INtegrated TOol chain for model-based design of CPSs

INTO-CPS

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

Version: 1.0

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
0.1	23-04-2018	Peter Gorm Larsen	Initial version.
0.2	01-06-2018	Hugo Daniel Macedo	Added MAN D&T text.
0.3	04-07-2018	Mihai Neghina	Added IPP4CPPS case.
0.4	20-07-2018	Christian König	Added tool chain section.
0.5	26-07-2018	Stylianos Basagiannis	Completed case studies.
0.6	20-08-2018	Jim Woodcock	Added section of foundations.
0.7	23-08-2018	Claudio Gomes	Added related work section.
0.8	01-09-2018	John Fitzgerald	Added methods section.
0.9	21-09-2018	John Fitzgerald	Edited methods section.
1.0	07-10-2018	Peter Gorm Larsen & John Fitzgerald	Global Edit.

Abstract

The successful design, implementation and maintenance of Cyber-Physical Systems (CPSs) 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 tool chains. The INTO-CPS technology is described briefly, and the semantic foundation of the approach is outlined. Methods for exploiting the toolchain within existing systems engineering processes are outlined, and the current toolchain itself is described alongside several 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 digital twins using machine learning in order to gain intelligence.

Contents

1 Introduction

Cyber-Physical Systems (CPSs) present major business and societal opportunities in a variety of application areas— *if* they can be developed economically [CBM+13]. Model-Based Development (MBD) has the potential to enhance the development of CPSs, increasing the competitiveness of industry by shortening time to market and reducing development costs. In the interface between disciplines, different formalisms and technical cultures meet, and the traditional approaches for designing systems vary significantly among the relevant fields. Some researchers advocate for describing such hybrid systems using a single formalism/tool [Pto14, Pla18], but here we believe that it is better to enable stakeholders with different disciplinary backgrounds to produce their constituent models using their preferred formalism/tool and then enable joint analysis using co-simulation [GTB+18] and ensuring that there is an underlying common foundation for all of them.

Different stakeholders can produce constituent models of the parts they are responsible for, and we will call these constituent models that together form the CPS. The main challenge is to ensure that such constituent models connect and thus can be combined in different analysis conducted of the behaviour of the CPS in its desired surroundings, typically called its environment. The design of CPSs involves the usage of results obtained using a combination of different formalisms serving different engineering disciplines.

Different research projects have targeted the development of chains of tools which collectively would enable the envisaged combination of different formalisms and tools in the development of CPSs. The DESTECS[1] project [BLV+10] combined the Overture/VDM tool [LBF+10] with the 20-sim tool [Kle06] with a dedicated co-simulation combination with a Crescendo tool [FLV14]. The MODELISAR project[2] developed an open standard for interfacing between different constituent models called the Functional Mockup Interface (FMI) enabling co-simulation between any tool supporting this standard maintained by the Modelica Association[3]. The INTO-CPS project[4] took this further with a tool chain going all the way from requirements to final realisations using the FMI standard. This developed the INTO-CPS technol-

[1] This is an acronym for "Design Support and Tooling for Embedded Control Software", see `http://destecs.org/`.

[2] See `https://itea3.org/project/modelisar.html`.

[3] See `https://www.modelica.org/`.

[4] See `http://projects.au.dk/into-cps/`.

ogy which consists of 1) a common semantic foundation, 2) a methodology with guidelines for the development of CPS and 3) an open tool chain.

Before the end of the INTO-CPS project the Intellectual Property (IP) developed was transferred to the non-profit INTO-CPS Association[5]. The Association maintains and further develops the INTO-CPS technology and grows the open tool chain by adding additional tools from its partners, maintaining the documentation and tutorial material. It attempts to ensure that the different tools are kept in sync and that the importable examples expressed using diverse notations and tools are kept up to date.

This guide to INTO-CPS begins with an overview of challenges in the engineering of CPSs in Section 2. Then Section 3 provides a short overview of the INTO-CPS project in a nutshell. Section 4 gives an overview of the CPS foundations; Section 5 gives an overview of the CPS methodology; and Section 6 an overview of the tool chain. The industrial use of the INTO-CPS technology is summarised in Section 7. Finally, Section 8 provides an overview of related work, and Section 9 looks at potential future directions for the INTO-CPS technology. There are two appendices: Appendix A provides a list of acronyms, and Appendix B is an overview of the individual tools used in the INTO-CPS tool chain.

[5]This is registered as a legal entity in Denmark, see `into-cps.org/`.

2 Challenges in Engineering CPSs

The vision underpinning INTO-CPS is that teams of developers from diverse disciplines and organisations are enabled to collaborate and converge more rapidly than today on system designs that perform optimally. Realising this vision requires methods and tools that allow each discipline and organisation to make its contribution without compromising its intellectual property or significantly altering its well-tried and established techniques. It should be possible to federate these diverse design artefacts to allow analysis of the system-level consequences of design decisions made in any one domain, and the trade-offs between them. How, then, can we use semantically well-founded approaches to support such multidisciplinary design in a cost-effective way? This question poses several significant challenges.

2.1 Time to Market

There is a clear need for model-based methods that permit early design space exploration, optimisation and experimentation without delaying the launch of a new product to the market. Important issues here are:

Faster route to market for engineering CPSs: In a highly active CPS marketplace, getting the right solution first time is essential. We believe that the interoperability of tools in the INTO-CPS tool suite enables a more agile close collaboration between stakeholders with diverse disciplinary backgrounds.

Exploring large design spaces efficiently: CPS design involves making design decisions in both the cyber and physical domains. Trade-off analysis can be challenging. Co-simulation enables the systematic exploration of large design spaces in the search for optimal solutions.

Limiting expensive physical tests: CPS development often relies on the expensive production and evaluation of a series of physical prototypes. Co-simulation enables users to focus on testing different models of CPS elements in a virtual setting, gaining early assessment of CPS-level consequences of design decisions.

2.2 Diversity of Design Models

Disciplines such as software, mechatronic and control engineering have evolved notations and theories that are tailored to their needs. It is undesirable to suppress this diversity by enforcing uniform general-purpose models [FGL+15, LFW+16b]. The semantics of these notations and theories will have very different foundations in discrete or continuous domains. The goal, then, must be to support the effective federation of such highly diverse design models.

2.3 Collaboration

There is a clear need to provide mechanisms to support collaborative model-based engineering without compromising the independence of contributors. Important issues here are:

Avoiding vendor lock-in by open tool chain: Some commercial solutions provide at least a part of the functionality provided by the INTO-CPS tool chain with a high level of interoperability. However, in particular for Small and Medium-sized Enterprises (SMEs), there is a risk of being restricted in the choice of specialist tools.

Traceability and Provenance: CPS development often relies on the expensive production and evaluation of a series of physical prototypes. Co-simulation enables users to focus on testing different models of CPS elements in a virtual setting, gaining early assessment of CPS-level consequences of design decisions.

The foundations, methods and tools of CPS engineering should incorporate both the Discrete-Event (DE) models of computational processes, and the continuous-value and Continuous-Time (CT) formalisms of physical dynamics engineering. Our approach is to support the development of collaborative models containing DE and CT elements expressed in diverse notations, and to support their analysis by means of co-simulation based on a reconciled operational semantics of the individual notations' simulators [FLV14]. This enables exploration of the design space and allows relatively straightforward adoption in businesses already exposed to some of these tools and techniques.

3 INTO-CPS in a Nutshell

To address the challenges presented above in Section 2, the 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.

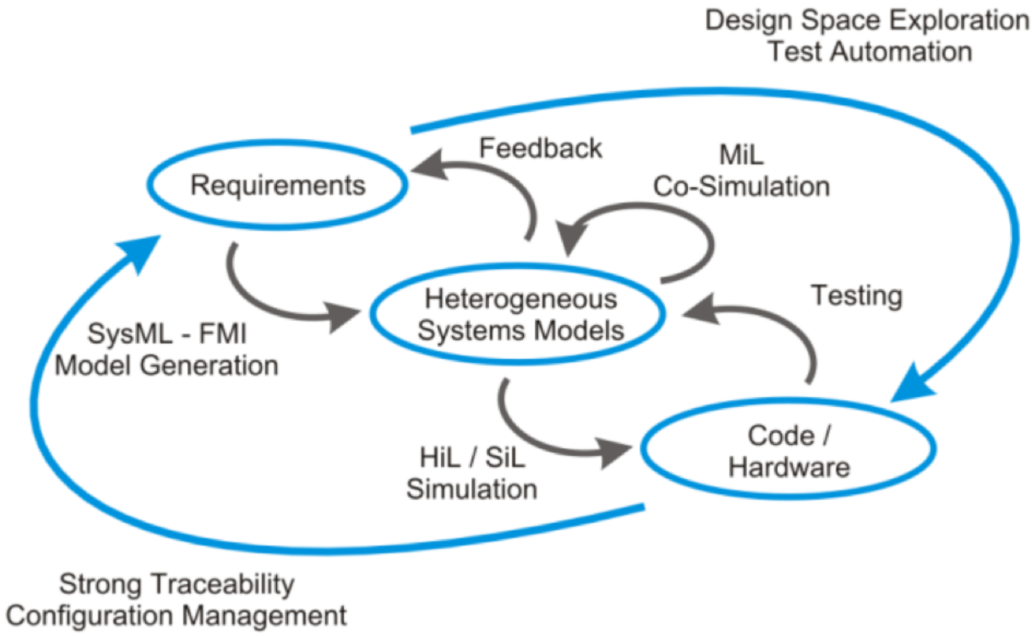


Figure 1: Connections in the INTO-CPS tool chain.

