrial Case Studies

Stylianos Basagiannis

8 Related Work

Claudio Gomes

Claudio: Scope: co-simulation frameworks.

9 Future Directions

Peter Gorm Larsen

References

[BHJ+06] Armin Biere, Keijo Heljanko, Tommi A. Juntilla, Timo Latvala, and Viktor Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5), 2006.

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

[Con13] Controllab Products B.V. http://www.20sim.com/, January 2013. 20-sim official website.

[Fav05] Jean-Marie Favre. Foundations of Model (Driven) (Reverse) Engineering : Models – Episode I: Stories of The Fidus Papyrus and of The Solarus. In *Language Engineering for Model-Driven Software Development*, March 2005.

[FE98] Peter Fritzson and Vadim Engelson. Modelica - A Unified Object-Oriented Language for System Modelling and Simulation. In *ECCOP '98: Proceedings of the 12th European Conference on Object-Oriented Programming*, pages 67–90. Springer-Verlag, 1998.

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

[GFR+12] Anand Ganeson, Peter Fritzson, Olena Rogovchenko, Adeel Asghar, Martin Sjölund, and Andreas Pfeiffer. An OpenModelica Python interface and its use in pysimulator. In Martin Otter and Dirk Zimmer, editors, *Proceedings of the 9th International Modelica Conference*. Linköping University Electronic Press, September 2012.

[KG16] C. Kleijn and M.A. Groothuis. *Getting Started with 20-sim 4.5*. Controllab Products B.V., 2016.

[KGD16] C. Kleijn, M.A. Groothuis, and H.G. Differ. *20-sim 4.6 Reference Manual*. Controllab Products B.V., 2016.

[KR68] D.C. Karnopp and R.C. Rosenberg. *Analysis and Simulation of Multiport Systems: the bond graph approach to physical system dynamic*. MIT Press, Cambridge, MA, USA, 1968.

[KS08] Daniel Kroening and Ofer Strichman. *Decision Procedures - An Algorithmic Point of View*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2008.

[LBF+10] Peter Gorm Larsen, Nick Battle, Miguel Ferreira, John Fitzgerald, Kenneth Lausdahl, and Marcel Verhoef. The Overture Initiative – Integrating Tools for VDM. *SIGSOFT Softw. Eng. Notes*, 35(1):1–6, January 2010.

[LLJ+13] Peter Gorm Larsen, Kenneth Lausdahl, Peter Jørgensen, Joey Coleman, Sune Wolff, and Nick Battle. Overture VDM-10 Tool Support: User Guide. Technical Report TR-2010-02, The Overture Initiative, www.overturetool.org, April 2013.

[Ope] Open Source Modelica Consortium. OpenModelica User's Guide.

[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.

[Pnu77] Amir Pnueli. The Temporal Logic of Programs. In *18th Symposium on the Foundations of Computer Science*, pages 46–57. ACM, November 1977.

[Ver13] Verified Systems International GmbH. RTT-MBT Model-Based Test Generator - RTT-MBT Version 9.0-1.0.0 User Manual. Technical Report Verified-INT-003-2012, Verified Systems International GmbH, 2013. Available on request from Verified System International GmbH.

[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

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
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
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	Softeam
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

B Underlying Principles

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

Claudio: I suggest that references to introductions, references, and so forth, be made in these subsections. Currently, the text does not cover each topic adequately, providing merely a summary. Also, there is a risk that these sections will repeat what is already described as part of the tools descriptions. As such, I suggest that these topics be moved to appendix A, coming before the tool descriptions...

B.1 Co-simulation

Co-simulation refers to the simultaneous simulation of individual models which together make up a larger system of interest, for the purpose of obtaining a simulation of the larger system. A co-simulation is performed by a co-simulation orchestration engine. This engine is responsible for initializing the individual simulations as needed; for selecting correct time step sizes such that each constituent model can be simulated successfully for that duration, thus preventing time drift between the constituent simulations; for asking each individual simulation to perform a simulation step; and for synchronizing information between models as needed after each step. The result of one such round of simulations is a single simulation step for the complete multi-model of the system of interest.

