INtegrated TOol chain for model-based design of CPSs

The INtegrated TOolchain for Cyber-Physical Systems (INTO-CPS): a Guide

Version: 0.01

Date: October, 2018

The INTO-CPS Association

http://into-cps.org

Contributors:

Peter Gorm Larsen, Aarhus University
John Fitzgerald, Newcastle University
Jim Woodcock, University of York
Christian König, TWT
Stylianos Basagiannis, UTRC
Etienne Brosse, Softeam
Cláudio Gomes, University of Antwerp
José Cabral, Fortiss
Hugo Daniel Macedo, Aarhus University
Casper Thule, Aarhus University
Andrey Sadovykh, Softeam
Constantin-Bala Zamfirescu,"Lucian Blaga", University of Sibiu
Mihai Neghina, "Lucian Blaga", University of Sibiu
Ken Pierce, Newcastle University
Carl Gamble, Newcastle University
Richard Payne, Newcastle University

Editors:

Peter Gorm Larsen, Aarhus University
John Fitzgerald, Newcastle University

Document History

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

Abstract

The successful design, implementation and maintenance of cyber-physical systems requires collaboration between diverse engineering disciplines and organisations, each of which may use radically different tools and notations. The INtegrated TOolchain for Cyber-Physical Systems (INTO-CPS) is an open framework that permits the coupling of tools for model-based CPS engineering. Its formal foundations and the use of the Functional Mockup Interface standard permit the coherent integration of tools that describe CPS architecture, data, and discrete-event and continuous-time models of system elements. This allows engineers to produce collaborative models (co-models) and undertake co-simulation of behaviour at the whole-CPS level. It permits the machine-assisted trade space analysis, analytic detection and resolution of defects, generation of tests and of code. The value of the approach in reducing time to market and particular in reducing the number of physical prototype iterations required in product development, has been demonstrated in industry.

This report is a guide to INTO-CPS. It includes a review of the challenges facing CPS engineers today and argues for open and integrated toolchains. The INTO-CPS technology is described briefly, and the semantic foundation of the approach are outlined. Methods for exploiting the toolchain within existing systems engineering processes are outlined, and the current toolchain itself is described alongside several real industry studies. We argue that the future of CPS engineering relies on the integration of existing tools and processes, and we offer a potential roadmap for future research in the field, notably in realising the potential of co-models as learning digital twins.

Contents

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 1.

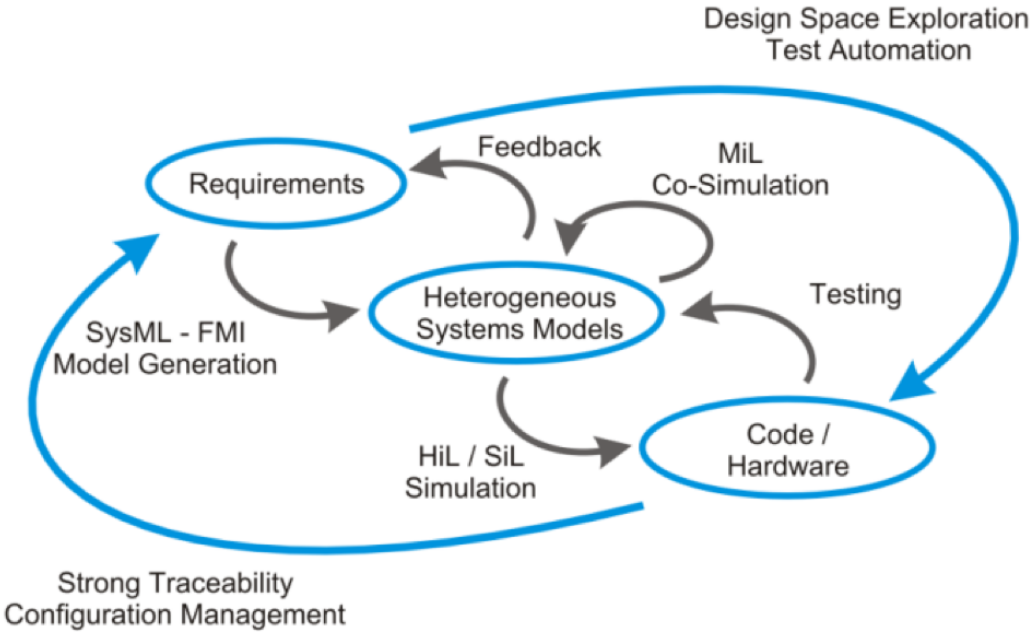


Figure 1: Connections in the INTO-CPS tool chain.

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 "optima" 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) called Maestro has been built on the baseline technologies and in accordance with requirements driven by the industry case studies outlined below [TLLM18]. 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) [BLV+10]. The goals for the COE include, among others, optimised scalability and performance, and data exchange between the different models facilitated by the Functional Mockup Interface (FMI) [Blo14]. 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 (on top of Electron). 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.

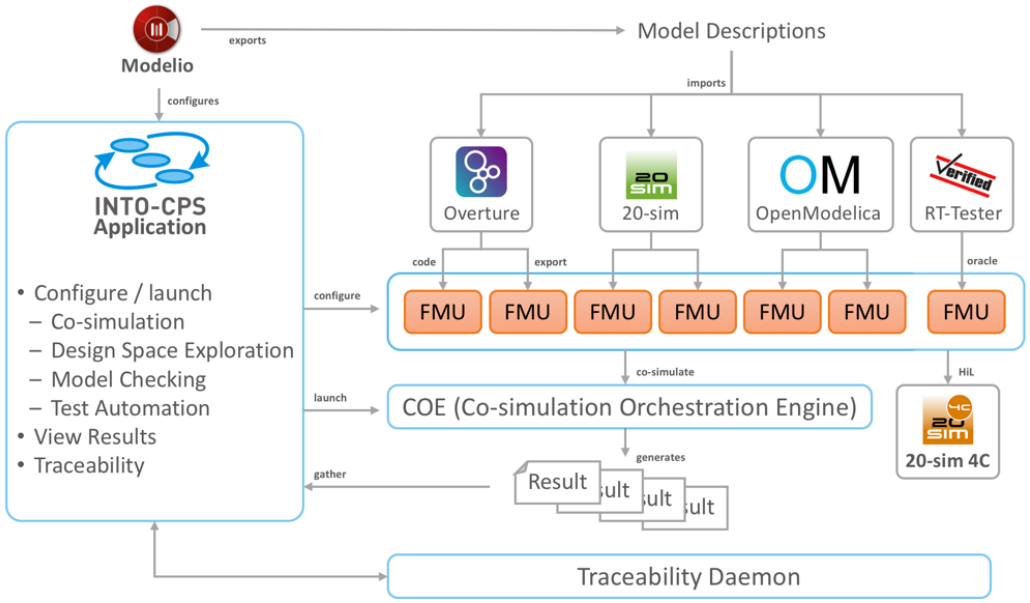


Figure 2: Overview of the INTO-CPS tool chain.

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 even between artefacts produced by different tools. The INTO-CPS tool chain is described in more detail in Section 6.

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. These cases (and a couple of industrial cases conducted by external companies) are described further in Section 7. A brief overview of the industrial cases inside the INTO-CPS project and their main challenges are:

In the Railways case study led by the French company ClearSy, 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 independence 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 the Danish company Agrointelli 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. In addition, the development of the autonomous control for the lawn mower was carried out. 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.

In the Building Automation case study led by the Irish part of United Technology Research Centre (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. 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 the German company 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 assessment 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 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.

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. Both languages have been formalised and mechanised 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. There has also been a close integration with the industrial case studies (in particular, the railways and building applications) and supported the development of the INTO-CPS tool chain. It has been demonstrated 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

JCPW: Add a short introduction

4.1 Foundations of the SysML profile for CPS modelling

The INTO-CPS project proposes a novel technique for proof-based analysis of co-simulations that considers both architectural and behavioural properties of co-simulations. In D2.3a [ZCWO17], the technique is illustrated by way of two case studies, one from from railways and another one from the area of smart buildings control. D2.2a [ACM+16] instantiates the approach to robotic control.

4.1.1 SysML

The Systems Modelling Language (SysML) [OMG12] builds on the Unified Modelling Language (UML) to provide a general-purpose notation for systems engineering. SysML supports the modelling of cyber-physical systems (CPSs), which are designed to actively engage with the physical world in which they reside. They tend to be heterogeneous: their subsystems tackle a wide variety of domains (such as, mechanical, hydraulic, analogue, and a plethora of software domains) that mix phenomena of both continuous and discrete nature, typical of physical and software systems, respectively. Such systems are typically engineered using a variety of languages and tools that adopt complementary paradigms; examples are physics-related models, control laws, and sequential, concurrent, and real-time programs. This diversity makes CPS generally difficult to analyse and study.

4.1.2 Co-simulation

CPSs are often handled modularly to tackle this heterogeneity and complexity. To separate concerns effectively, the global model of the system is decomposed into subsystems, each typically focused on a particular phenomenon or domain and tackled by the most appropriate modelling technique. Simulation, the standard validation technique for CPS, is often carried

out modularly also, using co-simulation [GTB+17, GTB+18a], the coupling of subsystem simulations. This constitutes the backdrop of the industrial Functional Mockup Interface (FMI) standard [Int14, BBG+13, CWA16] for co-simulation of components built using distinct modelling tools. The Functional Mock-up Interface (FMI) Standard has been proposed to address the challenge of interoperability, coupling different simulators and their high-level control components via a bespoke FMI API.

While co-simulation is currently the predominant approach to analyse CPS, INTO-CPS proposes a proof-based complementary technique that uses mathematical reasoning and logic. Simulation is useful in helping engineers to understand modelling implications and spot design issues, but cannot provide universal guarantees of correctness and safety. It is usually impossible to run an exhaustive number of simulations as a way of testing the system. For these reasons, it is often not clear how the evidence provided by simulations is to be qualified, since simulations depend on parameters and algorithms, and are software systems (with possible faults) in their own right.

Proof-based techniques, on the other hand, hold the promise of making universal claims about systems. They can potentially abstract from particular simulation scenarios, parametrisations of models, and interaction patterns used for testing. In traditional software engineering, they have been successfully used to validate the correctness of implementations against abstract requirements models [WLBF09]. Yet, their application to CPS is fraught with difficulties: the heterogeneous combination of languages used in typical descriptions of CPS raises issues of semantic integration and complexity in reasoning about those models. The aspiring ideal of any verification technique is a compositional approach, and such approaches are still rare for CPS [NLFS18].

4.1.3 The INTO-CPS approach to verification and co-simulation

Our approach is to formally verify the well-formedness and healthiness of SysML CPS architectural designs as a prelude to co-simulation. The designs are described using INTO-SysML [APC+15], a profile for multi-modelling and FMI co-simulation. The well-formedness checks verify that designs comply with all the required constraints of the INTO-SysML meta-model; this includes connector conformity, which checks the adequacy of the connections between SysML blocks (denoting components) with respect to the types of the ports being wired. The healthiness checks concern detection of algebraic loops, a feedback loop resulting in instantaneous cyclic dependencies; this is

relevant because a desirable property of co-simulation, which often reduces to coupling of simulators, is convergence (where numerical analyses approximate the solution), which is dependent on the structure of the subsystems and cannot be guaranteed if this structure contains algebraic loops [KS00, BBG+13]. The work in INTO-CPS demonstrates the capabilities of our verification workbench for modelling languages and engineering theories mechanised in the Isabelle proof assistant [NK14], and the CSP process algebra [Hoa85a] with its accompanying FDR3 refinement-checker [RABR16].

Our technique is based on abstraction: we use a relational view of FMUs (functional mock-up units, the components in an FMI architecture) that abstracts from reactive behaviours as well as the API imposed by FMI. This allows us to focus on the fundamental properties of a co-simulation, while introducing details into the model view refinement that preserves those properties.