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

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

A Co-simulation Orchestration Engine (COE) 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[6].

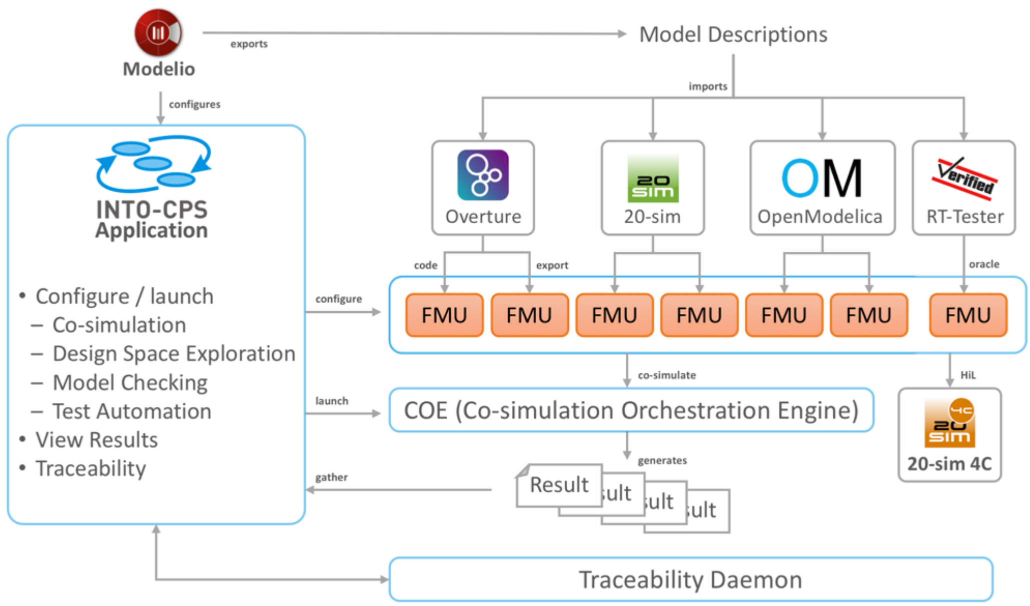


Figure 2: Overview of the INTO-CPS tool chain.

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

[6]These connections was all established inside the INTO-CPS project. See Figure **??** for the full current tool chain.

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

3.2 Industrial 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. The foundations are described in more detail in Section 4.

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. The methods and guidelines are described in more detail in Section 5.

4 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. We have formalised and mechanised both languages in this framework using Isabelle/UTP. We have a semantics of the relevant parts of SysML that can be used with FMI. These foundations allow for the formal checking of the validity of analysis and co-simulation results. The value of the foundational tools, with both theorem proving and model checking, have been demonstrated to add value to the INTO-CPS tool chain.

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 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+17a, GTB+18], the coupling of subsystem simulations. This constitutes the backdrop of the industrial Functional Mockup Interface (FMI) standard [FMI14, BBG+13b, CWA16a] for co-simulation of components built using distinct modelling tools. The 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+13b]. The work in the INTO-CPS project 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 [Hoa85] with its accompanying FDR3 refinement-checker [RABR16].

Our technique is based on abstraction: we use a relational view of FMUs 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 [Inca]. 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 [CWA16a] 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 [CWA16a]. 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.

INTO-CPS multi-models are composed of individual models whose foundations lie in a variety of modelling notations, each of which has its own unique syntax, semantics, and underlying paradigmatic concepts, such as discrete or continuous time. The purpose of a multi-model is to assign behaviour to a CPS by composing the behaviours of the constituent models. Thus, in order to provide an integrated tool chain for trustworthy CPS development, there is a necessity for unification of these underlying semantic models to allow consistent integration of heterogeneous system components. This will then allow us to substantiate statements made about the multi-model with respect to the underlying mathematical core. Hoare and He's UTP [HH98] has been designed as a framework in which the integration of languages, through the common semantic domain of the alphabetised relation calculus, can be achieved. In the next two sections, we describe how UTP is used to provide the foundations for continuous-time modelling in the INTO-CPS tool chain.

4.2 Discrete Event Models

VDM-RT is a real-time dialect of the VDM formal modelling language that can be applied to the specification of discrete controllers for CPSs. VDM-RT is object oriented, where all models are defined as classes that are instantiated as objects. It supports concurrency through threading and communication between threads through shared objects. The real-time features of the language comprise abstractions for deployment of objects to computing units that are connected by buses, and the time taken to evaluate expressions that advance a global "wall clock" to predict the computation time of a model.

A denotational semantics exists for the core specification language [LP95], and a structured operational semantics (SOS) exists for the real-time aspects [LCL13], but there is currently no full semantic description of VDM-RT. To address this, the INTO-CPS project has established a comprehensive denotational semantics for the VDM-RT language, including object orientation, real time, and concurrency.

We have given a UTP semantics to the language, and mechanised this in the Isabelle/UTP theorem prover. The basis for our treatment of object orientation is an extended calculus for classes and objects, including novel healthiness conditions that allow handling of multiple inheritance. Our semantics includes a new approach to handling static attributes, methods, and constructors. We have mechanised Lausdahl's operational semantics of VDM-RT [LCL13] in Isabelle/HOL, which allowed us to gain greater insight into the language.

We use UTP [HH98, CW06] to give a denotational semantics to VDM-RT. VDM-RT is a discrete real-time language, which leads us to employ the UTP theory of timed reactive designs as the semantic model, as embodied in the COMPASS Modelling Language (CML) [WCF+12, Woo14, WCF+14]. We use the constructs of CML to describe VDM-RT objects, threads, CPUs, and busses, together with actions that encode their orchestrated execution. In order to accomplish this, we also extend CML with a universe type for VDM-RT, and also timed expressions that cause language constructs like assignment to expend time during execution.

Our semantics of VDM-RT is based on a pattern commonly employed in the INTO-CPS project to describe the discrete time component of a CPS. Such a "cyber component" consists of one or more controller objects, each of which owns a number of sensors and actuators through which to interact with the physical components. The topology of such a cyber component is thus fixed at instantiation, and there is no necessity to support dynamic object creation, which thus favours the use of static CML processes to represent objects and threads. Limiting ourselves to static topologies enables the application of static analysis techniques like model checking [GRABR14, OSF14, BB15], which typically requires a tractable state space.

4.3 Continuous Models

Modelling of continuous dynamical systems in the INTO-CPS tool chain is provided by the Modelica and 20-sim tools, both of which are based on differential equations. We have created a formal denotational semantics for continuous-time models written using the Modelica language [Mod14]. The creation of such a semantics provides firm mathematical foundations for the language, allowing us to consider formal links between Modelica and other languages in INTO-CPS, and enabling theorem-proving support for continuous models. The Modelica language supports modelling based on ordinary differential equations (ODEs) and differential algebraic equations (DAEs) combined with an event handling mechanism.

We have provided a *flattening* process, whereby a collection of Modelica objects is converted to a pure hybrid DAE system, the core of the Modelica language. In our work, we have compared this to the FMI representation of Hybrid ODEs. Once more, we have used the UTP semantic framework [HH98] to give Modelica a formal semantics, along with other continuous time and dynamical systems modelling languages. Our theory of differential algebraic equations allows the definition of hybrid programs that mix continuous and discrete behaviour, and also specifications regarding their behaviour.

We have mechanised our UTP theory in the established Isabelle/HOL proof assistant [NWP02]. This allowed us to also show that our calculus satisfies well-known laws of programming. Our combination of continuous invariants with timed reactive designs [HDM10, CW15], forms the basis of a refinement technique for hybrid systems.

Our hybrid combination of discrete and continuous models is known as *CyPhyCircus*. We have defined mappings from the core languages in *CyPhyCircus* which, as illustrated Figure 3, will also enable access to a number of static analysis tools and techniques, such as model checking [OSF14, BB15, Beg16] and theorem proving [FZW14, FZW16, ZFF16]. *CyPhyCircus* build on the existing work of the *Circus* language family [WC01, OCW07, Woo14], a suite of formal languages that combines rich state modelling (like as in the Z specification language [WD96]) with concurrency (as in CSP [Hoa85]), with various other programming paradigms such as object orientation [CSW05] and discrete real-time modelling [WWC13]. The intention is to have a language that combines rich-state modelling, concurrent reactive processes, real-time modelling, continuous variables, and differential equations. The theory of hybrid relations provides the foundations for such hybrid dynamical behaviour in *CyPhyCircus*.

4.4 Functional Mock-up Interface

Because CPSs comprise both real-world entities and digital components, their modelling and designing typically requires a combination of different languages and tools that adopt complementary specification paradigms. For real-world artefacts, physics models in the form of differential equations are the norm. Digital components, such as software controllers, are typically described via control diagrams, state machines, and real-time programs. This diversity of specification and design methods makes CPS challenging to study and analyse.