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

B.2 Design Space Exploration

During the process of developing a CPS, either starting from a completely blank canvas or constructing a new system from models of existing components, the architects will encounter many design decisions that shape the final product. The activity of investigating and gathering data about the merits of the different choices available is termed Design Space Exploration. Some of the choices the designer will face could be described as being the selection of parameters for specific components of the design, such as the exact position of a sensor, the diameter of wheels or the parameters affecting a control algorithm. Such parameters are variable to some degree and the selection of their value will affect the values of objectives by which a design will be measured. In these cases it is desirable to explore the different values each parameter may take and also different combinations of these parameter values if there are more than one parameter, to find a set of designs that best meets its objectives. However, since the size of the design space is the product of the number of parameters and the number of values each may adopt, it is often impractical to consider performing simulations of all parameter combinations or to manually assess each design.

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

Once a simulation has been performed, there are two steps required to close the loop. The first is to analyze the raw results output by the simulation to determine the value for each of the objectives by which the simulations are to be judged. Such objective values could simply be the maximum power consumed by a component or the total distance traveled by an object, but they could also be more complex measures, such as the proportion of time a device was operating in the correct mode given some conditions. As well as numerical objectives, there can also be constraints on the system that are either passed or failed. Such constraints could be numeric, such as the

maximum power that a substation must never exceed, or they could be based on temporal logic to check that undesirable events do not occur, such as all the lights at a road junction not being green at the same time.

The final step in a closed loop is to rank the designs according to how well each performs. The ranking may be trivial, such as in a search for a design that minimizes the total amount of energy used, or it may be more complex if there are multiple objectives to optimize and trade off. Such ranking functions can take the form of an equation that returns a score for each design, where the designs with the highest/lowest scores are considered the best. Alternatively, if the relationship between the desired objectives is not well understood, then a Pareto approach can be taken to ranking, where designs are allocated to ranks of designs that are indistinguishable from each other, in that each represents an optimum, but there exist different tradeoffs between the objective values.

B.3 Model-Based Test Automation

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

B.4 Code Generation

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

The relationship that must be maintained between the source model and translated program must be one of refinement, in the sense that the translated program must not do anything that is not captured by the original

model. This must be considered when translating models written in high-level specification languages, such as VDM. The purpose of such languages is to allow the specification of several equivalent implementations. When a model written in such a language is translated to code, one such implementation is essentially chosen. In the process, any non-determinism in the specification, the specification technique that allows a choice of implementations, must be resolved. Usually this choice is made very simple by restricting the modelling language to an executable subset, such that no such non-determinism is allowed in the model. This restricts the choice of implementations to very few, often one, which is the one into which the model is translated via code generation.

C Background on the Individual Tools

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

C.1 Modelio

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

Among the extension modules, some are dedicated to IT system architects. For system engineering, SysML or MARTE modules can be used. They provide dedicated modelling support for dealing with general, software and hardware aspects of embedded or cyber physical systems. In addition, several utility modules are available, such as the Document Publisher which provides comprehensive support for the generation of different types of document.

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

C.2 Overture

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

Overture supports the following activities:

- The definition and elaboration of syntactically correct specifications in any of the three dialects, via automatic syntax and type validation.
- The inspection and assay of automatically generated proof obligations which ensure correctness in those aspects of specification validation which can not be automated.
- Direct interaction with a specification via an execution engine which can be used on those elements of the specification written in an executable subset of the language.
- Automated testing of specifications via a custom test suite definition language and execution engine.
- Visualization of test coverage information gathered from automated testing.
- Visualization of timing behaviours for specifications incorporating timing information.
- Translation to/from UML system representations.
- For specifications written in the special executable subset of the language, obtaining Java implementations of the specified system automatically.

For more information and tutorials, please refer to the documentation distributed with Overture.