4.1.4 Instantiation for robotics applications

We have extended and restricted the INTO-SysML profile to deal with mobile and autonomous robotic systems. For modelling the controllers, we use RoboChart [LMR+17]. For modelling the robotic platform and the environment, we use Simulink [Inc]. We have also given a behavioural semantics for models written in the profile using CSP. The semantics is agnostic to RoboChart and Simulink, and captures a co-simulation view of the multi-models based on the FMI API.

Our semantics can be used in two ways. First, by integration with a semantics of each of the multi-models that defines their specific responses to the simulation steps, we can obtain a semantics of the system as a whole. Such semantics can be used to establish properties of the system, as opposed to properties of the individual models. In this way, we can confirm the results of co-simulations via model checking or theorem proving, for example.

There are CSP-based formal semantics for RoboChart [MCR+16] and Simulink [MZC12, CMW13] underpinned by a precise mathematical semantics. Our next step is their lifting to provide an FMI-based view of the behaviour of models written in these notations. With that, we can use RoboChart and Simulink models as FMUs in a formal model of a co-simulation as suggested here, and use CSP and its semantics to reason about the co-simulation.

It is also relatively direct to wrap existing CSP semantics for UML state machines [DC03, RW05] to allow the use of such models as FMUs in a co-

simulation. In this case, traditional UML modelling can be adopted.

Secondly, we can use our semantics as a specification for a co-simulation. The work in [CWA16] provides a CSP semantics for an FMI co-simulation; it covers not only models of the FMUs, but also a model of a master algorithm of choice. The scenario defined by an INTO-SysML model identifies inputs and outputs, and their connections. The traces of the FMI co-simulation model should be allowed by the CSP semantics of the INTO-SysML model.

There is no support to establish formal connections between a simulation and the state machine and physical models (of the robotic platform and the environment). The SysML profile proposed here supports the development of design models via the provision of domain-specific languages based on familiar diagrammatic notations and facilities for clear connection of models. Complementarily, as explained above, the semantics of the profile supports the verification of FMI-based co-simulations. There are plans for automatic generation of simulations of RoboChart models [CWA16]. The semantics we propose can be used to justify the combination of these simulations with Simulink simulations as suggested above.

4.1.5 Future work

We first suggest the development of a tool that supports the user of our technique in automatically generating the Isabelle/UTP architectural model, as well as a sketch of the behavioural model. The formal developer can use the sketch as a starting point, completing it with a detailed encoding of functional behaviours of FMUs. Secondly, elements of the refinement strategy from abstract into concrete FMU models ought be explored for a larger spectrum of case studies and examples, beyond the ones we presented in this report. Both these works could be tackled by the INTO-CPS Association.

5 The INTO-CPS Method and Guidelines

John Fitzgerald

1. Draft 1 (based on Deliverables only): end June
2. Draft 2 (pared down, appropriate material dealt with by external references): end July
3. Draft 3 (consistent with other sections, cross-references): end August

5.1 Introduction

The INTO-CPS tool chain enables collaborative multidisciplinary model-based design of CPSs. Each discipline has its own culture, abstractions, and approaches to problem solving that inform how they are used. Many of these things are tacit and tend to be discovered only after trying to combine them. This chapter aims to help the reader overcome these challenges, and to understand how best to use these technologies.

This chapter complements the tools User Manual (Deliverable D4.3a [BLL+17]) —which gives detail on how to use the features of the tool chain— by providing information on when and why you might use these features. The guidance in this document has been distilled from experience gained in a series of pilot studies and applications of INTO-CPS technologies to real industrial case studies. These pilot studies now appear as examples that can be opened directly from the INTO-CPS Application, supported by descriptions in the Examples Compendium (Deliverable D3.6 [MGP+17]). Industrial applications can be read about in the Case Studies report (Deliverable D1.3a [OLF+17]).

Since this document is aimed at both new and experienced users of the INTO-CPS technologies, it has been divided into two parts. Sections 5.1–5.3 covers introductory material including the terminology used in INTO-CPS, and the various activities that INTO-CPS enables. Sections 5.4–5.9, covers more advanced topics that require a basic familiarity with the INTO-CPS technologies. Although Sections 5.4–5.9 are ordered based on a start-to-end "work flow" of system development with INTO-CPS, it is not necessary to read them in order. Experienced users may read any section on which they require further guidance, new users are recommended to:

- Read the introductory material in Sections 5.1–5.3.

- Follow the first tutorial to experience using the INTO-CPS Application.
- Import one or two examples from the Examples Compendium (Deliverable D3.6 [MGP+17]) into the INTO-CPS Application and interact with them.
- As you start your own multi-modelling, return to this chapter for guidance on particular topics as required.

5.2 Concepts and Terminology

CPSs bring together domain experts from diverse backgrounds such as software engineering and control engineering. Each discipline has developed its own terminologies, principles and philosophy, often using similar terms for quite different meanings and different terms that have the same meaning. In this section we discuss the meanings that we give to common concepts.

5.2.1 Systems

A ***System*** is "a combination of interacting elements organized to achieve one or more stated purposes" [INC15]. Any given system will have an ***environment***, considered to be everything outside of the system. The behaviour exhibited by the environment is beyond the direct control of the developer [BFG+12]. We also define a ***system boundary*** as being the common frontier between the system and its environment. The definition of the system boundary is application-specific [BFG+12].

Cyber-Physical Systems (CPSs) refer to "ICT systems (sensing, actuating, computing, communication, etc.) embedded in physical objects, interconnected (including through the Internet) and providing citizens and businesses with a wide range of innovative applications and services" [Tho13, DAB+15].

Many CPSs are ***Systems of Systems (SoSs)***. An SoS is a "collection of constituent systems that pool their resources and capabilities together to create a new, more complex system which offers more functionality and performance than simply the sum of the constituent systems" [HIL+14]. CPSs may exhibit the characteristics of SoSs.

5.2.2 Models

A ***model*** is a potentially partial and abstract description of a system, limited to those components and properties of the system that pertain to the current goal [HIL+14]. A model should be "just complex enough to describe or study the phenomena that are relevant for our problem context" [vA10]. Models should be abstract "in the sense that aspects of the product not relevant to the analysis in hand are not included" [FL98]. A model "may contain representations of the system, environment and stimuli" [FLV14] Further discussion may be required on the

In a CPS model, we describe systems with cyber, physical and network elements. These components are often drawn from different domains, and are modelled in a variety of languages, with different notations, concepts, levels of abstraction, and semantics, which are not necessarily easily mapped one to another. This heterogeneity presents a significant challenge to simulation in CPSs [HIL+14]. We use ***continuous time (CT)*** and ***discrete event (DE)*** models to represent physical and cyber elements as appropriate. A CT model has state that can be changed and observed *continuously* [vA10] and is described using either explicit continuous functions of time either implicitly as a solution of differential equations. A DE model has state that can be changed and observed only at fixed, *discrete*, time intervals [vA10]. The approach used in the DESTECS project was to use *co-models* – "a model comprising a DE model, a CT model and a contract" [BFG+12]. In INTO-CPS we use the term ***multi-models*** – "comprising multiple ***constituent*** DE and CT models". Related to this is a ***Hybrid Model***, which contains both DE and CT elements.

A ***requirement*** may impose restrictions, define system capabilities or identify qualities of a system and should indicate some value or use for the different stockholders of a CPS. ***Requirements Engineering (RE)*** is the process of the specification and documentation of requirements placed upon a CPS. Requirements may be considered in relation to different ***contexts*** – that is the point of view of some system component or domain, or interested stakeholder.

A ***design parameter*** is a property of a model that can be used to affect the model's behaviour, but remains constant during a given simulation [BFG+12]. A ***variable*** is feature of a model that may change during a given simulation [BFG+12]. ***Non-functional properties (NFPs)*** pertain to characteristics other than functional correctness. For example, reliability, availability, safety and performance of specific functions or services are NFPs that are quantifiable. Other NFPs may be more difficult to mea-

sure [PF10].

The activity of creating models may be referred to as ***modelling*** [FLV14] and related terms include ***co-modelling*** and ***multi-modelling***. A ***workflow*** is a sequence of ***activities*** performed to aid in modelling. A workflow has a defined purpose, and may cover a subset of the CPS engineering development lifecycle.

The term ***architecture*** has many different definitions, and range in scope depending upon the scale of the product being 'architected'. We use the simple definition from [PHP+14]: "an architecture defines the major elements of a system, identifies the relationships and interactions between the elements and takes into account process. Those elements are referred to as ***components***. An architecture involves both a definition of structure and behaviour. Importantly, architectures are not static but must evolve over time to reflect the change in a system as it evolves to meet changes to its requirements". In a CPS architecture, components may be either ***cyber components*** or ***physical components*** corresponding to some functional logic or an entity of the physical world respectively.

We consider both a ***holistic architecture*** and a ***design architecture***. An example of their use is given in Section 5.6. The aim of a holistic architecture is to identify the main units of functionality of the system reflecting the *terminology and structure of the domain of application*. It describes a conceptual model that highlights the main units of the system architecture and the way these units are connected with each other, taking a holistic view of the overall system. The design architectural model of the system is effectively a multi-model. The INTO-CPS SysML profile [APCB15] is designed to enable the specification of CPS design architectures, which emphasises a decomposition of a system into ***subsystems***, where each subsystem is an assembly of cyber and physical components and possibly other subsystems, and modelled separately in isolation using a special notation and tool designed for the domain of the subsystem. ***Evolution*** refers to the ability of a system to benefit from a varying number of alternative system components and relations, as well as its ability to gain from the adjustments of the individual components' capabilities over time (Adjusted from SoS [NLF+13]).

Considering the interactions between components in a system architecture, an ***interface*** "defines the boundary across which two entities meet and communicate with each other" [HIL+14]. Interfaces may describe both digital and physical interactions: digital interfaces contain descriptions of operations and attributes that are *provided* and *required* by components. Physical interfaces describe the flow of physical matter (for example fluid and electrical power)

between components.

There are many methods of describing an architecture. In this report, an ***architecture diagram*** refers to the symbolic representation of architectural information contained in a model. An ***architectural framework*** is a "defined set of viewpoints and an ontology" and "is used to structure an architecture from the point of view of a specific industry, stakeholder role set, or organisation. [HIL+14]. In the application of an architecture framework, an ***architectural view*** is a "work product (for example an architecture diagram) expressing the architecture of a system from the perspective of specific system concerns" [PHP+14].

The INTO-CPS SysML profile comprises diagrams for architectural modelling and ***design space exploration*** specification. There are two architectural diagrams. The ***Architecture Structure Diagram (ASD)*** specialises SysML block definition diagrams to support the specification of a system architecture described in terms of a system's components. ***Connections Diagrams (CDs)*** specialise SysML internal block diagrams to convey the internal configuration of the system's components and the way they are connected. The system architecture defined in the profile should inform a co-simulation multi-model and therefore all components interact through connections between flow ports. The profile permits the specification of ***cyber*** and ***physical*** components and also components representing the ***environment*** and ***visualisation*** elements. The INTO-CPS SysML profile includes three design space exploration diagrams: a ***parameters diagram***; an ***objective diagram***; and a ***ranking diagram***. See Section 5.2.4 for concepts relating to design space exploration.

5.2.3 Tools

The ***INTO-CPS tool chain*** is a collection of software tools, based centrally around FMI-compatible co-simulation, that supports the collaborative development of CPSs. The ***INTO-CPS Application*** is a front-end to the INTO-CPS tool chain. The Application allows the specification of the co-simulation configuration, and the co-simulation execution itself. The application also provides access to features of the tool chain without an existing user interface (such as design space exploration and model checking).

Central to the INTO-CPS tool chain is the use of the ***Functional Mockup Interface (FMI)*** standard. This is a tool-independent standard to support both model exchange and co-simulation of dynamic models using a combi-

nation of XML-files and compiled C-code [Blo14]. Part of the FMI standard for model exchange is specification of a ***model description*** file. This is an XML file that supplies a description of all properties of a model (for example input/output variables). A ***Functional Mockup Unit (FMU)*** is a tool component that implements FMI. Data exchange between FMUs and the synchronisation of all simulation solvers [Blo14] is controlled by a ***Master Algorithm***.