Co-simulation [GTB+17a] is perhaps the de facto technique for analysing the behaviour of CPS. It requires that models of artefacts are simulated in isolation, while master algorithms control the various simulators and thereby orchestrate the co-simulation as a whole. This raises issues of interoperability between the master algorithm and the simulators. The Functional Mock-up Interface (FMI) Standard [BOA+11] has been proposed to alleviate those

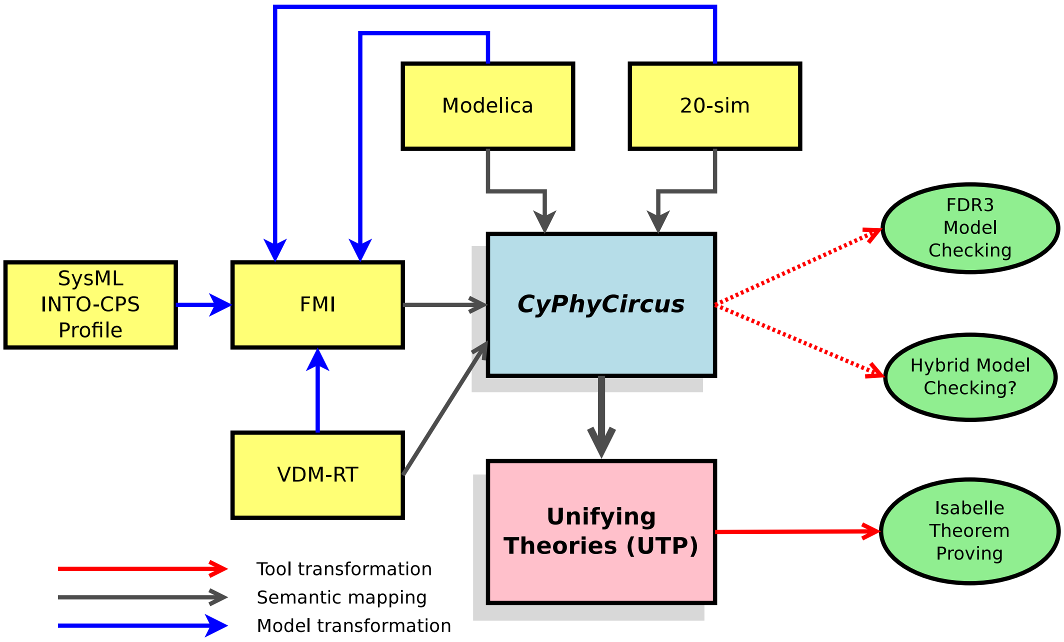


Figure 3: *CyPhyCircus* as the INTO-CPS lingua franca

issues, and has since been successfully used in many industrial applications. The FMI standard prescribes how master algorithms (MA) and simulators communicate. It does so using a bespoke API that simulators have to implement, and that can be used to devise compliant master algorithms. The API enables master algorithms to exchange data between the components of a co-simulation, called FMUs (Functional Mock-up Units), perform simulation steps, and suitably deal with errors in simulators. It also allows for advanced features such as roll-back of already performed steps.

While (co)simulation is currently the predominant approach to validate CPS models, the INTO-CPS approach uses a complementary technique based on a formal model of an FMI system. Our technique formalises both the master algorithm and the simulated FMUs, and allows for verification of their properties.

Whereas (co)simulation helps engineers to quickly gauge the implications of modelling and design decisions, our formal analysis has the potential to complement simulation with universal guarantees, both about the master algorithm and co-simulated system. The former is important since simulations depend on parameters and algorithms, and are software systems (with possible faults) in their own right. The latter is important since it is usually not possible to run an exhaustive number of simulation scenarios as a means of testing the system towards producing strong certification evidence.

For our formal modelling, we use Circus: a process algebra with added features for supporting stateful models. It has proved adequate and useful for modelling master algorithms [CWA16b] due to its capabilities of capturing concisely the data and control aspects of such algorithms, including data exchange between the FMUs and their concurrent execution. Circus models can be subjected to verification techniques. These include both model-checking approaches [**?**], refinement [**?**], and (automatic) theorem proving [**?**].

We use an abstract relational model of FMI co-simulations that focuses on the essence of the FMI computational paradigm [D2.3a]. We have considered a concrete reactive model of FMI that faithfully models the FMI interface as well as master algorithms. This extends and elaborates the work in [D2.2c] by providing a comprehensive Circus model that has been mechanised in the theorem prover Isabelle/UTP [**?**].

We have presented a complete and final Circus model of the FMI standard for co-simulation. To accomplish this, we have embedded the Circus language into Isabelle/UTP, allowing us to mechanise our FMI Circus model. In [D2.3c], we illustrated the use of our mechanisation by applying it to one of the industrial INTO-CPS case studies (a railways system).

For proof support, we pursue a technique, based on refinement, to show compliance of master algorithms with regards to the FMI standard citeModelica2014. Unlike other approaches, such as [BBG+13a], we can profit from high-level algebraic laws and a stepwise approach that culminates in executable code, for both the FMUs and master algorithm.

We have shown how our work completes the general reasoning technique presented in [D2.3a]. That technique proposes a refinement-based approach: we start with a discrete abstraction of a co-simulation that does not need to consider the master algorithm and is used to establish fundamental safety properties. Our work has filled an important gap: the transformation of an abstract FMU model into a concrete one that can be translated into code.

5 INTO-CPS Method Guidelines

5.1 Introduction

The INTO-CPS tool chain enables collaborative multidisciplinary model-based design of CPSs. Although each discipline involved in a CPS engineering enterprise has its own culture, abstractions, and approaches to problem solving, it may only be on federating them that knowledge which is otherwise tacit in some disciplines has to be made explicit. To date, there is only limited experience in model-based multidisciplinary design of CPSs, and so the methods and approaches for bringing models together are only beginning to emerge. This section aims to distil the methods and guidelines that have emerged in our experience with the INTO-CPS toolchain, and to do so in a way that helps the reader understand how best to use INTO-CPS co-modelling technologies.

This section complements Section 6 and the Tool Chain User Manual [LBL+18] — which give detail on how to use the features of the tool chain. Here we provide guidance on when and why these features might be used. The guidance given here has been distilled from experience gained through pilot studies and applications of INTO-CPS technologies to real industrial cases. These pilot studies now appear as examples that can be opened directly from the INTO-CPS Application, supported by descriptions in the Examples Compendium [FPG+18]. Sections 5.1–5.3 provide an introduction to core INTO-CPS terminology and the activities that INTO-CPS enables. Advanced topics including traceability, design space exploration and architectural modelling are outlined in Section 5.5.

5.2 Concepts and Terminology

Given the diversity of backgrounds in CPS engineering teams, it is worth clarifying some common concepts.

5.2.1 Systems

A ***System*** is "a combination of interacting elements organized to achieve one or more stated purposes" [WRF+15]. 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]. A ***system boundary*** is the common frontier between a system of interest and its environment [BFG+12]. An ***interface*** is a shared boundary between two entities, which can be defined in terms of physical and digital interactions and flows [WRF+15].

Cyber-Physical Systems (CPSs) are "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 independent systems, integrated into a larger system that delivers unique capabilities" [INCb].

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]

In a model of a CPS, we describe systems with cyber, physical and network elements. These components are often modelled in a variety of languages, with different notations, concepts, levels of abstraction, and semantics, which are not necessarily easily mapped one to another. 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 or 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]. We use the term ***multi-model*** to refer to the federation of several constituent DE and CT models.

A ***requirement*** may impose restrictions, define capabilities or identify qualities of a system, and should indicate some value or use for the stakeholders in a CPS. ***Requirements Engineering (RE)*** is the process of specifying and documenting 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 measure [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.

5.2.3 Architectures and Architectural Models

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*** and they describe computational or physical elements, respectively.

We consider both ***holistic*** and ***design*** architectures (Section 5.5.3). The aim of a holistic architecture is to identify the units of functionality of the system reflecting the *terminology and structure of the domain of application.* It describes a conceptual model of these units and their interconnections, giving a holistic view of the overall system. The design architectural model of the system is effectively a multi-model. The INTO-CPS SysML profile is designed to assist in the specification of CPS design architectures. It helps the architect describe a system as a decomposition into interconnected ***subsystems***, each of which is an assembly of cyber and physical components and possibly other subsystems. Each of these components and subsystems can be modelled separately in a domain-specific notation and tool.

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.

There are many methods of describing architectures. An ***architecture diagram*** is a 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. 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].

5.2.4 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 initialising 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 synchronising 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.

5.2.5 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 (DSE)***. 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.

5.2.6 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.

5.2.7 Code Generation

Code generation refers to the translation of a modelling language to a common programming language. This 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.

5.3 Activities Enabled by INTO-CPS

The following activities are all enabled by one or more of the INTO-CPS technologies. They are grouped into broad categories and include both existing, embedded systems activities and activities enabled by INTO-CPS, since INTO-CPS extends traditional embedded systems design capabilities towards CPS design. The choice of granularity for defining these activities naturally affects the size of such a list. The level chosen is instructive for describing workflows, but one that does not make the described workflows overly long.

In the following descriptions (and corresponding summary in Table 1), we identify the tools that support the activities, where applicable, using the following icons:

- The INTO-CPS Application, COE and its extensions.
- Modelio.
- The Overture tool.
- RT-Tester.
- OM OpenModelica.
- 20-sim.