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

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

VDM++ The VDM++ dialect supports a specification style inspired by object-oriented programming. In this specification paradigm, a system is understood as being composed of entities which encapsulate both state and behaviour, and which interact with each other. Entities are defined via templates known as *classes*. A complete system is defined by specifying *instances* of the various classes. The instances are independent of each other, and they may or may not interact with other instances. As in object-oriented programming, the ability of one component to act directly on any other is specified in the corresponding class as a state element. Interaction is naturally carried out via precisely defined interfaces. Usually a single class is defined which represents the entire system, and it has one instance, but this is only a convention. This class may have additional state elements of its own. Whereas a system in VDM-SL has a central state which is modified throughout the lifetime of the system, the state of a VDM++ system is distributed among all of its components. Examples can be found in the VDM++ tutorials distributed with Overture.

VDM-RT VDM-RT is a small extension to VDM++ which adds two primary features:

- The ability to define how the specified system is envisioned to be allocated on a distributed execution platform, together with the communication topology.
- The ability to specify the timing behaviours of individual components, as well as whether certain behaviours are meant to be cyclical.

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

C.3 20-sim

20-sim [Con13, Bro97] is a commercial modelling and simulation software package for mechatronic systems. With 20-sim, models can be created graphically, similar to drawing an engineering scheme. With these models, the behaviour of dynamic systems can be analyzed and control systems can be designed. 20-sim models can be exported as C-code to be run on hardware for rapid prototyping and HiL-simulation. 20-sim includes tools that allow an engineer to create models quickly and intuitively. Models can be created using equations, block diagrams, physical components and bond graphs [KR68]. Various tools give support during the model building and simulation. Other toolboxes help to analyze models, build control systems and improve system performance. Figure 4 shows 20-sim with a model of a controlled

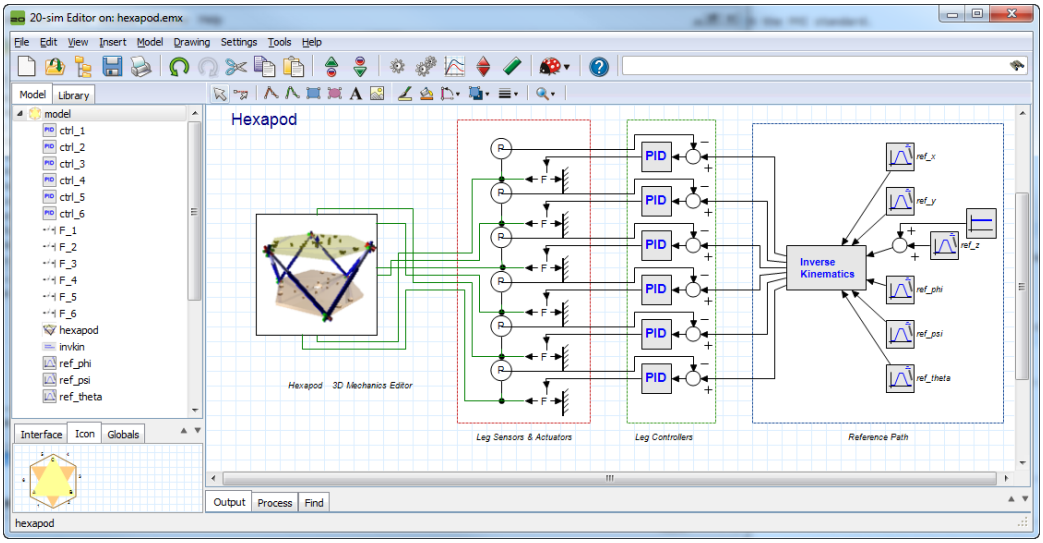

Figure 4: Example of a hexapod model in 20-sim.

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

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

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

C.4 OpenModelica

OpenModelica [Fri04] is an open-source Modelica-based modelling and simulation environment. Modelica [FE98] is an object-oriented, equation based language to conveniently model complex physical systems containing, e.g., mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents. The Modelica language (and OpenModelica) supports continuous, discrete and hybrid time simulations. OpenModelica already compiles Modelica models into FMU, C or C++ code for simulation. Several integration solvers, both fixed and variable step size, are available in OpenModelica: euler, rungekutta, dassl (default), radau5, radau3, radau1.

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

- via command line invocation of the omc compiler
- via C API calls to the omc compiler dynamic library
- via the CORBA interface
- via OMPython interface [GFR+12]

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

The integration of OpenModelica into the INTO-CPS tool chain is realized via compliance with the FMI standard, and is described in Deliverable D4.3b [PBL+17].