Co-simulation is the simultaneous, collaborative, execution of models and allowing information to be shared between them. The models may be CT-only, DE-only or a combination of both. The ***Co-simulation Orchestration Engine (COE)*** combines existing co-simulation solutions (FMUs) and scales them to the CPS level, allowing CPS multi-models to be evaluated through co-simulation. This means that the COE implements a ***Master Algorithm***. The COE will also allow real software and physical elements to participate in co-simulation alongside models, enabling both Hardware-in-the-Loop (HiL) and Software-in-the-Loop (SiL) simulation.

In the INTO-CPS Application, a ***project*** comprises: a number of FMUs, optional source models (from which FMUs are exported); a collection of ***multi-models***; and an optional SysML architectural model. A multi-model includes a list of FMUs, defined instances of those FMUs, specified connections between the inputs/outputs of the FMU instances, and defined values for design parameters of the FMU instances. For each multi-model a ***co-simulation configuration*** defines the step size configuration, start and end time for the co-simulation of that multi-model. Several configurations can be defined for each multi-model.

Code generation is the transformation of a model into generated code suitable for compilation into one or more target languages (e.g. C or Java).

We consider two tool-supported methods for recording the rationale of design decisions in CPSs. ***Traceability*** is the association of one model element (e.g., requirements, design artefacts, activities, software code or hardware) to another. ***Requirements traceability*** "refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction" [GF94]. ***Provenance*** "is information about entities, activities, and people involved in producing a piece of data or thing, which can be used to form assessments about its quality, reliability or trustworthiness" [MG13]. In INTO-CPS traceability between model elements defined in the various modelling tools is achieved through the use of ***OSLC messages***, handled by a traceability ***daemon tool***. This supports the ***impact analysis*** and general ***traceability queries***.

Two broad groups of users are considered in the INTO-CPS project. A ***Tool Chain User*** is an individual who uses the INTO-CPS Tool Chain and its various analysis features. A ***Foundations Developer*** is someone who uses the developed foundations and associated tool support (see Section 5.2.6) to reason about the development of tools.

5.2.4 Analysis

Design-Space Exploration (DSE) is "an activity undertaken by one or more engineers in which they build and evaluate [multi]-models in order to reach a design from a set of requirements" [BFG+12]. "The ***design space*** is the set of possible solutions for a given design problem" [BFG+12]. Where two or more models represent different possible solutions to the same problem, these are considered to be ***design alternatives***. In INTO-CPS design alternatives are defined using either a range of parameter values or different multi-models. Each choice involves making a selection from alternatives on the basis of an ***objective*** – criteria or constraints that are important to the developer, such as cost or performance. The alternative selected at each point constrains the range of design alternatives that may be viable next steps forward from the current position. Given a collection of alternatives with corresponding objective results, a ***ranking*** may be applied to determine the 'best' design alternative.

Test Automation (TA) is defined as the machine assisted automation of system tests. In INTO-CPS, we concentrate on various forms of ***model-based testing*** – centering on testing system models, against the requirements on the system. The ***System Under Test (SUT)*** is "the system currently being tested for correct behaviour. An alias for system of interest, from the point of view of the tester" [HIL+14]. The SUT is tested against a collection of ***test cases*** – a finite structure of input and expected output [UPL06], alongside a ***test model***, which specifies the expected behaviour of a system under test [CMC+13]. TA uses a ***test suite*** – a collection of ***test procedures***. These test procedures are detailed instructions for the set-up and execution of a given set of test cases, and instructions for the evaluation of results of executing the test cases [WG-92].

INTO-CPS considers three main types of test automation: ***Hardware-in-the-Loop (HiL)***, ***Software-in-the-Loop (SiL)*** and ***Model-in-the-Loop (MiL)***. In ***HiL*** there is (target) hardware involved, thus the FMU is mainly a wrapper that interacts (timed) with this hardware; it is perceivable that realisation heavily depends on hardware interfaces and timing properties. In

Software-in-the-Loop (SiL) testing the object of the test execution is an FMU that contains a software implementation of (parts of) the system. It can be compiled and run on the same machine that the COE runs on and has no (defined) interaction other than the FMU-interface. Finally, in ***Model-in-the-Loop (MiL)*** the test object of the test execution is a (design) model, represented by one or more FMUs. This is similar to the SiL (if e.g., the SUT is generated from the design model), but MiL can also imply that running the SUT-FMU has a representation on model level; e.g., a playback functionality in the modelling tool could some day be used to visualise a test run.

Model Checking (MC) exhaustively checks whether the model of the system meets its specification [CGP99], which is typically expressed in some temporal logic such as ***Linear Time Logic (LTL)*** [Pnu77] or ***Computation Tree Logic (CTL)*** [CE81]. As opposed to testing, model checking examines the entire state space of the system and is thus able to provide a correctness proof for the model with respect to its specification. In INTO-CPS, we can concentrate on ***Bounded Model Checking (BMC)*** [CBRZ01, CKOS04, CKOS05], which is based on encodings of the system in propositional logic, for a timed variant of LTL. The key idea of this approach is to represent the semantics of the model as a Boolean formula and then apply a ***Satisfiability Modulo Theory (SMT)*** [KS08] solver in order to check whether the model satisfies its specification. A powerful feature of model checking is that, if the specification is violated, it provides a counterexample trace that shows exactly how an undesired state of the system can be reached [CV03].

5.2.5 Existing Tools and Languages

The INTO-CPS tool chain uses several existing modelling tools. ***Overture***[1] supports modelling and analysis in the design of discrete, typically, computer-based systems using the ***VDM-RT*** notation. VDM-RT is based upon the ***object-oriented*** paradigm where a model is comprised of one or more ***objects***. An object is an instance of a ***class*** where a class gives a definition of zero or more ***instance variables*** and ***operations*** an object will contain. Instance variables define the identifiers and types of the data stored within an object, while operations define the behaviours of the object.

The ***20-sim***[2] tool can represent continuous time models in a number of ways. The core concept is that of connected ***blocks***. ***Bond graphs*** may

[1] `http://overturetool.org/`
[2] `http://www.20sim.com/`

implement blocks. Bond graphs offer a domain-independent description of a physical system's dynamics, realised as a directed graph. The vertices of these graphs are idealised descriptions of physical phenomena, with their edges (***bonds***) describing energy exchange between vertices. Blocks may have input and output ***ports*** that allow data to be passed between them. The energy exchanged in 20-sim is the product of ***effort*** and ***flow***, which map to different concepts in different domains, for example voltage and current in the electrical domain.

OpenModelica[3] is an open-source ***Modelica***-based modelling and simulation environment. Modelica is an "object-oriented language for modelling of large, complex, and heterogeneous physical systems" [FE98]. Modelica models are described by ***schematics***, also called ***object diagrams***, which consist of connected components. Components are connected by ports and are defined by sub components or a textual description in the Modelica language.

Modelio[4] is an open-source modelling environment supporting industry standards like UML and SysML. INTO-CPS will make use of Modelio for high-level system architecture modelling using the ***SysML*** language and proposed extensions for CPS modelling. The systems modelling language (SysML) [Sys12] extends a subset of the UML to support modelling of heterogeneous systems.

5.2.6 Formalisms

The ***semantics*** of a language describes the meaning of a (grammatically correct) program [NN92] (or model). There are different methods of defining a language semantics: ***structural operational semantics***; ***denotational semantics***; and ***axiomatic semantics***.

A structural operational semantics (SOS) describes how the individual steps of a program are executed on an abstract machine [Plo81]. An SOS definition is akin to an interpreter in that it provides the meaning of the language in terms of relations between beginning and end states. The relations are defined on a per-construct basis. Accompanying the relations are a collection of semantic rules which describe how the end states are achieved. Where an operational semantics defines how a program is executed, a denotational approach defines a language in terms of denotations, in the form of abstract

[3] `https://www.openmodelica.org/`
[4] `http://www.modelio.org/`

mathematical objects, which represent the semantic function that maps over the inputs and outputs of a program [SS71].

The Unifying Theories of Programming (UTP) [HJ98] provides a unified framework for describing language semantics. A theory of a language is composed of an ***alphabet***, a ***signature*** and a collection of ***healthiness conditions***.

The Communicating Sequential Processes ***CSP*** notation [Hoa85b] is a formal process algebra for describing communication and interaction. ***INTO-CSP*** is a version of CSP, which will be used to provide a model for the SysML-FMI profile, FMI, VDM-RT and Modelica semantics. It is a front end for a UTP theory of reactive concurrent continuous systems customised for the needs of INTO-CPS. ***Hybrid-CSP*** is a continuous version of CSP defined originally by He Jifeng [Jif94]. It will be used as a basis to inform the design of INTO-CSP.

Several forms of verification are enabled through the use of formally defined languages. ***Refinement*** is a verification and formal development technique pioneered by [BW98] and [Mor90]. It is based on a behaviour preserving relation that allows the transformation of an abstract specification into more and more concrete models, potentially leading to an implementation. ***Proof*** is the process of showing how the validity of one statement is derived from others by applying justified rules of inference [BFL$^+$94].

For the purposes of verification in INTO-CPS, and in particular the work of WP2, we make use of the Isabelle/HOL theorem prover and the FDR3 refinement checker. These are not considered part of the INTO-CPS tool chain, and are used in the INTO-CPS project primarily to support the development of foundation work.

5.3 Introduction

5.4 Traceability

5.5 Requirements Engineering

5.6 Design Space Exploration

5.7 Initial Multi-Modelling using a Discrete-Event Notation: VDM

5.8 Modelling Networks with VDM in Multi-models

5.9 Design Space Exploration

How best to integrate the tutorials?

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

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]

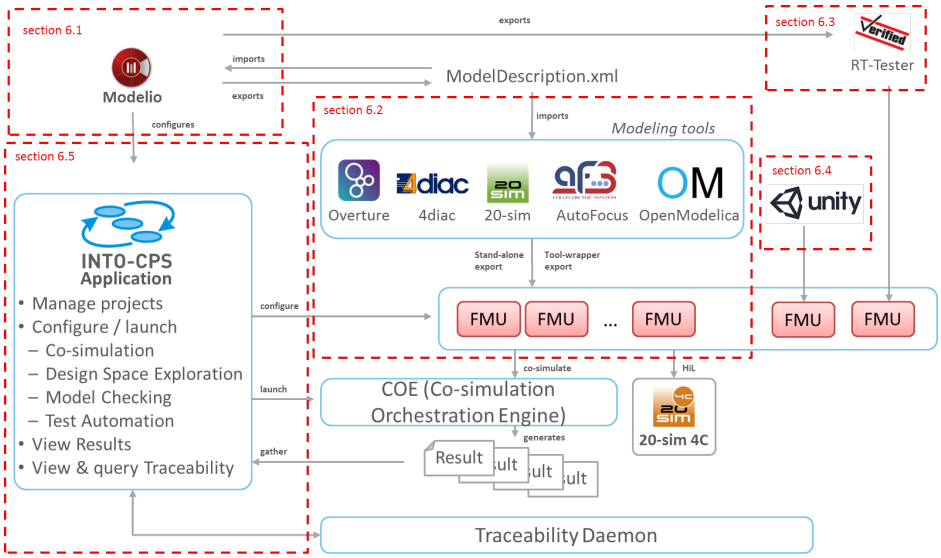


Figure 3: Overview of the different tools and their arrangement in a tool-chain.

6.1 Modelio

Modelio is an open-source modelling 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. [FGP17]), Modelio is a tool for the *architectural modelling* and for *requirements management*.

During the INTO-CPS project, a SysML profile was created, which is currently available as a module for Modelio 3.4 and 3.6 [5]. This INTO-CPS SysML profile extends Modelio with several functionalities that described in detail elsewhere [BQ15, BQ16, Bro17]. 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 Modelling diagram. Importing ModelDescription.xml files creates a SysML block with the corresponding flow ports and attributes, exporting them allows import in other modelling tools, such as those described below in Section 6.2.