Descriptions of these tools can be found in Section 6 and in appendix B they are explained in more detail. Those activities in *italics* can be recorded by the traceability features of INTO-CPS, which are described in Section 5.5.1.

Requirements and Traceability Writing *Design Notes* ( ) includes documentation about what has been done during a design, why a decision was made and so on. *Requirements* () includes requirements gathering and analysis. *Validation* () is any form of validation of a design or implementation against its required behaviour.

Architectural Modelling INTO-CPS primarily supports architectural modelling in SysML. *Holistic Architectural Modelling* () and *Design Architectural Modelling* () are described in Section 5.5.3. The former focuses on a domain-specific view, whereas the latter targets multi-modelling using a special SysML profile. The *Export Model Descriptions* () activity indicated passing component descriptions from the Design Architectural Model to other modelling tools.

Modelling The *Import Model Description* (OM) activity means taking a component interface description from the Design Architectural Model into another modelling tool. *Cyber Modelling* () means capturing a "cyber" component of the system, e.g. using a formalism/tool such as VDM/Overture. *Physical Modelling* (OM) means capturing the "physical" component of the system, e.g. in 20-sim or OpenModelica. Collectively, these can be referred to as *Simulation Modelling* (OM) to distinguish from other forms, such as *Architectural Modelling* (). *Co-modelling* () means producing a system model with one DE and one CT part, e.g. in Crescendo. *Multi-modelling* () means producing a system model with multiple DE or CT parts with several tools.

Design *Supervisory Control Design* means designing some control logic that deals with high-level such as modal behaviour or error detection and recovery. *Low Level Control Design* means designing control loops that control physical processes, e.g. PID control. *Software Design* is the activity of designing any form of software (whether or not modelling is used). *Hardware Design* means designing physical components (whether or not modelling is used).

Analysis In INTO-CPS, the RT-Tester tool enables the activities of *Model Checking* (), *Creating Tests* () and creating a *Test Oracle* () FMU. The *Create a Configuration* () activity means preparing a multi-model for co-simulation. The *Define Design Space Exploration Configurations* () activity means preparing a multi-model for multiple simulations. *Export FMU* (OM) means to generate an FMU from a model of a component. *Co-simulation* () means simulating a co-model, e.g. using Crescendo baseline technology or the INTO-CPS COE called Maestro.

Prototyping *Manual Code Writing* means creating code for some cyber component by hand. *Generate Code* (OM) means to automatically create code from a model of a cyber component. *Hardware-in-the-Loop (HiL) Simulation* () and *Software-in-the-Loop (HiL) Simulation* () mean simulating a multi-model with one or more of the models replaced by real code or hardware.

5.4 Configuring Multi-Models

As discussed in Section 5.2, a multi-model is a collection of FMUs with a configuration file that: defines instances of those FMUs, specifies connections between the inputs/outputs of the FMU instances, defines values for design parameters of the FMU instances, and defines other simulation settings such as a start, end time, and Master algorithm settings. As seen above, creating a multi-model is a key part of using the INTO-CPS tool chain as it is a pre-requisite for many of the analysis techniques that INTO-CPS can perform.

The INTO-CPS Application supports a project, a view of a folder containing source models, generated FMUs, and configuration files for co-simulation (multi-models) as well as configuration files for other analyses (design space exploration, model checking, test automation). Multi-model configurations can be created in three ways:

1. Created manually using the GUI of the INTO-CPS Application; or
2. Generated from a SysML model created in Modelio; or
3. Created manually by editing JSON configuration files.

All three approaches produce the same configuration file, so the choice of which to use depends on the engineer's background. Those comfortable with SysML may find it best to follow the SysML route, but this is not required. So those unfamiliar with SysML can use the Application directly.

5.5 An Overview of Advanced Methods

We recommend that new users follow the first tutorial[7] to experience the INTO-CPS Application, and then import one or two examples from the Examples Compendium [FPG+18] and interact with them. The more advanced

[7]Updated tutorials supporting newer versions of the toolchain can be found at `https://github.com/INTO-CPS-Association/training/releases`.

Table 1: Activities in existing embedded systems design workflows or enhanced INTO-CPS workflows.

Requirements Engineering	
Stakeholder Documents	
Requirement Definition	
Validation	
Architectural Modelling	
Holistic Architectural Modelling	
Design Architectural Modelling	
Export Model Descriptions	
Modelling	
Import a Model Description	OM
Physical Modelling (Simulation Modelling)	OM
Cyber Modelling (Simulation Modelling)	
Co-modelling	
Multi-modelling	
Design	
Supervisory Controller Design	
Low Level Controller Design	
Software Design	
Hardware Design	
Analysis	
Create Tests	
Model Checking	
Create Test Oracle	
Create a Configuration	
Define Design Space Exploration Configurations	
Export FMU	OM
Co-simulation	
Prototyping	
Generate Code	OM
Hardware-in-the-Loop (HiL) Simulation	
Software-in-the-Loop (SiL) Simulation	
Manual Code Writing	

modelling and analysis methods outlined in this section are described in greater depth in the INTO-CPS Methods Guidelines [PFG+18].

5.5.1 Traceability

The technologies in the INTO-CPS tool chain are able to capture traceability information automatically as activities are performed using the various elements of the tool chain. This includes information about who created or modified an artefact (model, simulation result etc.), and which requirements it is linked to. The traceability features of the INTO-CPS tool are described in depth in [PFG+18] and [LBL+18].

The INTO-CPS tool chain builds a graph of traceability relations, as there can be multiple relationships between different artefacts. The graph is however tree-like in the sense that there must be some root node(s) to trace from or back to. These root nodes are *requirements*. The traceability graph is initialised by using Modelio from the beginning of the development process. The traceability graph is then subsequently updated by the baseline tools as models are created from the model descriptions, FMUs are exported and so on, and co-simulation runs and results will be recorded by the INTO-CPS Application. By performing the required manual input of requirements and links to SysML elements, it is then possible to automatically trace forward to models, FMUs and simulation results, and to trace backwards from these artefacts to individual requirements.

Once a graph has been built, queries can be executed over it to analyse forward traceability (from requirements to entities); backwards traceability (from FMU to requirements); finding sources and sinks for a simulation; assessing coverage such as finding requirements without positive simulation results; and evaluating user impact, such as finding all artefacts influenced by a given user.

5.5.2 Requirements Engineering

Requirements placed on a CPS may, for example, impose restrictions, define system capabilities or identify qualities of a system. In order to use machine-assisted traceability support, requirements need to be recorded explicitly. In the INTO-CPS Methods Guidelines [PFG+18], we describe one possible approach to Requirements Engineering (RE) for CPS, adapting an approach already piloted on systems-of-systems (SoSs). This approach is not mandatory, and in general RE processes and tools vary widely across organisations and domains. In the approach, we propose a collection of views that could be represented as diagrams in SysML, or could equally be represented in other tools where these are already used (e.g., Excel). Examples of views include a *Source Element View*, which defines a collection of source materials from which requirements are derived, a *Requirement Description View* which is used to define the requirements of a system and forms the core of the requirement definition, and the *Context Definition View* which is used to identify interested stakeholders and points of context in the system development, including customers, suppliers and system engineers themselves. The full RE process is a disciplined approach involving identifying and recording source elements, system-level functional and non-functional requirements, an initial system structure which identified cyber and physical elements), and relevant contexts in context definition views. Requirements may then be traced using INTO-CPS tool chain models and results.

5.5.3 SysML and Multi-modelling

Standard SysML can be used as part of a development process to build a model of a system and link elements to requirements. The INTO-CPS tool chain also provides an extended SysML profile that helps users to configure multi-models for co-simulation and configure DSE. The multi-modelling SysML profile defines two diagrams for configuring a co-simulation. The INTO-CPS Application can run a co-simulation based on a configuration file which describes the FMUs, their parameters and connections between them. The design space exploration (DSE) SysML profile is an addition to the multi-modelling SysML profile described above. As with single co-simulation, the INTO-CPS Application can run a DSE based on a configuration file. Alternatively, a configuration can be generated by Modelio, from a set of diagrams defined in the profile. Five diagram types are used to define the objectives for a DSE and their instantiations, define the parameters that will be changed on each co-simulation in a DSE and their instantiations, and the objectives to be used for ranking competing designs.

Holistic and Design Architectural Modelling A system architecture defines the major components of a system, and their relationships, behaviour and interactions. A model of the architecture is potentially partial (representing some or all of the system) and abstract, limited to those elements pertinent to the modelling goal. In CPS engineering, this goal may include understanding the system in terms of the application domain (a *holistic* model), or capturing the system components in a way that targets multi-modelling (a *design* model).

The diagrams in the two profiles described above divide architectural models into subsystems composed of cyber or physical components. Defining an architecture this way may not be the best approach when designing a system ab initio, with systems comprising entities across different domains requiring diverse domain expertise. The Methods Guidelines [PFG+18] discuss and exemplify both holistic and design architectural modelling approaches, and provides some commentary and guidance on how to model in a way which is natural for domain experts, and how to move from holistic to design models when multi-modelling.