C.5 RT-Tester

The RT-Tester [Ver15a] 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. In the INTO-CPS tool chain, RT-Tester is responsible for model-based testing, as well as for model checking. This section gives some background information on the tool from these two perspectives.

C.5.1 Model-based Testing

The RT-Tester Model Based Test Case and Test Data Generator (RTT-MBT) [Ver15b] 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-ends for this. The most important technical challenge in model-based test automation is the extraction of test cases from test models. RTT-MBT combines an SMT solver with a technique akin to bounded model checking so as to extract finite paths through the test model according to some predefined criterion. This criterion can, for instance, be MC/DC coverage, or it can be requirements coverage (if the requirements are specified as temporal logic formulae within the model). A further aspect is that the environment can be modelled within the test model. For example, the test model may contain a constraint such that a certain input to the system-under-test remains in a predefined range. This aspect becomes important once test automation is lifted from single test models to multi-model cyber-physical systems. 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.

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

C.5.2 Model Checking of Timed State Charts

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

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

- A component called *TestModel* that is annotated with stereotype *TE*.
- A component called *SystemUnderTest* that is annotated with stereotype *SUT*.

RTT-MBT uses the stereotypes to infer the role of each component. The interaction between these two parts is implemented via input and output interfaces that specify the accessibility of variables using UML stereotypes.

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

A simple example is depicted in Figure 5, which shows a simple composite structure diagram in Modelio for a turn indication system. The purpose of the system is to control the lamps of a turn indication system in a car. Further details are given in [Ver13]. The test model consists of the two aforementioned components and two interfaces:

- **Interface1** is annotated with stereotype *TE2SUT* and contains three variables `voltage`, `TurnIndLvr` and `EmerSwitch`. These variables are controlled by the environment and fed to the system under test as inputs.
- **Interface2** is annotated with stereotype *SUT2TE* and contains two variables `LampsLeft` and `LampsRight`. These variables are controlled 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:

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

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

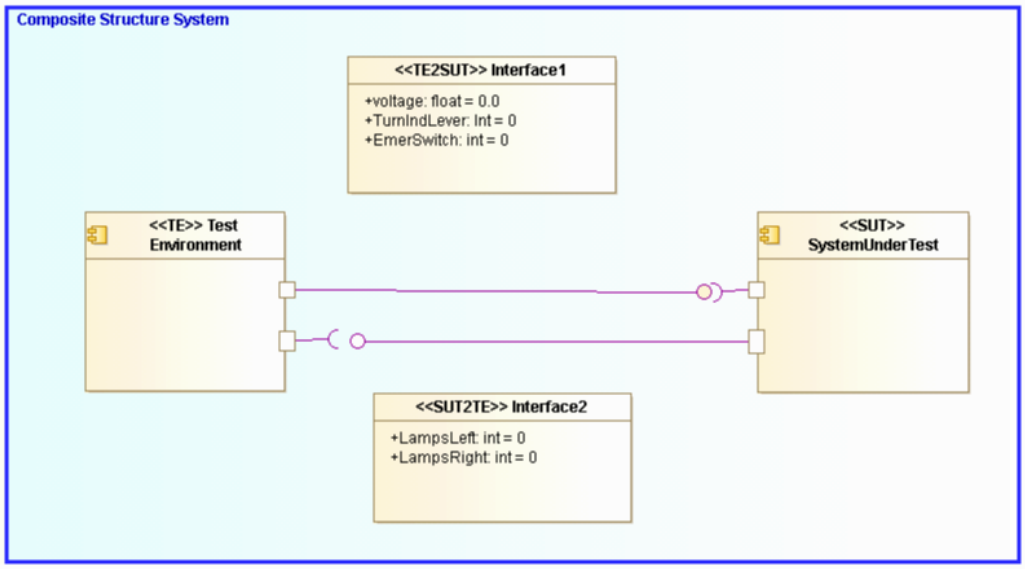

Figure 5: Simple model that highlights interfaces between the environment and the system-under-test.

C.6 4DIAC

Jose Cabral

C.7 AutoFOCUS-3

Jose Cabral