The Connections Diagram describes the signal flow between the different SysML blocks, which can each correspond to one FMU. Using the 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.

Diagrams for handling of Design Space Exploration (DSE) were created for Modelio, also included in the INTO-CPS SysML profile. These diagrams allow connection of parameters with signals, definition of objectives for a DSE, connection of signals with objectives, and ranking of results. Using these diagrams, a complete DSE configuration can be exported from Modelio.

Behavioural models that are designed in Modelio as state machines can be exported as `.xmi` files, so that they can be imported to the RT Tester tool.

Furthermore, Modelio allows Requirements management, and supports traceability in the context of INTO-CPS.

[5] see `http://forge.modelio.org/projects/intocps`

6.2 Modelling tools

At the core of the tool-chain are several modelling tools that describe a system or a sub-system in a specific formalism, and perform calculations to understand the dynamic behaviour of the (sub-)system. While the formalisms or application areas can be vastly different, the modelling tools share some common features, which are summarized in this section.

20-sim is a commercial tool for modelling and simluation of mechatronic systems. Together with the related software, 20-sim 4C, Hardware-in-the-Loop simulations can be performed. `http://www.20sim.com/`

OpenModelica is an open-source environment which is based on the Modelica language. It features numerous free libraries to easily model systems from different domains. `https://openmodelica.org/`

Overture is an open-source tool that supports the modelling method *The Vienna Development Method (VDM)*, which is a formal method to describe computing systems. `http://overturetool.org/`

4Diac is an open-source tool for distributed process measurement and control systems based on the IEC 61499 standard. `https://www.eclipse.org/4diac/`

AutoFocus3 is an open-source model based tool to develop embedded software systems. `https://af3.fortiss.org/`

ABS is a language for Abstract Behavioral Specification, which combines implementation-level specifications with verifiability, high-level design with executablity, and formal semantics with practical usability. ABS is a concurrent, object-oriented, modeling language that features functional data-types. `http://abs-models.org/`

Most modelling tools support the same functions in the context of INTO-CPS. 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. After the

actual modelling work is done, the model can be exported as Functional Mock-up Unit (FMU), in accordance with the FMI 2.0 for Co-Simulation standard. This FMU can either contain all the necessary models and solvers, so that is a self-contained model (also called stand alone), or it contains libraries which call a simulation tool to execute the simulation. The latter case is called a tool wrapper FMU. Furthermore, the different steps of importing, saving and exporting generate traces which are sent to the traceability engine of the INTO-CPS Application. The following table summarizes the status of the different tools at the time of writing of this document.

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

Table 1: Functionalities of the modelling tools

6.3 RT Tester

In the framework of INTO-CPS, the RT Tester tool suite (see `https://www.verified.de/products/rt-tester/`) is extended with mainly two objectives: Integration of Test-Automation and of Model Checking in the INTO-CPS tool-chain. Both functions are integrated into the INTO-CPS application, and both support traceability.

6.3.1 Test Automation

Test Automation within INTO-CPS uses the RT Tester tools to generate, perform and analyze tests, based on Co-simulation of a system. The Test Automation functionalities are integrated into the INTO-CPS Application. The behavioural model can be generated in Modelio, and exported as `.xmi` file, which in turn can be read by RT Tester. After the test is created in RT Tester, the test procedure can be cast into an FMU file. Together with a Co-Simulation scenario, and using the COE, the test procedure is used to run a test project. More information on Test Automation in INTO-CPS can be found in [BC+17].

6.3.2 Model Checking

Model checking in INTO-CPS is used to verify system properties of multi-models, consisting of continuous-time (CT) and discrete-event (DE) models. Similar to the Test Automation features, Model Checking is based on the RT Tester tool suite. From a tool-chain perspective, Model Checking is integrated in the INTO-CPS Application, which allows the complete configuration, execution and analysis of a Model Checking experiment. More information on Model Checking in INTO-CPS can be found in [BH17].

6.4 3D animation

The 3D animation FMU allows visualisation of the simulation. It is based on the Unity engine (see `https://unity3d.com`), and extends it by exporting the scenario and the 3D rendering as a FMU [FLG17]. The Modelio SysML profile (see Section 6.1) takes the visualisation FMU into account. The 3D animation FMU also supports Virtual Reality (VR) headsets.

6.5 The INTO-CPS Application

The INTO-CPS Application is the central tool to integrate the different tools and artefacts, to configure and run simulations, manage results, and more. It allows configuration and execution of DSE scenarios (which can be imported from Modelio), and is a front-end for Model-checking and Test automation, by using the RT Tester tool (see Section 6.3). Furthermore, traceability data

can be viewed in the Application, either in an expert view, using the Neo4J visualisation[6], or by pre-configured queries.

[6] https://neo4j.com/

7 The INTO-CPS Industrial Case Studies

The industrial domains benefited from the INTO-CPS technologies in different ways, as expected due to their different multidisciplinary engineering activities. Nonetheless, certain aspects of the tool chain were of benefit to all industrial partners. We thus argue that these are the most broadly applicable benefits of INTO-CPS and the ones that can most successfully be transferred to other domains (fig. 4). Modelling and simulation is heavily used in industry today, to evaluate performance and robustness of the system with regards to requirements. Aspects such as simulation speed, and model fidelity are of high importance. Going beyond model-based design and single model simulation, co-simulation enables the analysis of physical interactions of systems that were previously not captured, due to the different domains at which the physics were modelled. Multi-physics analysis enables early analysis and detection of issues that were only uncovered at the physical prototyping stage, thus saving time and money.

The use of SysML modelling with the INTO-CPS profile was valuable to the majority of the industrial case studies, as a way to provide a common documentation of the system structure (particularly valuable for larger teams). It also enhanced communication within each case study project and across the disciplines and tools of the project. The connection with the simulation tools and the COE via the export of Model Description and Co-Simulation Configuration adds even more value to the SysML model. The DSE component was valuable as a way to sweep parameters and automate co-simulations. This led to faster prototyping (and lessened the need for physical prototypes) and a better understanding of the complex interactions between the system parameters. The DSE component was well integrated into the toolchain and used in the industrial studies through a user friendly interface, providing also its capability to reuse existing parts of the tools and models (through respectively the INTO-CPS application and FMUs).

The 3D visualisation component was valuable for all industrial partners. As commercial entities, the 3D visualisation has great value as a marketing and sales tool. It also greatly enhances the user experience when analysing the models. These benefits are in addition to engineering benefits which can be gained for certain case studies, where the visual aspects of the problem are of interest and can be studied using the 3D visualisation. This is not the case for all CPS problems, but any CPS company can benefit from the 3D capabilities for marketing purposes, which are of high importance to any business. More generally, one of the greater benefits of the INTO-CPS tool

Figure 4: INTO-CPS Industrial Case studies

chain was the ability to reuse models, including existing legacy models for new purposes. The broad tool compatibility (including tools outside the tool chain, this attesting its openness) increased the reuse even more. The tool chain is also fully compliant with the FMI standard which increases the value of models developed with the INTO-CPS tool chain, as they can be reused further in the future. Finally, the baseline tools of INTO-CPS were all made compatible and tested with industry-grade and open source tools through the FMI standard, thus opening new possibilities for the tools and their users.

7.1 The Automotive Case Study

This sections presents mainly a case study that develops functions for vehicles, in particular electric vehicles using the INTO-CPS technology. Its goal is to create an assistant system for estimating the range of an electric vehicle, based on a vehicle model and real data from the environment, such as route topology or weather. Furthermore, the range estimation is dynamic, as it takes changes in the initial assumptions into account, and influences the vehicle behaviour accordingly.

The case study can be considered a CPS because it contains local intelligence and autonomy in the vehicle. This is assisted by information about its environment typically derived from a cloud context (here, information on weather and traffic / route) and the logic depends upon the physical dynamics of the electric vehicle (Fig. 5).

A part of the system is transferred seamlessly from a simulation model to

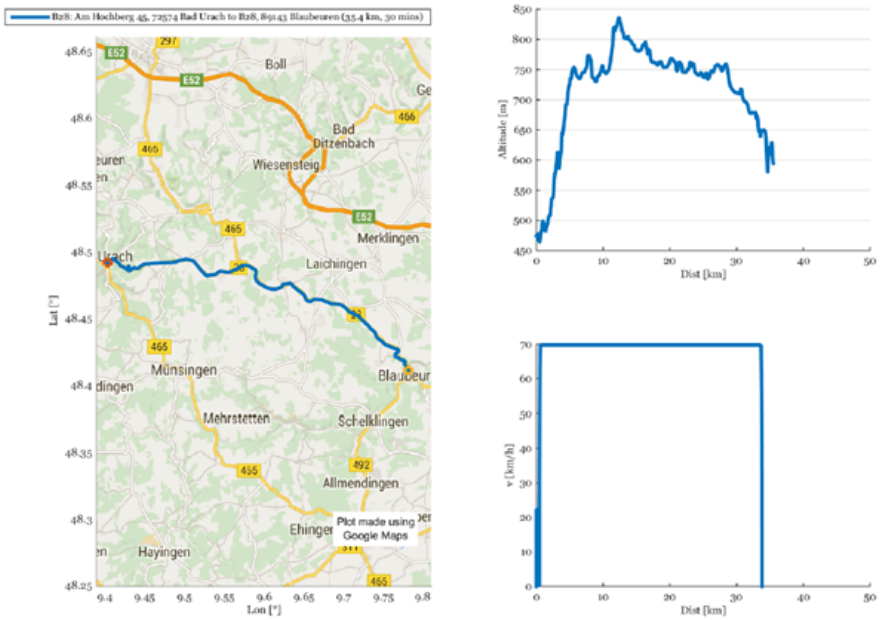


Figure 5: Automotive case study using INTO-CPS: Velocity and altitude profile for a route of 35 km in the vicinity of Stuttgart. The route consists here of a country road only, and thus the velocity is stable at 70 km/h, while the altitude varies between 450m and 850m above sea level.

real hardware (here, as Raspberry Pi) and simulated with the remainder of the system. Since the case study was developed as part of the INTO-CPS project [4], one aim was to evaluate the INTO-CPS tools and methods. Here this is in particular the Co-simulation orchestration engine (COE), which is a FMI 2.0 compliant master algorithm that allows coupling of continuous-time (CT) and discrete event (DE) models in a Co-Simulation setup [4,5]. furthermore, the system was modelled in SysML, using the CPS-extension of the Modelio tools [7]. The models themselves were created using Matlab1, 20-sim2, C++ and Overture3 [2].

7.2 The Agricultural Case Study

The focus of the case study is the development of the agricultural field robot, Robotti using the INTO-CPS technology. Models of the machine dynamics and controllers have been developed using the baseline tools and Co-simulated using the COE. The DSE feature is applied in the development

and assessment of the steering controller which is crucial to the performance of the robot. The agricultural case study has been extended with an additional industrial application, namely the AI-Mower. The dynamics of the mower is modelled and co-simulated to assess and optimise a new approach for steering the mower.