Representing Non-Design Elements in SysML Using the INTO-CPS tool chain, we generate co-simulation configurations using an architectural model defined with the INTO-SysML profile. This model defines the structure of a system in terms of the composition of its components and their connections. There are however circumstances where elements in the multi-model are not part of the design of the final system, for example where an FMU is used purely for visualisation. This FMU must be connected to the system components, however is not itself a system component. This is also true when considering the environment of the system. The Methods Guidelines [PFG+18] present an example of the use of these extensions.

5.5.4 A 'DE-first' Approach to Developing Multi-models

After carrying out requirements engineering (RE), and design architectural modelling in SysML the engineering team should have Architecture Structure Diagrams (ASDs) defining the composition of components to be realised as FMUs in cyber or physical formalisms, along with model descriptions exported for each component, and some Connections Diagrams (CDs) that will be used ot configure a multi-model. The next step is to generate a multi-model configuration in the INTO-CPS Application and populate it with FMUs, then run a first co-simulation. This however requires the source models for each FMU to be ready. If they already exist this is easy, however they may not exist if this is a new design. In order to generate these models, the model descriptions for each component can be passed to relevant engineering teams to build the models, then FMUs can be passed back to be integrated.

It can be useful however to create and test simple, abstract FMUs first (or in parallel), then replace these with higher-fidelity FMUs as the models become available. This allows the composition of the multi-model to be checked early, and these simple FMUs can be reused for regression testing. This approach also mitigates the problem of modelling teams working at different rates. Where these simple FMUs are built within the DE formalism (such as VDM), this is called a *DE-first* approach. This approach is particularly appropriate where complex DE control behaviours —such as supervisory control or modal behaviours— are identified as a priority or where the experience of the modelling team is primarily in the DE domain [FLV14].

5.5.5 Modelling Networks with VDM in Multi-models

When modelling and designing distributed controllers, it is necessary to model communications between controllers as well. While controller FMUs can be connected directly to each other through for co-simulation, this quickly becomes unwieldy due to the number of connections increasing exponentially. We suggest employing an 'ether' pattern in which a representation of an abstract communications medium is introduced [FLV14]. In the INTO-CPS setting, the ether is an FMU that is connected to each controller that handles message-passing between them. This reduces the number of connections needed, particularly for large numbers of controllers such as swarms. In the Methods Guidelines [PFG+18], we describe how to pass messages between VDM FMUs using string types, how the ether class works, some of the consequences of using the ether pattern, and finally some extensions for providing quality of service (QoS) guarantees.

5.5.6 Design Space Exploration

Our guidelines for DSE over multi-models of CPSs are intended to (a) support decision management by helping engineers to articulate clearly the parameters, objectives and metrics of a DSE analysis; and (b) enable the tuning of DSE methods for given domains and systems of interest. The Methods Guidelines [PFG+18] describe a SysML profile for systematically describing DSE experiments by defining *parameters*, *objectives* and *rankings*.

DSE is performed in the DSE tool (see the INTO-CPS User Manual [LBL+18]) by processing the DSE configuration using scripts that contain the required algorithms. The main scripts contain the search algorithm that determines which parameters to use in each simulation, the simplest of these is the exhaustive algorithm that methodically runs through all combinations of parameters and runs a simulation of each. The log files produced by each simulation are then processed by other scripts to obtain the objective values defined in the previous section. Finally, the objective values are used by a ranking script to place all the simulation results into a partial order according to the defined ranking. The ranking information is used to produce tabular and graphical results that may be used to support decisions regarding design choices and directions.

An Approach to Effective DSE Given a "designed" design space using the method detailed above, we use the INTO-CPS Tool Chain to simulate each design alternative. Whilst this approach is acceptable on small-scale studies, it quickly becomes infeasible as the design space grows. Inspired by processes found in nature, genetic algorithms "breed" new generations of optimal CPS designs from the previous generation's best candidates. This mimics the concept of survival of the fittest in Darwinian evolution. The approach is detailed in the Methods Guidelines [PFG+18]. An alternative to the genetic search, which is automated, is to use repeated exhaustive searches to home in on better regions of the design space. In this approach the user would plan to perform multiple DSE experiments, each using some portion of their total simulation budget. The first DSE experiment is used to cover the whole range of the design space, but not including all values for each parameter. In this way the first DSE is used to locate regions of interest within the design space. The regions of interest are areas of the design space that produced the better designs according to the ranking results, with the bounds of the 'area' defined by the parameter values that produced good results. The user then divides up their remaining simulation budget between the one or more areas of interest and perform further DSE on those areas.

6 The INTO-CPS Tool Chain

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 [BLL+17] and to Appendix B). The main concepts of the tool-chain are discussed above in Section 5.2.

An overview of the different tools that form the tool-chain of the INTO-CPS association, is given below in Figure 4, where the red boxes indicate the different sections of this chapter.

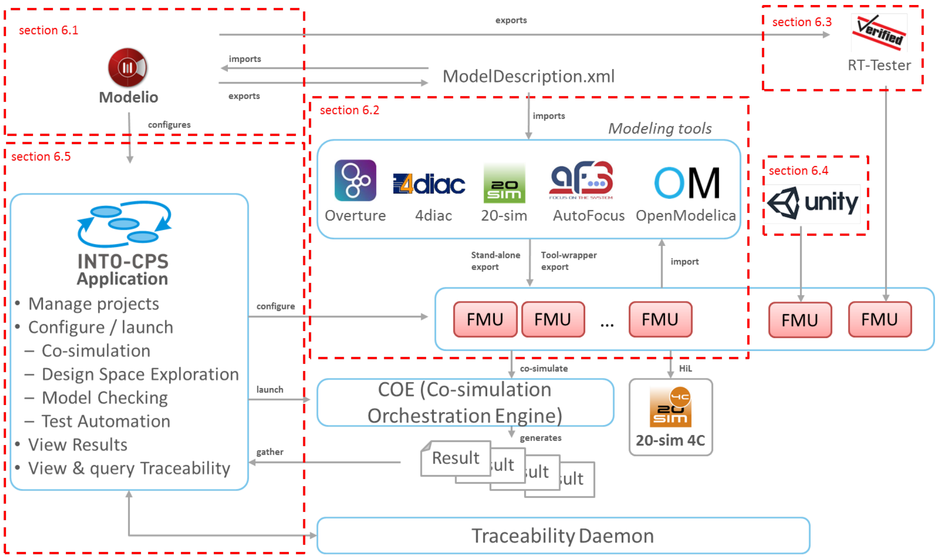


Figure 4: Overview of the different tools and their arrangement in a tool-chain.

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. [PFG+18]), 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[8]. 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. More details about Modelio can be found in Section B.1.

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 un-

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

derstand 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 summarised in this section.

20-sim is a commercial tool for modelling and simulation of mechatronic systems. Together with the related software, 20-sim 4C, Hardware-in-the-Loop simulations can be performed (`http://www.20sim.com/`). This is described further in Section B.3.

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/`). This is described further in Section B.4.

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/`). This is described further in Section B.2.

4Diac is an open-source tool for distributed process measurement and control systems based on the IEC 61499 standard[9]. This is described further in Section **??**.

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

ABS is a language for Abstract Behavioural Specification, which combines implementation-level specifications with verifiability, high-level design with executability, and formal semantics with practical usability. ABS is a concurrent, object-oriented, modelling 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,

[9]See `https://www.eclipse.org/4diac/`.

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 summarises 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	yes	no	no
AutoFocus 3	no	under development	no	no	no
ABS	no	no	no	planned	no

Table 2: 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. This is described further in Section B.5.

6.3.1 Test Automation

Test Automation within INTO-CPS uses the RT Tester tools to generate, perform and analyse 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. The source code for this is available for the members of the INTO-CPS Association in a special SVN.

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[10], or by pre-configured queries.

[10] https://neo4j.com/

7 The INTO-CPS Industrial Case Studies

The INTO-CPS technologies have been applied in a series of industry-led case studies to date. They have each demonstrated different benefits, as might be expected given the different character of the multidisciplinary engineering activities undertaken in each company. Nonetheless, certain aspects of the tool chain have been beneficial in all cases. 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. 5). 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 has been valuable in the majority of the industrial case studies that have been conducted so far. It has provided a means of delivering common documentation of system structure (particularly valuable for larger teams). It has also enhanced communication in each case study project and across the disciplines and tools used. 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. DSE has proved valuable as a way to sweep parameters and automate co-simulations, leading to faster prototyping (and lessening the need for physical prototypes) and a better understanding of the complex interactions between the system parameters.

To commercial entities, the 3D visualisation element has proved valuable 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 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

Figure 5: INTO-CPS Industrial Case studies

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 section presents a case study carried out by TWT Innovation from Germany 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. 6).