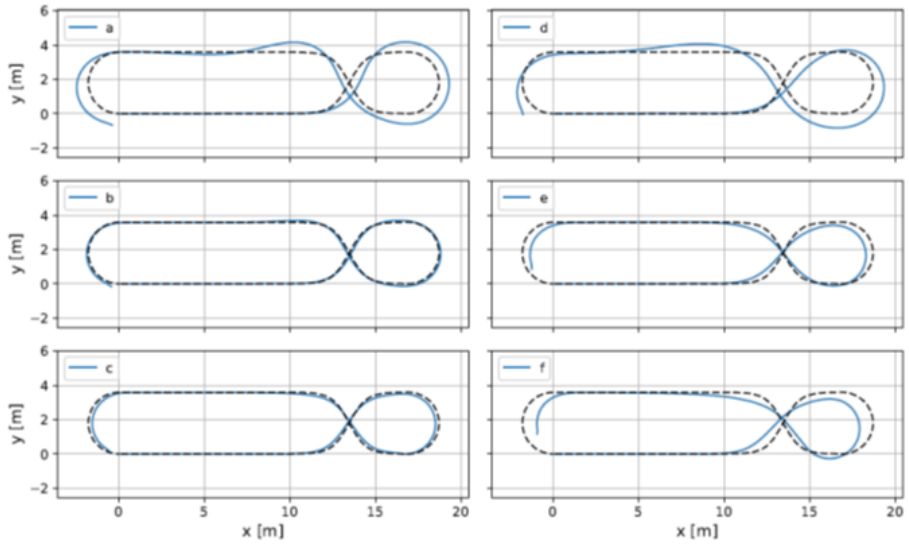


Figure 6: Simulated trajectories of the six controller configurations (a,b,...,f)

Additionally, the 3D FMU feature has been applied to visualise the machine based on the models of the kinematics and dynamics. The details on the mower are kept confidential contrary to the Robotti project which is public. Through the model development using INTO-CPS, the work related to Robotti have been focused towards DSE-based optimisation of the steering controller which is crucial to the performance of the machine. Several scenarios of different steering controller configurations are simulated using the COE and the DSE feature is applied to estimate the optimal controller configuration of the Robotti. The influence of the controller parameters are shown in Fig. 6 where the six simulated trajectories are shown corresponding to six combinations of two controller parameters. The dashed line represents the desired route and the blue line represents the simulated trajectory of the Robot.

Related Publications can be found at:

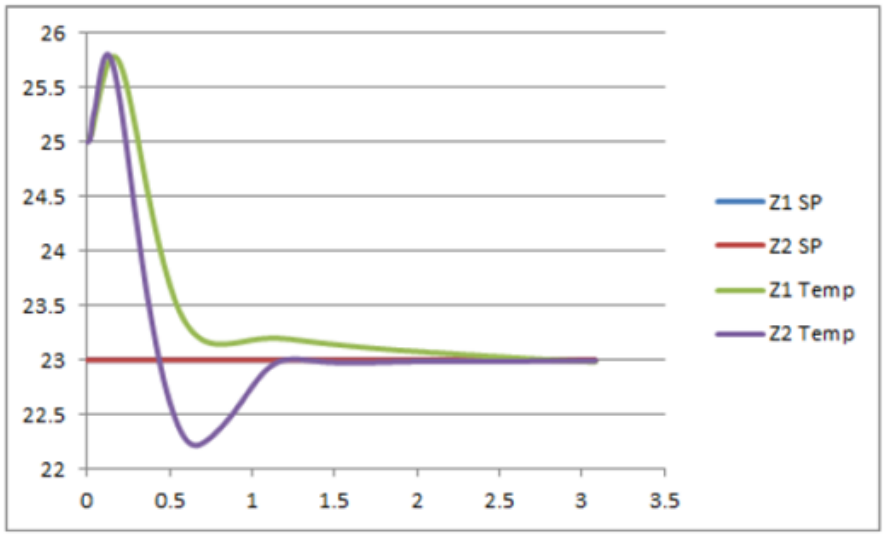
7.3 The Building Case Study

The building case study focuses on modelling and analysis of energy and comfort for Heating, Ventilation and Air Conditioning (HVAC) systems that control the temperature of connected areas inside building premises. The case study models various concepts shown in fig. 7 such as: a) Fan Coil Unit (FCU) and control; b) Supervision and fault detection of FCUs; c) Communication between master-slave FCUs; d) Communication between FCUs and supervisor; e) Air Handling Unit (AHU) and control; f) Chiller load and control; g) Physical rooms and air flow; h) Water and air pipe connections.

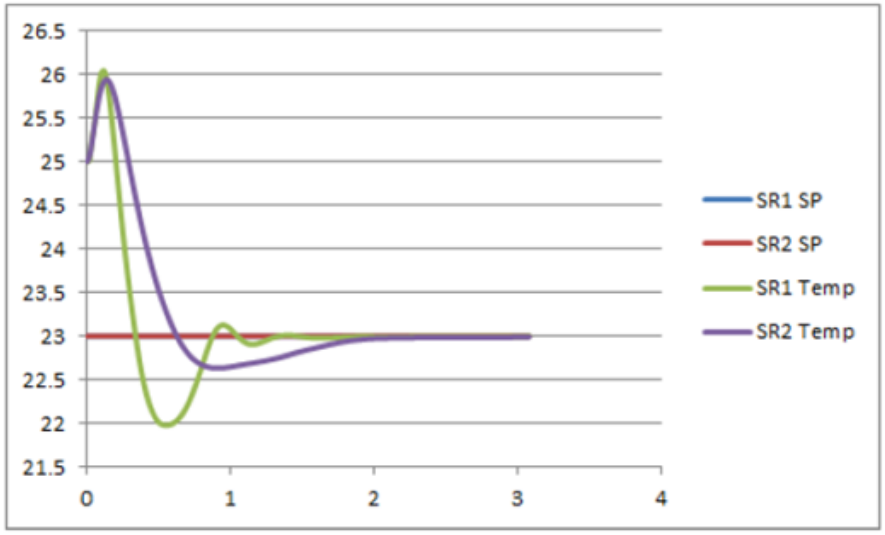
The functionality of the HVAC is to regulate operation of various devices to ensure user comfort. User inputs are taken into account from room and zone thermostats and are compared with current Room Air Temperature (RAT) sensed by the FCUs, triggering certain action on the FCUs to reach the desired temperature by a) regulating the air flow using its fan, b) regulating the water pipe valves to control the cooled water into the coil, c) synchronising with the supervisor to coordinate with the rest of the FCUs. Fresh air is provided to the FCUs by the AHU and cooled by the Chiller.

Modelling and simulation is heavily used to evaluate performance and robustness of the system with regards to requirements. Aspects such as simulation speed, and model fidelity are of high importance. Going beyond model-based design and single model simulation, co-simulation enables the analysis of physical interactions of systems that were previously not captured, due to the different domains at which the physics were modelled. Multi-physics analysis enables early analysis and detection of issues that were only uncovered at the physical prototyping stage, thus saving time and money. The 3D visualisation feature is particularly appealing for several spatial exploration of HVACs effectiveness, as well as for engaging with non-technical stakeholders and demonstrating results. In addition, certain industrial domains operate in context where the visual aspect of the 3D co-simulation can bring genuine insights. Other capabilities of the INTO-CPS tool chain such as test automation and verification are not particularly in demand for HVAC systems, but are highly valuable for aerospace applications.

Related Publications can be found at:



(a) Zone Temperature



(b) Single Rooms Temperature

Figure 7: INTO-CPS Co-Simulation Results for Room temperature in building zones

7.4 The Railway case study

In railway signalling, an interlocking is an arrangement of signal apparatus that prevents conflicting movements of trains through an arrangement of

tracks, junctions and crossings. Usually, interlocking is in charge of a complete railways or tram line, computing the status of actuators (switches and signals) based on signalling safety rules that are encoded as "binary equations" as shown in Figure 2, usually managing s 180.000 equations that have to be recalculated several times per second. These equations compute the commands to be issued to track-side devices: they encode the safety behaviour that enable trains to move from one position to another through routes that are allocated and then released. Currently, there are attempts to find the right trade-off between 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 emergency system).

In this case study, an Interlocking system was considered that controls a part of a tramway line, including two platforms and a bidirectional track (between SW5 and SW2). It involves eleven track circuits; sensors that detect the absence of a train on a railway track; three commands that can accept several positions and are activated by the train; five mechanical switches that allow changing direction (those switches have to be set accordingly to the route chosen) and three light signals, red when the train is not allowed on the track and green when it can pass. The interlocking system also makes use of five mechanical safety relays that externalise the state of a route and allow redundancy between software logic and electronic circuits.

7.5 The Aerospace case study

IDA2- Aerospace - MISSION

7.6 Integrated product-production co-simulation for cyber-physical production system (iPP4CPPS)

The case study involved the virtual design and validation of a CPS-based manufacturing system for assembling USB sticks, inspired from Continental's real manufacturing and testing processes in a production line. It is a representative example of distributed heterogeneous systems in which products, manufacturing resources, orders and infrastructure are all cyber-physical. In this setting, several features (such as asynchronous communication, messages flow, autonomy, self-adaptation, etc.) could be investigated at design time, for example using a collaborative modelling approach. Consequently, the

case study offered a balance between being sufficiently simple to be easily followed as a production line example, including generating a tangible output, and at the same time being sufficiently general to allow the study of the co-simulation complexity. Furthermore, by choosing a USB stick, the example opened the (unexplored) possibility of extending the purpose of the study to interactions between generated hardware and generated software solutions in the production line.

Obviously, this small experiment, in terms of scale and time, could not give a full and clear assessment of benefits for developing an integrated product-production co-simulation for CPS-based industrial control. Nevertheless, there were some recognisable benefits compared to the current state of technology:

- the possibility to simulate, test and validate from a holistic perspective and with an increased level of accuracy an entire production system that needs cross-functional expertise; The initial development of a homogeneous co-simulation in VDM for the iPP4CPPS prototype was particularly useful in driving cooperation and making clear the assumptions of the distributed teams involved in modelling the specific components. This phase proved to be the most difficult and time-consuming in building the co-simulation, requiring a very intensive communication for a shared understating of the requirements. Once the VDM co-simulation was running, the independent developments of units could be integrated, validated and deployed in any order.
- to a certain extent, the ability to handle unpredictable integration requirements. The employment of co-simulations when designing an automated production system avoided the build-up inertia of subsequent design constraints, facilitating the low and late commitment for these decisions, i.e. the specific micro-controllers or PLCs, the layout of the plant, the number of memory boxes from the warehouse etc. For example, the possibility to generate code - from all the simulation tools used in this experiment (i.e. 4DIAC, 20-Sim, Overture) – for an extended set of computational devices was a clear advantage in respect to late commitment for the computational system used in the production system.

The methodology adopted in the IPP4CPPS project to develop the co-simulation closely followed the classical stages of agent-oriented or component-based software engineering methodologies. Following the mechanical model derived from the requirements, the high-level abstraction for the behaviour of each simulation was implemented and the interactions among the compo-

nents could be analysed. It included distinct simulations for each component type (Table 1): production (i.e. warehouse station, robot, transporting wagons, and testing station), orders (i.e. placed via mobile devices), and factory infrastructure (i.e. part tracker).

The co-simulation model had been initially implemented in VDM and validated on the INTO-CPS tools chain. The main goal of this implementation was manifold: a) to validate the interaction protocols among the composite simulations; b) to have an early working co-simulation where the specific simulations may be gradually added, tested and validated; c) to allow for a more independent development among the dispersed teams involved in modelling the specific simulations, while at the same time keeping the co-simulation functional at all times; and d) to cover the left-over parts of the co-simulation whose modelling was not needed in detail for the validation of the interaction protocol (e.g. test station) or for which there is was FMI-compliant tool (i.e. factory infrastructure).

Component type	**Unit**	**Technology**	**Deployment**
Orders	HMI	4DIAC + MQTT *or* Overture (VDM)	smartphones and tablets
Infrastructure	Part Tracker	Overture (VDM)	NVIDIA Tegra Jetson
Production	Warehouse + Robotic Arm	20-sim	Raspberry Pi with UniPi Expansion Board + Stäubli robot
Production	Wagons	4DIAC	Raspberry Pi controlling DC motors, position sensors and anti-collision ultrasonic sensors
Production	Test Station	4DIAC	Cognex Vision Insight 1100 camera connected to Raspberry Pi for actuators control
General	Unity	20-sim animation	PC

Table 2: Technologies used for different system components

The detailed model of each simulation covered a continuous-time model realised in 20-Sim for the warehouse and robotic arm, a discrete-time model in 4DIAC for the transportation system and test station, and a discrete-time model in Overture for the infrastructure. All units modelled and tested by the heterogeneous co-simulation were then deployed in a demo stand for fine tuning under real-life conditions (Table 2). This phase presumed the extension of code generation capabilities of the simulation tools, such as: 20-sim 4C has been extended with MQTT, Modbus, I2C colour sensor, I2C multiplexer and UniPi board for the Raspberry Pi; Overture for employing MQTT as communication protocol on a Raspberry Pi 3; and 4DIAC for accelerometers control.

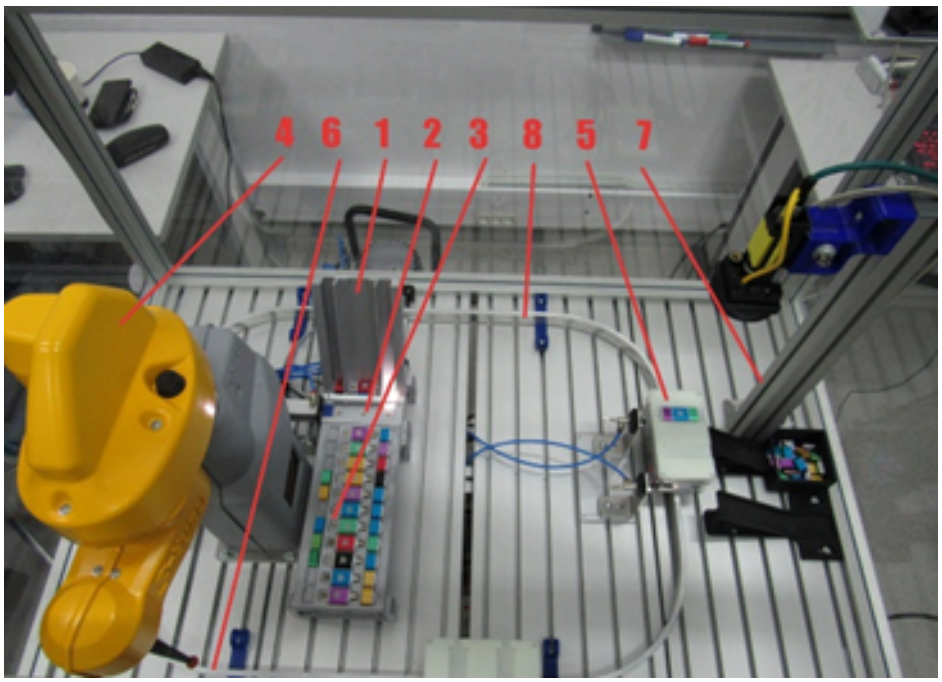


Figure 8: Demo stand for deployment of the co-simulated units, containing: 1) the warehouse stacks; 2) the assembly box at the base of the warehouse stacks; 3) the memory boxes of the warehouse unit; 4) the robotic arm for moving parts around the warehouse; 5) wagons on different locations of the track; 6) the loading station; 7) the test station; 8) the circular track for the wagons..

The experiment assessed the benefits and the maturity level of model-driven engineering technologies for future adoption into CPS-based production systems. It covered the entire engineering life-cycle (i.e. from requirements to deployment into a real infrastructure) and contributed to several advance-

ments of engineering methods and tools. The experiment delivered an effective proof-of-concept for model-driven engineering of CPS-based production system as a feasible and promising approach to (re)engineer the factory of the future with the employed technologies (i.e. INTO-CPS, Overture, 20-Sim and 4DIAC). Nevertheless, the experiment also identified a number of issues that may have further impact over the adoption of model-driven engineering technologies into real settings: the FMI-compatibility of the simulation tools used in industry; the hardware/software-in-the-loop simulations still display complex synchronisation problems for dissimilar time-scales; the extended set of low-level devices (i.e. sensors, industrial communication standards etc.) that are used in today and future industry require special standardisation effort to enhance the deployment capabilities of the simulation tools[7]

7.7 Use of the INTO-CPS Technology at MAN Diesel & Turbo

As reported in [PLS+17], at MAN Diesel & Turbo (MDT) the conventional approach for developing two-stroke combustion engines with a distributed embedded control system is being challenged. In particular, for diesel engines pollution is a key element that it is desirable to reduce from a competitive perspective. New emission legislation focuses on the reduction of especially NO_x emission. Widely known emission reduction technologies for reducing NO_x are selective catalytic reduction and Exhaust Gas Recirculation (EGR), both being developed at MDT [PLS+17].

These systems require advanced algorithms to control the complexity of the physical dynamics of large engines. MDT is divided into different departments with different responsibilities in the same way as many other large organisations. In the control department at MDT, control algorithms are created directly in the target software framework with the possibility of performing Software In the Loop (SIL) simulation during development. Models of the physical behaviour are created in other departments of MDT using the tools most suitable for the specific constituent system.

For the control system development, the physical dynamics models are implemented in an internally developed tool for Continuous-Time (CT) simulation called the Dynamic Simulation Environment (DSE) which is part of

[7]More information can be found at `http://centers.ulbsibiu.ro/incon/index.php/ipp4cpps/` and `http://www.cpse-labs.eu/experiment.php?id=c3_uk_gs_ipp4cpps`.

the software framework. The primary focus in DSE is SIL/Hardware In the Loop (HIL), and the physics models implemented here are often an abstraction of high-fidelity models. Historically it has been challenging inside MDT to enable heterogeneous collaborations between the different teams producing models in different departments. As a result different models are typically fragmented and solely used within one department for the dedicated purpose each of the models serve. Thus, efforts that goes across these individual insights are only found at the test on the real platform.

At MDT the models used in the control department are based on a software framework and DSE is implemented in C++ and run on a 32-bit Linux platform while the physical modelling tools often require Windows. During the INTO-CPS project it was illustrated how a transition from the current simulation process at MDT to one using co-simulation utilising the Functional Mock-up Interface (FMI) standard can be performed by using the Co-simulation Orchestration Engine (COE) from the Integrated Tool Chain for Model-based Design of Cyber-Physical Systems (INTO-CPS) project.

The aim with the approach suggested was to reduce redundancy in the development process and reuse and combine models from different departments [PLS+17]. One of the main challenges for such a transition is to enable co-simulation across different hardware architectures and Operating System (OS) platforms due to constraints from software frameworks, physical simulation tools and version compatibility, and INTO-CPS Technology was key in overcoming it.

7.8 Use of the INTO-CPS Technology at the European Space Agency

At the European Space Agency (ESA) many systems fall into the CPS category. As reported in [FAVL17], the Mars Rover case study, which was developed in Crescendo, a previous project, had been restricted to the combination of two models: one Discrete Event (DE) model expressed using the Vienna Development Method (VDM) [FLV08] using the Overture tool [LBF+10] and one Continuous-Time (CT) model expressed using bond graphs [KR68] and the 20-sim tool [Kle06].

Moreover, the CT model would have to be kept confidential. Thus, it was problematic to share this co-model with other parties. As reported in [FAVL17], this could only be circumvented by running the co-simulation over the Internet, with the proprietary constituent CT model running at

ESA. While technically feasible, this of course causes many other problems, such as allowing remote access through corporate firewalls, poor simulation performance, etc.

For this issue, FMI and the INTO-CPS technology offer a potential solution since the Functional Mockup Units (FMUs) produced for each constituent model do not necessarily need to contain the model itself. Thus, it is possible to protect the Intellectual Property (IP) in this manner.

In [FAVL17], the authors report on their successful attempt of migrating the Mars Rover co-model from Crescendo to the INTO-CPS technology, and how the INTO-CPS co-model enables a solution satisfying the requirements of an agency as ESA which, manages a multitude of suppliers are involved in multiple missions. The ability management of IP and ease of model construction, system of systems mission analysis, and validating on-board software were reported as successes emerging from the usage of INTO-CPS.

8 Related Work

Claudio Gomes

Claudio: Scope: related work into-cps.

9 Future Directions

It is envisaged that the INTO-CPS technology will be further extended as the FMI standard evolves and in particular in future research projects. Thus, the future directions here will depend both on the members of the INTO-CPS Association as well as which externally funded research projects that will be successful in achieving funding.

In the subsections below candidate future directions are proposed.

9.1 Adapting FMUs Easily to Ones Needs

In the context of continuous system co-simulation, it is well known that there is no one-size-fits-all co-simulation approach. Different kinds of systems are best co-simulation different ways (cite survey). At the same time, different domains have specialised numerical solvers, which means we cannot ignore the solvers in the FMUs. A future research direction is to understand how to reconcile such contradicting requirements. A possible way is to allow the user to preserve the exported FMUs, but change the way these interact with the environment, by wrapping an FMU around them [GMD+0]. This way does not solve all challenges in this regard, and the approach lacks validation from multiple domains. It is envisaged that this kind of Domain Specific Languages (DSLs) will be developed to make it easier to make semantic adaptations of FMUs.

Claudio: Maybe providing“connector” FMUs for the most common connections in different domains. For example, if I’m building a co-simulation of an hydraulic system, I may want to connect a large pipe to a small pipe.

9.2 Enlarging the tools and standards supported by the INTO-CPS Tool Suite

The INTO-CPS tool suite is on purpose open to any tool that live up to the requirements in the FMI version 2.0 standard for co-simulation. As indicated in Section 6 a possible entry point to the co-simulation setup is a special SysML profile supporting CPS models. Right now this is discussed in OMG as a potential new standard in a SysML setting and this is only supported by the Modelio tool (supporting export of both model descriptions as well as configurations of co-simulation). It would be great to see this special profile

by other SysML tools as well. In addition, one can imagine that alternative tool supporting AADL [AAD04] or Capella [Roq17].

There are also a lot of other standards for co-simulations [GTB+18b] and it is possible to imagine that bridges from the INTO-CPS tool chain will be made to a number of these. Here the most obvious candidate is High Level Architecture (HLA) [IEE10]. Initial work has been carried out in other research groups for such a combination [NGL+14, ACP17] but we imagine that more work is needed here to get this combination working smoothly.

9.3 Use in a Cloud-based Eco-system/Marketplace

The INTO-CPS Application is made using Electron so it is using web technology but still requires local installation on the local computers. It is possible to imagine that it will be possible to move this to become a cloud application where it will no longer be necessary to install it locally. The DSE feature is already available in a cloud context as explained above. In a very long context it can also be imagined that the different modelling and simulation tools at some stage will become available on-line and possibly in a cloud context.

In addition to the tools becoming available in a cloud context one can imagine that it some point in the future will be possible to share constituent models in a cloud context. Such models could then either be available in a source form or a generated form in the form of an FMU. Optimally these could include both free as well as commercial models in a marketplace setting. We believe that this could enable a shortage of time required to develop multi-models for CPSs.

9.4 Use in a Digital Twin setting

In order to increase the value of multi-models one can imagine making use of them in a deployed setting, i.e., after the CPS has been deployed if data from it can be fed back to a cloud where a co-simulation can be used to predict how alternative interventions can be made and what the consequences of these will be [GPF+18]. This is known as a *digital twin* but there are naturally a number of research challenges in connection with something like this since you need to determine how to set up such a co-simulation as well as what the consequences will be of the frequency of the data arriving in pseudo-real-time. For some types of application this could work, whereas for others

the predictions would be far from representing reality. Research is needed to determine when this would work.

9.5 Increased Support for Dynamic Evolution Scenarios

In a System of System setting it is regular that the composition of constituent systems involved in a scenario change dynamically over time. In an FMI setting this is currently not possible since it is necessary to have a static composition of the FMUs used in a co-simulation. In order to be able to support dynamic evolution in a co-simulation scenario it is desirable to conduct research exploring to what extend this could be a possibility.

9.6 Incorporation of Computational Fluid Dynamics Co-simulations

To accurately model flow of material (typically liquid or gasses) Computational Fluid Dynamics (CFD) models are typically used. These are typically represented at a very detailed level, and as a consequence CFD simulations can be really slow. In addition, in case CFD simulations fail the error control is not properly aligned with the orchestration enabled in a FMI based setting. In order to properly support CFD elements in an INTO-CPS setting research is needed to determine how this can be carried out efficiently in a semantically sound manner. Here it is imagined that a part of this will be approximating the CFDs with Reduced Order Models (ROMs) [CFCA13].