A part of the system is transferred seamlessly from a simulation model to 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 [FGL+15, LFW+16a, LTL+16, LFW+16b, LFW+17], one aim was to

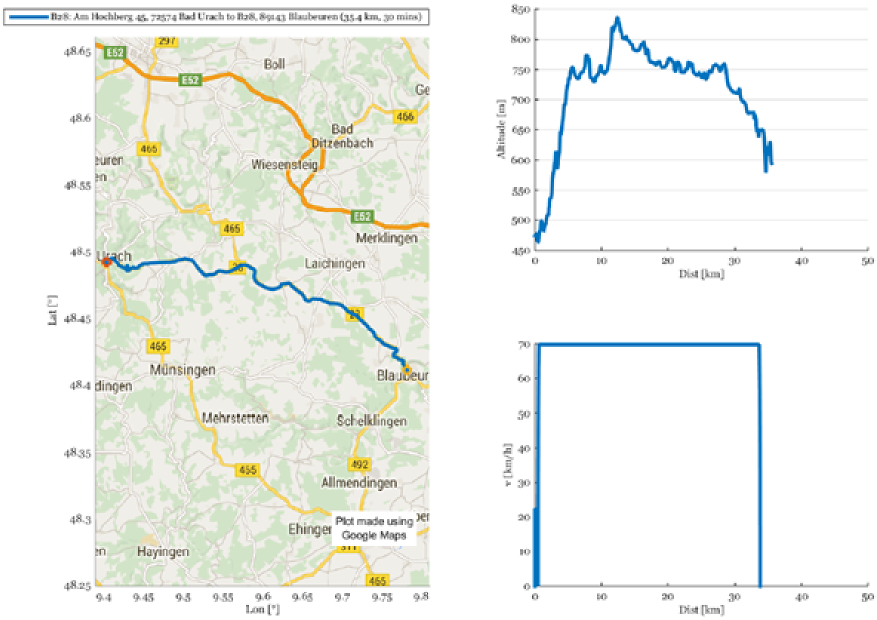


Figure 6: Automotive case study using INTO-CPS: Velocity and altitude profile for a route of 35 km in the vicinity of Stuttgart. The route consists here of a country road only, and thus the velocity is stable at 70 km/h, while the altitude varies between 450m and 850m above sea level.

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 [BBQS15]. The models themselves were created using Matlab[11], 20-sim[12] [Kle06], C++ and Overture[13] [LBF+10].

7.2 The Agricultural Case Study

The focus of the case study is the development of an agricultural field robot called Robotti developed by the Danish company called AgroIntelli using the INTO-CPS technology. Models of the machine dynamics and controllers have

[11] https://www.mathworks.com/products/matlab.html
[12] http://www.20sim.com/
[13] http://overturetool.org/

been developed using the baseline tools and co-simulated using the COE. DSE has been applied in the development and assessment of the steering controller which is crucial to the robot's performance. 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. 7 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. Additionally, the 3D FMU feature has been applied to visualise the machine based on the models of the kinematics and dynamics. Related Publications can be found at [FLG17, FBG+18].

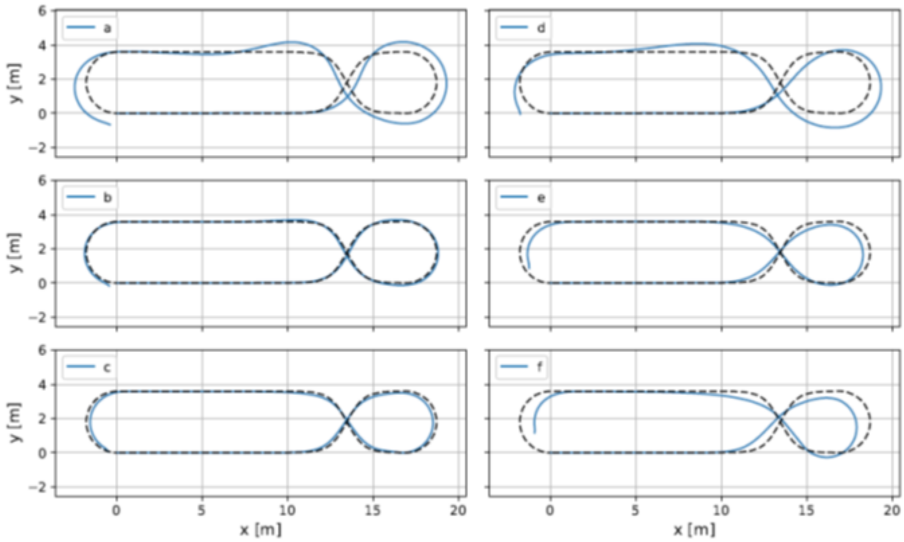


Figure 7: Simulated trajectories of the six controller configurations (a,b,...,f)

7.3 The Building HVAC Case Study

A case study led by United Technologies (their research centre in Ireland) has focused 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. 8 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

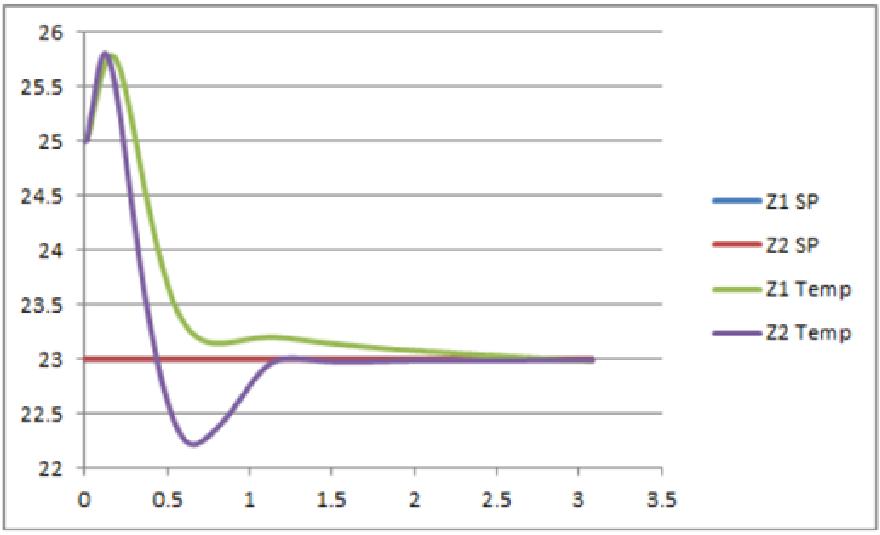
Handling Unit (AHU) and control; f) chiller load and control; g) physical rooms and air flow; h) water and air pipe connections.

The functionality of the HVAC system 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.

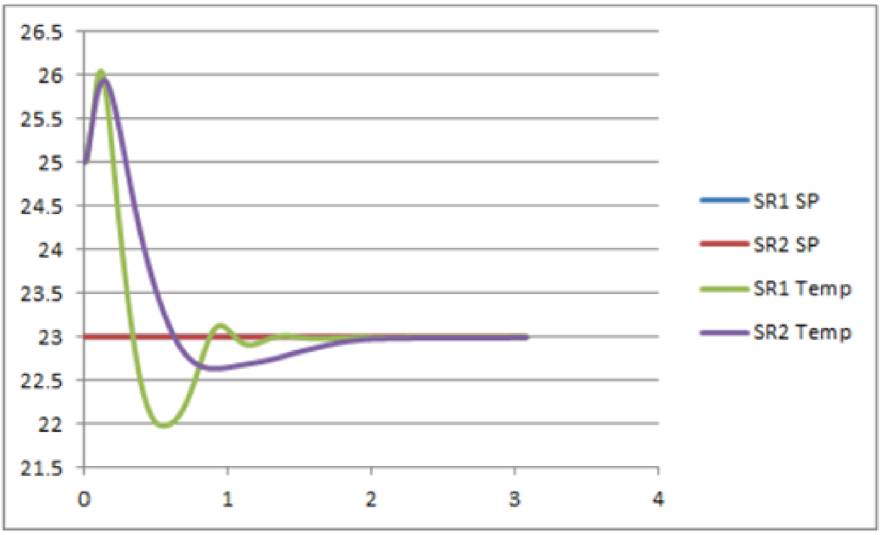
Modelling and simulation was heavily used to evaluate performance and robustness of the system with regards to requirements. Aspects such as simulation speed, and model fidelity were of high importance. Going beyond model-based design and single model simulation, co-simulation enabled the analysis of physical interactions of systems that were previously not captured, due to the different domains at which the physics were modelled. Multi-physics analysis enabled 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 HVAC's 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 [FGP+16, CBM+17].

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 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



(a) Zone Temperature



(b) Single Rooms Temperature

Figure 8: INTO-CPS Co-Simulation Results for Room temperature in building zones

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 led by ClearSy from France, 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.

Related Publications can be found at [LFWL16, FPG+18, HFB+ms].

7.5 The Aerospace Case Study

INTO-CPS and its co-simulation features have been also exploited in Aerospace use cases. As a member of INTO-CPS association, UTRC Ireland have used INTO-CPS for its Irish Development Agency (IDA) internal project[14] and its ongoing Clean Sky2 MISSION project [BMF+17].

The case study involved the co-simulation of a CPS motor actuation. Control modules have been designed and developed using MATLAB/Simulink. Through the INTO-CPS 3D co-simulation feature, we have successfully demonstrated the virtual co-simulation of the control through the Unity Environment (Fig. 9).