9.7 Increased support for Human Interaction

Claudio: Can you provide example advancements to support this? Example, real-time co-simulation seems an easy one. Maybe the ability to pause, inspect, and debug co-simulations? Maybe fault injection?

9.8 Increased support for Network Considerations

FMI is not by itself good at modelling the communication layers between different constituent systems. In the INTO-CPS project it was attempted to

model this as an FMU ether where messages can be lost. However, it would be ideal to be able to appropriate model each FMU being at a particular address (e.g., a URL or an IP address) and then have a library of alternative connections between such addresses, where one could experiment with non-ideal behaviour (e.g., delays and loosing messages). Such non-ideal behaviour may actually influence the way the real system behaves and thus it makes a lot of sense that it also would be possible to incorporate such aspects at a modelling level so it could be analysed either in plain co-simulations or with DSEs.

9.9 Intelligence, Adaptivity and Autonomy

Claudio: We need to make this more concrete. Do we mean the co-simulation of adaptive systems? Are we looking at variable structure systems? Or the development of a COE that is adaptive (nice challenge, to ensure stability with adaptive master algorithms)? Also, another possible direction is to support certification processes by, e.g., having built in mechanisms to calculate the errors being made (during or after the co-simulation is run). This is interesting because the user just has to provide the bounds he can tolerace, and the co-simulation will be run as many times as required, so that these tolerances are met. Another possible direction is to support hybrid co-simulation (even when there is no possibility of rolling back from the FMUs).

9.10 Tradeoff in Abstraction between Speed and Accuracy

Claudio: An interesting direction is to support DSE's, while using multiple abstraction models. For example, higher abstraction models might be used to optimize the design globally, while lower abstraction models validate the candidate found, and further optimize each of them to find the best one.

References

[AAD04] Architecture analysis & design language (aadl). Aerospace Standard AS5506, SAE Aerospace, November 2004.

[ACM+16] Nuno Am/'alio, Ana Cavalcanti, Alvaro Miyazawa, Richard Payne, and Jim Woodcock. Foundations of the SysML profile for CPS modelling. Deliverable D2.2a, version 1.0, INTO-CPS project, December 2016.

[ACP17] Muhammad Usman Awais, Milos Cvetkovic, and Peter Palensky. Hybrid simulation using implicit solver coupling with HLA and FMI. *International Journal of Modeling, Simulation, and Scientific Computing*, 2017.

[APC+15] Nuno Am/'alio, Richard Payne, Ana Cavalcanti, Etienne Brosse, and Jim Woodcock. Foundations of the SysML profile for CPS modelling. Deliverable D2.1a, version 1.0, INTO-CPS project, December 2015.

[APCB15] Nuno Amalio, Richard Payne, Ana Cavalcanti, and Etienne Brosse. Foundations of the SysML profile for CPS modelling. Technical report, INTO-CPS Deliverable, D2.1a, December 2015.

[BBG+13] David Broman, Christopher X. Brooks, Lev Greenberg, Edward A. Lee, Michael Masin, Stavros Tripakis, and Michael Wetter. Determinate composition of fmus for co-simulation. In *Proceedings of the International Conference on Embedded Software, EMSOFT 2013, Montreal, QC, Canada, September 29 - Oct. 4, 2013*, pages 2:1–2:12. IEEE, 2013.

[BC+17] Jö Brauer, Luis Diogo Couto, , Marcel Groothuis, Miran Hasanagic, and Kangfeng Ye. Demonstration of Integrated Co-Simulation and Testing. Technical report, INTO-CPS Deliverable, D5.3b, December 2017.

[BFG+12] Jan F. Broenink, John Fitzgerald, Carl Gamble, Claire Ingram, Angelika Mader, Jelena Marincic, Yunyun Ni, Ken Pierce, and Xiaochen Zhang. Methodological guidelines 3. Technical report, The DESTECS Project (INFSO-ICT-248134), October 2012.

[BFL+94] Juan Bicarregui, John Fitzgerald, Peter Lindsay, Richard Moore, and Brian Ritchie. *Proof in VDM: A Practitioner's Guide*. FACIT. Springer-Verlag, 1994. ISBN 3-540-19813-X.

[BH17] Jörg Brauer and Miran Hasanagic. Implementation of a model-checking component. Technical report, INTO-CPS Deliverable, D5.3c, December 2017.

[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.

[BLL+17] Victor Bandur, Peter Gorm Larsen, Kenneth Lausdahl, Casper Thule, Anders Franz Terkelsen, Carl Gamble, Adrian Pop, Etienne Brosse, Jörg Brauer, Florian Lapschies, Marcel Groothuis, Christian Kleijn, and Luis Diogo Couto. INTO-CPS Tool Chain User Manual. Technical report, INTO-CPS Deliverable, D4.3a, December 2017.

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

[BLV+10] J. F. Broenink, P. G. Larsen, M. Verhoef, C. Kleijn, D. Jovanovic, K. Pierce, and F. Wouters. Design Support and Tooling for Dependable Embedded Control Software. In *Proceedings of Serene 2010 International Workshop on Software Engineering for Resilient Systems*, pages 77–82. ACM, April 2010.

[BQ15] Etienne Brosse and Imran Quadri. COE Contracts from SysML. Technical report, INTO-CPS Deliverable, D4.1c, December 2015.

[BQ16] Etienne Brosse and Imran Quadri. SysML and FMI in INTO-CPS. Technical report, INTO-CPS Deliverable, D4.2c, December 2016.

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

[Bro17] Etienne Brosse. SysML and FMI in INTO-CPS. Technical report, INTO-CPS Deliverable, D4.3c, December 2017.

[BW98] Ralph-Johan Back and Joakim Wright. *Refinement Calculus: A Systematic Introduction*. Springer, 1998.

[CBRZ01] Edmund M. Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded Model Checking Using Satisfiability Solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[CE81] E. M. Clarke and A. E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *IBM Logics of Programs Workshop*, volume LNCS 131. Springer Verlag, 1981.

[CFCA13] Kevin Carlberg, Charbel Farhat, Julien Cortial, and David Amsallem. The gnat method for nonlinear model reduction: Effective implementation and application to computational fluid dynamics and turbulent flows. *Journal of Computational Physics*, 242:623 – 647, 2013.

[CGP99] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, 1999.

[CKOS04] Edmund M. Clarke, Daniel Kroening, Joël Ouaknine, and Ofer Strichman. Completeness and Complexity of Bounded Model Checking. In *5th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2004)*, volume 2937 of *Lecture Notes in Computer Science*, pages 85–96. Springer, 2004.

[CKOS05] Edmund M. Clarke, Daniel Kroening, Joël Ouaknine, and Ofer Strichman. Computational challenges in bounded model checking. *STTT*, 7(2):174–183, 2005.

[CMC+13] Joey Coleman, Anders Kaels Malmos, Luis Couto, Peter Gorm Larsen, Richard Payne, Simon Foster, Uwe Schulze, and Adalberto Cajueiro. Third release of the COMPASS tool — symphony ide user manual. Technical report, COMPASS Deliverable, D31.3a, December 2013.

[CMW13] Ana Cavalcanti, Alexandre Mota, and Jim Woodcock. Simulink timed models for program verification. In Zhiming Liu, Jim Woodcock, and Huibiao Zhu, editors, *Theories of Programming and Formal Methods - Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday*, volume 8051 of *Lecture Notes in Computer Science*, pages 82–99. Springer, 2013.

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

[CV03] E.M. Clarke and H. Veith. Counterexamples revisited: Principles, algorithms, applications. In *Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, volume 2772 of *Lecture Notes in Computer Science*, pages 208–224. Springer, 2003.

[CWA16] Ana Cavalcanti, Jim Woodcock, and Nuno Amálio. Behavioural models for FMI co-simulations. In Augusto Sampaio and Farn Wang, editors, *Theoretical Aspects of Computing - ICTAC 2016 - 13th International Colloquium, Taipei, Taiwan, ROC, October 24-31, 2016, Proceedings*, volume 9965 of *Lecture Notes in Computer Science*, pages 255–273, 2016.

[DAB+15] Lipika Deka, Zoe Andrews, Jeremy Bryans, Michael Henshaw, and John Fitzgerald. D1.1 definitional framework. Technical report, The TAMS4CPS Project, April 2015.

[DC03] Jim Davies and Charles Crichton. Concurrency and refinement in the unified modeling language. *Formal Asp. Comput.*, 15(2-3):118–145, 2003.

[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.

[FAVL17] Sergio Feo-Arenis, Marcel Verhoef, and Peter Gorm Larsen. The Mars-Rover Case Study Modelled Using INTO-CPS. In Fitzgerald, Tran-Jørgensen, Oda, editor, *The 15th Overture Workshop: New Capabilities and Applications for Model-based Systems Engineering*, pages 130–144, Newcastle, UK, September 2017. Newcastle University, Computing Science. Technical Report Series. CS-TR- 1513.

[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.

[FGP17] John Fitzgerald, Carl Gamble, and Ken Pierce. Method Guidelines 3. Technical report, INTO-CPS Deliverable, D3.3a, December 2017.

[FL98] John Fitzgerald and Peter Gorm Larsen. *Modelling Systems – Practical Tools and Techniques in Software Development*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1998. ISBN 0-521-62348-0.

[FLG17] Frederik Foldager, Peter Gorm Larsen, and Ole Green. Development of a Driverless Lawn Mower using Co-Simulation. In *1st Workshop on Formal Co-Simulation of Cyber-Physical Systems*, Trento, Italy, September 2017.

[FLV08] J. S. Fitzgerald, P. G. Larsen, and M. Verhoef. Vienna Development Method. *Wiley Encyclopedia of Computer Science and Engineering*, 2008. edited by Benjamin Wah, John Wiley & Sons, Inc.

[FLV14] John Fitzgerald, Peter Gorm Larsen, and Marcel Verhoef, editors. *Collaborative Design for Embedded Systems – Co-modelling and Co-simulation*. Springer, 2014.

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

[GF94] Orlena C.Z. Gotel and Anthony C.W. Finkelstein. An analysis of the requirements traceability problem. In *Proc. 1st Intl. Conf. on Requirements Engineering*, pages 94–101, April 1994.

[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.

[GMD+0] Claudio Gomes, Bart Meyers, Joachim Denil, Casper Thule, Kenneth Lausdahl, Hans Vangheluwe, and Paul De Meulenaere. Semantic adaptation for fmi co-simulation with hierarchical simulators. *SIMULATION*, 0(0):0037549718759775, 0.

[GPF+18] Carl Gamble, Richard Payne, John Fitzgerald, Sadegh Soudjani, Frederik F. Foldager, and Peter Gorm Larsen. Automated Exploration of Parameter Spaces as a Method for Tuning a Predictive Digital Twin. *Submitted for publication*, 2018.

[GTB+17] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. Co-simulation: State of the art. *CoRR*, abs/1702.00686, 2017.

[GTB+18a] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. Co-simulation: A survey. *ACM Comput. Surv.*, 51(3):49:1–49:33, 2018.

[GTB+18b] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. Co-simulation: a Survey. *ACM Comput. Surv.*, 51(3):49:1–49:33, May 2018.

[HIL+14] J. Holt, C. Ingram, A. Larkham, R. Lloyd Stevens, S. Riddle, and A. Romanovsky. Convergence report 3. Technical report, COMPASS Deliverable, D11.3, September 2014.

[HJ98] Tony Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice Hall, April 1998.

[Hoa85a] C.A.R. Hoare. *Communicating Sequential Processes*. International Series in Computer Science. Prentice Hall, 1985.

[Hoa85b] Tony Hoare. *Communication Sequential Processes*. Prentice-Hall International, Englewood Cliffs, New Jersey 07632, 1985.

[IEE10] IEEE Standard for Modeling and Simulation: High Level Architecture (HLA)– Framework and Rules. *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000)*, pages 1 –38, August 2010.

[Inc] MathWorks Inc. Simulink. `www.mathworks.com/products/simulink`.