The INTO-CPS technology already contains a suite of modelling and simulation tools that are being used in aerospace model-based system engineering applications. In addition to that, tools are FMI enabled and including 20-sim, Overture/VDM and the tool suite has been demonstrated to also work with Dymola, Matlab/Simulink/Modelon, OpenModelica, 4DIAC and SimulationX. This technology is already centered around FMI-based co-simulation and it has initial DSE capabilities that would automate and accelerate aerospace sensitivity analysis in model based development and analysis phases. In the same manner, initial support for test automation using RT Tester with static exploration of the space to be covered is also available, promoting an interesting choice for achieving MC/DC coverage need by aerospace regulations. In future projects the INTO-CPS association

[14] "Network of Excellence in Aerospace Cyber Physical Systems", funded by IDA Agency, 2015

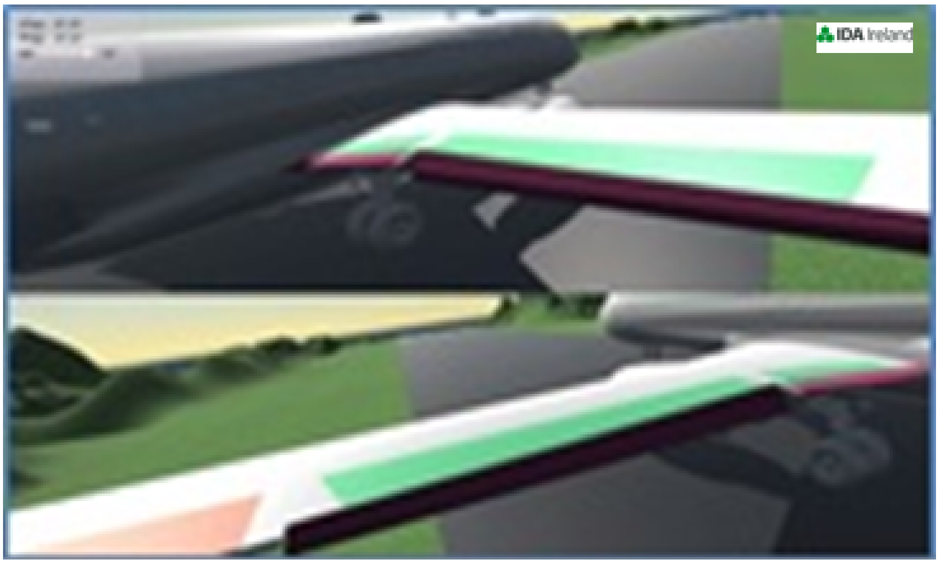


Figure 9: INTO-CPS 3D Co-Simulation Results for high lift actuation

members aim to mature INTO-CPS technology particularly for the aerospace industry and evolve it to TRL 6 and above.

7.6 The Manufacturing Case Study

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 to develop the co-simulation closely followed the classical stages of agent-oriented/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 components 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 + a 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 control of actuators
General	Unity	20-sim animation	PC

Table 3: 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 DE 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 3). 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.

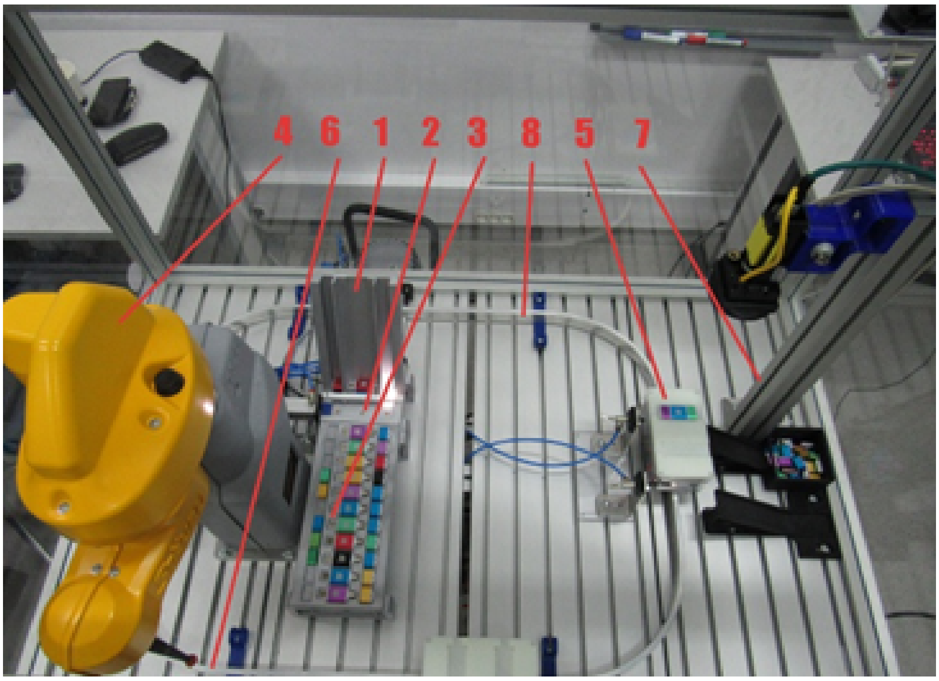


Figure 10: Demo stand for deployment of the co-simulated units, containing: 1) the warehouse stacks; 2) the assembly box at the base of the warehouse stacks; 3) the memory boxes of the warehouse unit; 4) the robotic arm for moving parts around the warehouse; 5) wagons on different locations of the track; 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 advancements 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[15]

7.7 The Combustion Engine Case Study

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 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. It was

[15]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.

illustrated how a transition from the current simulation process at MDT to one using co-simulation utilising FMI.

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 The Mars Rover Case Study

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 DE model expressed using the VDM [FLV08] and Overture [LBF+10] and one 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 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 multi-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 of managing 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

Extensive work has been carried out to identify the main concepts and essential challenges in co-simulation. In this section, we review some of these works.

[TWH07] reviews principles and implementation strategies of co-simulation applied to an HVAC system. It provides multiple experiments showing how the stability and accuracy of the co-simulation is affected by the choice of those strategies.

The work in [HP17] exposes the disparity in terminology related to co-simulation (e.g., the term "co-simulation" is understood as "cooperative simulation", or "coupled system simulation"), provides an in-depth discussion of the multiple concepts, and proposes a way to classify and structure co-simulation methods. The authors propose the distinction by:

state of development: the motivation being the use of co-simulation (e.g., optimise the simulation of a single model by partitioning, or couple the behaviour of wildly different subsystems);

application field: see, e.g., the application fields described in [GTB+17b];

model description: the kind of models being combined (e.g., Ordinary Differential Equations (ODEs), Differential-Algebraic Equations (DAEs), discrete event systems);

numerical approach: the kind of coupling algorithms employed; and

interfaces: the nature of the physical interfaces between the systems being coupled.

Recognising that co-simulation is not a new concept and that it has been applied in wildly different fields, [GTB+18] reviewed co-simulation approaches, research challenges, and research opportunities. They apply feature oriented domain analysis [KCH+90] to help map the field. The main result is a feature model that classifies the requirements of co-simulation frameworks and the participating simulators. They conclude that the main research needs are: finding generic approaches for modular, stable, valid, and accurate coupling of simulation units; and finding standard interfaces for hybrid co-simulation.

With a focus on power systems, but still covering the fundamental concepts, [PVDML+17] highlights the value of co-simulation for the analysis of large scale power systems. In a tutorial fashion, it goes over the main concepts and challenges, providing a great introduction for new researchers in the field.

Recognising the research in co-simulation should be driven by both industry and academia, [SES+18] reports on an empirical survey, given to both practitioners and academics. The preliminary results corroborate the challenges pointed out in the surveys we referenced here. Additionally, it becomes clear that co-simulation is being used without in-depth knowledge of the subject, which may lead to the improper use of the technique, as well as highlighting the need to develop more usable tools.

Finally, [GTD+18] discusses the past and future of co-simulation, providing an historical overview of the topic, as well as possible research directions.

9 Future Directions

It is envisaged that the INTO-CPS technology will be further extended as the FMI standard evolves, 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 [GTB+18]. 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.

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+18] 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 Systems 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

Human-in-the-Loop where people can give input to co-simulations while they are running is certainly relevant. This have actually been carried out both using the PVSio technology on the line following robot example [PBM17]. In addition, human intervention has been used in a part of co-simulations in the IPP4CPPS project (see Section 7.6). However, we imagine that substantial more automation and assistance can be made for this kind of human interaction. One can also imagine that debugging features such as pausing, inspecting values and possibly injecting faults could be included in the Co-simulation Orchestration Engine.

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

CPSs also ideally are smart in the sense that they process intelligent behaviour. In order to introduce autonomy it can be ideal to introduce Machine Learning (ML) in different fashions in order to learn how to best behave. ML can be used in different ways here:

- Based on time series data for an individual physical component one can imagine that ML can be used to automatically derive an approximation FMU corresponding to that physical component.
- In a digital twin setting one can imagine that ML can be used to learn to what extend the real system behave as predicted by the co-models. Based on this different actions could be taken automatically or manually if human intervention is desirable.
- FMUs themselves could contain ML elements for example in order to have adaptive control that will be valuable and express intelligent behaviour.

9.10 Tradeoff in Abstraction between Speed and Accuracy

Many models can be expressed at different levels of abstraction. There is a natural tradeoff between speed and accuracy of a co-simulation. An interesting direction could for example be to support DSE's, while using multiple abstraction models. For example, higher abstraction models might be used to optimise the design globally, while lower abstraction models validate the candidate found, and further optimise each of them to find the best one. In particular for FMUs that are very slow in simulating (e.g., CFD models) it may make a lot of sense to use more abstract models (at least initially) in order to be able to obtain results within a reasonable amount of time. For CFD models it is for example possible to produce ROMs and these can be automatically derived from more detailed models. Thus, it can be valuable to make it easy for users to select what level of abstraction to use in a co-simulation for each of the constituent models.