[INC15] INCOSE. Systems Engineering Handbook. A Guide for System Life Cycle Processes and Activities, Version 4.0. Technical Report INCOSE-TP-2003-002-04, International Council on Systems Engineering (INCOSE), January 2015.

[Int14] Functional Mock-Up Interfaace. Functional mock-up interface for model exchange and co-simulation. Technical Report 2.0, FMI development group, 2014.

[Jif94] He Jifeng. A classical mind. chapter From CSP to Hybrid Systems, pages 171–189. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1994.

[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.

[Kle06] Christian Kleijn. Modelling and Simulation of Fluid Power Systems with 20-sim. *Intl. Journal of Fluid Power*, 7(3), November 2006.

[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.

[KS00] R. Kübler and W. Schiehlen. Two methods of simulator coupling. *Mathematical and Computer Modelling of Dynamical Systems*, 6(2):93–113, 2000.

[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.

[LMR+17] Wei Li, Alvaro Miyazawa, Pedro Ribeiro, Ana Cavalcanti, Jim Woodcock, and Jon Timmis. From formalised state machines to implementations of robotic controllers. *CoRR*, abs/1702.01783, 2017.

[MCR+16] A. Miyazawa, A. Cavalcanti, P. Ribeiro, W. Li, J. Woodcock, and J. Timmis. RoboChart Reference Manual. Technical report, University of York, feb 2016.

[MG13] Luc Moreau and Paul Groth. PROV-Overview. Technical report, World Wide Web Consortium, 2013.

[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.

[Mor90] Carroll Morgan. *Programming from Specifications*. Prentice-Hall, London, UK, 1990.

[MZC12] Chris Marriott, Frank Zeyda, and Ana Cavalcanti. A tool chain for the automatic generation of circus specifications of simulink diagrams. In John Derrick, John S. Fitzgerald, Stefania Gnesi, Sarfraz Khurshid, Michael Leuschel, Steve Reeves, and Elvinia Riccobene, editors, *Abstract State Machines, Alloy, B, VDM, and Z - Third International Conference, ABZ 2012, Pisa, Italy, June 18-21, 2012. Proceedings*, volume 7316 of *Lecture Notes in Computer Science*, pages 294–307. Springer, 2012.

[NGL+14] Himanshu Neema, Jesse Gohl, Zsolt Lattmann, Janos Sztipanovits, Gabor Karsai, Sandeep Neema, Ted Bapty, John Batteh, Hubertus Tummescheit, and Chandrasekar Sureshkumar. Model-based integration platform for fmi co-simulation and heterogeneous simulations of cyber-physical systems. In *The 10th International Modelica Conference 2014*, Lund, Sweden, March 2014. Modelica Association.

[NK14] Tobias Nipkow and Gerwin Klein. *Concrete Semantics: With Isabelle/HOL*. Springer, 2014.

[NLF+13] Claus Ballegaard Nielsen, Peter Gorm Larsen, John Fitzgerald, Jim Woodcock, and Jan Peleska. Model-based engineering of systems of systems. *Submitted to ACM Computing Surveys*, June 2013.

[NLFS18] Pierluigi Nuzzo, Michele Lora, Yishai A. Feldman, and Alberto L. Sangiovanni-Vincentelli. CHASE: contract-based requirement engineering for cyber-physical system design. In *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*, pages 839–844. IEEE, 2018.

[NN92] Hanne Riis Nielson and Flemming Nielson. *Semantics With Applications – A Formal Introduction*. John Wiley & Sons Ltd, 1992.

[OLF+17] Julien Ouy, Thierry Lecomte, Frederik Forchhammer Foldager, Andres Villa Henriksen, Ole Green, Stefan Hallerstede, Peter Gorm Larsen, Luis Diogo Couto, Pasquale Antonante, Stylianos Basagiannis, Sara Falleni, Hassan Ridouane, Hajer Saada, Erica Zavaglio, Christian König, and Natalie Balcu. Case

Studies 3, Public Version. Technical report, INTO-CPS Public Deliverable, D1.3a, December 2017.

[OMG12] OMG. OMG Systems Modeling Language (OMG SysML), Version 1.3. Technical report, Object Management Group, 2012.

[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.

[PF10] Richard J. Payne and John S. Fitzgerald. Evaluation of Architectural Frameworks Supporting Contract-based Specification. Technical Report CS-TR-1233, School of Computing Science, Newcastle University, December 2010.

[PHP+14] Simon Perry, Jon Holt, Richard Payne, Jeremy Bryans, Claire Ingram, Alvaro Miyazawa, Luís Diogo Couto, Stefan Hallerstede, Anders Kaels Malmos, Juliano Iyoda, Marcio Cornelio, and Jan Peleska. Final Report on SoS Architectural Models. Technical report, COMPASS Deliverable, D22.6, September 2014. Available at http://www.compass-research.eu/.

[Plo81] Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.

[PLS+17] Nicolai Pedersen, Kenneth Lausdahl, Enrique Vidal Sanchez, Peter Gorm Larsen, and Jan Madsen. Distributed Co-Simulation of Embedded Control Software with Exhaust Gas Recirculation Water Handling System using INTO-CPS. In *Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2017)*, pages 73–82, Madrid, Spain, July 2017. ISBN: 978-989-758-265-3.

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

[RABR16] Gibson-Thomas Robinson, Philip Armstrong, Alexandre Boulgakov, and A. W. Roscoe. Fdr3: A parallel refinement checker

for csp. *Int. J. Softw. Tools Technol. Transf.*, 18(2):149–167, apr 2016.

[Roq17] Pascal Roques. *Systems Architecture Modeling with the Arcadia Method*. Elsevier, 1st edition, November 2017.

[RW05] Holger Rasch and Heike Wehrheim. Checking the validity of scenarios in UML models. In Martin Steffen and Gianluigi Zavattaro, editors, *Formal Methods for Open Object-Based Distributed Systems, 7th IFIP WG 6.1 International Conference, FMOODS 2005, Athens, Greece, June 15-17, 2005, Proceedings*, volume 3535 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2005.

[SS71] Dana Scott and Christopher Strachey. Towards a mathematical semantics for computer language. Technical Report PRG-6, Oxford Programming Research Group Technical Monograph, 1971.

[Sys12] OMG Systems Modeling Language (OMG SysML^{TM}). Technical Report Version 1.3, SysML Modelling team, June 2012. http://www.omg.org/spec/SysML/1.3/.

[Tho13] Haydn Thompson, editor. *Cyber-Physical Systems: Uplifting Europe's Innovation Capacity*. European Commission Unit A3 - DG CONNECT, December 2013.

[TLLM18] Casper Thule, Kenneth Lausdahl, Peter Gorm Larsen, and Gerd Meisl. Maestro: The INTO-CPS Co-Simulation Orchestration Engine. 2018. Submitted to Simulation Modelling Practice and Theory.

[UPL06] Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing. Technical Report 04/2006, Department of Computer Science, University of Waikato, Hamilton, New Zealand, 2006.

[vA10] Job van Amerongen. *Dynamical Systems for Creative Technology*. Controllab Products, Enschede, Netherlands, 2010.

[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.

[WG-92] RTCA SC-167/EUROCAE WG-12. Software Considerations in Airborne Systems and Equipment Certification. Technical Report RTCA/DO-178B, RTCA Inc, 1140 Connecticut Avenue, N.W., Suite 1020, Washington, D.C. 20036, December 1992.

[WLBF09] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John S. Fitzgerald. Formal methods: Practice and experience. *ACM Comput. Surv.*, 41(4):19:1–19:36, 2009.

[ZCWO17] Frank Zeyda, Ana Cavaclcanti, Jim Woodcock, and Julien Ouy. SysML foundations for INTO-CPS. Deliverable D2.3a, version 1.0, INTO-CPS project, December 2017.

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
CFD	Computational Fluid Dynamics
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
DSL	Domain Specific Language
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
HLA	High-Level Architecture
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

ROM	Reduced Order Model
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 9 shows 20-sim with a model of a controlled

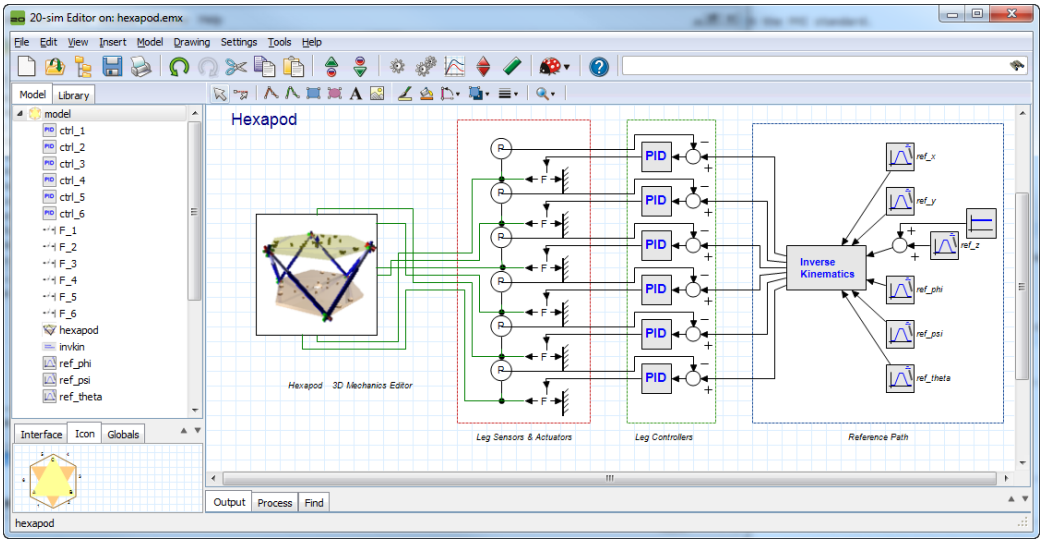


Figure 9: Example of a hexapod model in 20-sim.

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 10, 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].

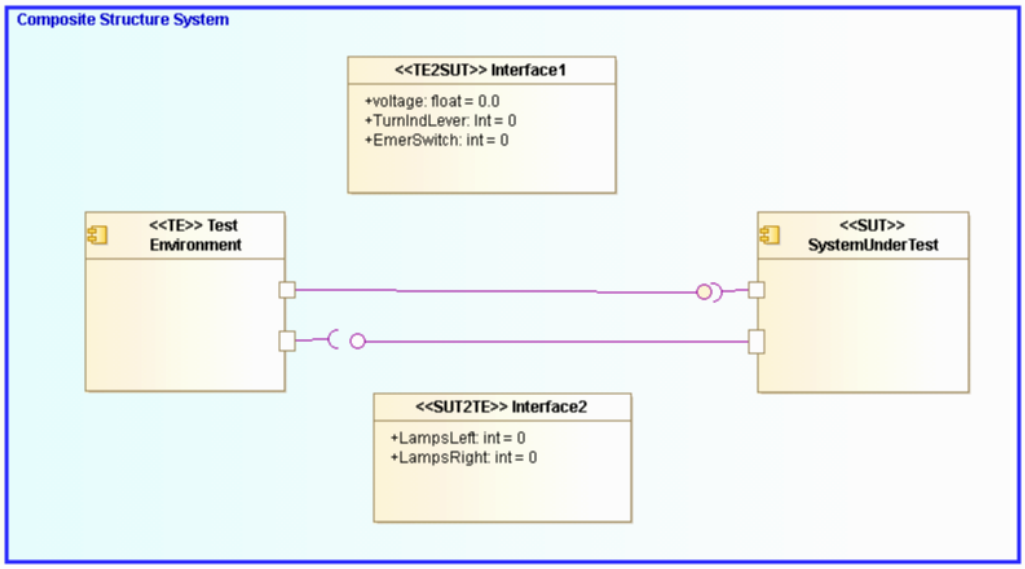


Figure 10: Simple model that highlights interfaces between the environment and the system-under-test.

C.6 4DIAC

Jose Cabral

C.7 AutoFOCUS-3

Jose Cabral