References

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

[ACM+16] Nuno Amálio, 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 Amálio, 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.

[BB15] A. Beg and A. Butterfield. Development of a prototype translator from Circus to CSPm. In *Proc. 9th Intl. Conf. on Open Source Systems and Technologies (ICOSST)*, pages 16–23. IEEE, December 2015.

[BBG+13a] David Broman, Christopher Brooks, Lev Greenberg, Edward A. Lee, Michael Masin, Stavros Tripakis, and Michael Wetter. Determinate composition of FMUs for co-simulation. In *Eleventh ACM International Conference on Embedded Software*, page Article No. 2, Montreal, Quebec, Canada, 2013. IEEE Press Piscataway, NJ, USA.

[BBG+13b] 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.

[BBQS15] A. Bagnato, E. Brosse, I. R. Quadri, and A. Sadovykh. INTO-CPS: An integrated "tool chain" for comprehensive model-based design of cyber-physical systems. In *Revue Génie Logiciel*, pages 31–35, June 2015.

[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.

[Beg16] M. M. A. Beg. *Translating from "State-Rich" to "State-Poor" Process Algebras*. PhD thesis, Department of Computer Science, Trinity College Dublin, April 2016.

[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.

[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.

[BMF+17] Gilberto Burgio, Leonardo Mangeruca, Alberto Ferrari, Marco Carloni, Virgilio Valdivia-Guerrero, Laura Albiol-Tendillo, Parithi Govindaraju, Marcel Gottschall, Olaf Oelsner, Sören

Reglitz, Jann-Eve Stavesand, Andreas Himmler, and Lionel Yapi. Framework for Modelling and Simulation of Multi-Physics Aircraft Systems with Distributed Electronic Controllers. In *SAE Technical Paper*. SAE International, 2017.

[BOA+11] Torsten Blochwitz, Martin Otter, Martin Arnold, C. Bausch, Christoph Clauss, Hilding Elmqvist, Andreas Junghanns, Jakob Mauss, M. Monteiro, T. Neidhold, Dietmar Neumerkel, Hans Olsson, J.-V. Peetz, and S. Wolf. The Functional Mockup Interface for Tool independent Exchange of Simulation Models. In *8th International Modelica Conference*, pages 105–114, Dresden, Germany, June 2011. Linköping University Electronic Press; Linköpings universitet.

[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.

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

[CBM+17] Luis Diogo Couto, Stylianos Basagianis, Alie El-Din Mady, El Hassan Ridouane, Peter Gorm Larsen, and Miran Hasanagic. Injecting Formal Verification in FMI-based Co-Simulation of Cyber-Physical Systems. In *The 1st Workshop on Formal Co-Simulation of Cyber-Physical Systems (CoSim-CPS)*, Trento, Italy, September 2017.

[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.

[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.

[CSW05] Ana Cavalcanti, Augusto Sampaio, and Jim Woodcock. Unifying classes and processes. *Software and System Modeling*, 4(3):277–296, 2005.

[CW06] A. Cavalcanti and J. Woodcock. A tutorial introduction to CSP in unifying theories of programming. In *Refinement Techniques in Software Engineering*, volume 3167 of *LNCS*, pages 220–268. Springer, 2006.

[CW15] Samuel Canham and Jim Woodcock. Three approaches to timed external choice in UTP. In *Unifying Theories of Programming*, volume 8963, pages 1–20. Springer, 2015.

[CWA16a] 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.

[CWA16b] Ana Cavalcanti, Jim Woodcock, and Nuno Amálio. Behavioural models for fmi co-simulations. In A. C. A. Sampaio and F. Wang, editors, *International Colloquium on Theoretical Aspects of Computing*, Lecture Notes in Computer Science. Springer, 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.

[FBG+18] Frederik Foldager, Ole Balling, Carl Gamble, Peter Gorm Larsen, Martin Boel, and Ole Green. Design Space Exploration in the Development of Agricultural Robots. In *AgEng conference*, Wageningen, The Netherlands, July 2018.

[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.

[FGL+15] John Fitzgerald, Carl Gamble, Peter Gorm Larsen, Kenneth Pierce, and Jim Woodcock. Cyber-Physical Systems design: Formal Foundations, Methods and Integrated Tool Chains. In *FormaliSE: FME Workshop on Formal Methods in Software Engineering*, Florence, Italy, May 2015. ICSE 2015.

[FGP+16] John Fitzgerald, Carl Gamble, Richard Payne, Peter Gorm Larsen, Stylianos Basagiannis, and Alie El-Din Mady. Collaborative Model-based Systems Engineering for Cyber-Physical Systems – a Case Study in Building Automation. In *Proc. INCOSE Intl. Symp. on Systems Engineering*, Edinburgh, Scotland, July 2016.

[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.

[FMI14] Functional mock-up interface for model exchange and co-simulation. Technical Report 2.0, FMI development group, 2014.

[FPG+18] John Fitzgerald, Richard Payne, Carl Gamble, Ken Pierce, Martin Mansfield, Simon Foster, Kangfeng Ye, Casper Thule, Rene Nilsson, Kenneth Lausdahl, Florian Lapschies, and Frederik Foldager. The into-cps examples compendium. Technical report, INTO-CPS Association, October 2018.

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

[FZW14] S. Foster, F. Zeyda, and J. Woodcock. Isabelle/UTP: A mechanised theory engineering framework. In David Naumann, editor, *Proc. 5th Intl. Symposium on Unifying Theories of Programming (UTP 2014)*, volume 8963 of *LNCS*, pages 21–41. Springer, 2014.

[FZW16] S. Foster, F. Zeyda, and J. Woodcock. Unifying heterogeneous state-spaces with lenses. In *Proc. 13th Intl. Conf. on Theoretical Aspects of Computing (ICTAC)*, volume 9965 of *LNCS*. Springer, 2016.

[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.

[GRABR14] Thomas Gibson-Robinson, Philip Armstrong, Alexandre Boulgakov, and A.W. Roscoe. FDR3 — A Modern Refinement Checker for CSP. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 8413 of *Lecture Notes in Computer Science*, pages 187–201, 2014.

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

[GTB+17b] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. Co-simulation: State of the art. Technical report, feb 2017.

[GTB+18] 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.

[GTD+18] Cláudio Gomes, Casper Thule, Julien DeAntoni, Peter Gorm Larsen, and Hans Vangheluwe. Co-simulation: The Past, Future, and Open Challenges. page to be published, Limassol, Cyprus, 2018. Springer Verlag.

[HDM10] I. J. Hayes, S. E. Dunne, and L. Meinicke. Unifying theories of programming that distinguish nontermination and abort. In *Mathematics of Program Construction (MPC)*, volume 6120 of *LNCS*, pages 178–194. Springer, 2010.

[HFB+ms] Miran Hasanagić, Tommaso Fabbri, Victor Bandur, Peter W. V. Tran-Jørgensen, Peter Gorm Larsen, and Julien Ouy. Code Generation for Distributed Embedded Systems. January 2018. Submitted for journal: Design Automation for Embedded Systems.

[HH98] Tony Hoare and Jifeng He. *Unifying Theories of Programming*. Prentice-Hall, 1998.

[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.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. International Series in Computer Science. Prentice Hall, 1985.

[HP17] Irene Hafner and Niki Popper. On the terminology and structuring of co-simulation methods. In *Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pages 67–76, New York, New York, USA, 2017. ACM Press.

[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.

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

[INCb] INCOSE Systems of Systems Primer. International Council on Systems Engineering (INCOSE), Technical Publication TP-2018-003-01.0.

[KCH+90] K. C. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-Oriented Domain Analysis. Feasibility study,. Technical report, Carnegie Mellon University, 1990.

[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.

[LBL+18] Peter Gorm Larsen, Victor Bandur, Kenneth Lausdahl, Casper Thule, Carl Gamble, Richard Payne, Adrian Pop, Etienne Brosse, Jörg Brauer, Florian Lapschies, Marcel Groothuis, Tom Bokhove, Christian Kleijn, Luis Diogo Couto, and Christian König. Into-cps tool chain user manual. Technical report, INTO-CPS Association, September 2018.

[LCL13] Kenneth Lausdahl, Joey W. Coleman, and Peter Gorm Larsen. Semantics of the VDM Real-Time Dialect. Technical Report ECE-TR-13, Aarhus University, April 2013.

[LFW+16a] Peter Gorm Larsen, John Fitzgerald, Jim Woodcock, Peter Fritzson, Jörg Brauer, Christian Kleijn, Thierry Lecomte, Markus Pfeil, Ole Green, Stylianos Basagiannis, and Andrey Sadovykh. Integrated Tool Chain for Model-based Design of Cyber-Physical Systems: The INTO-CPS Project. In *CPS Data Workshop*, Vienna, Austria, April 2016.

[LFW+16b] Peter Gorm Larsen, John Fitzgerald, Jim Woodcock, René Nilsson, Carl Gamble, and Simon Foster. Towards Semantically Integrated Models and Tools for Cyber-Physical Systems Design. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation, Proc 7th Intl. Symp.*, volume 9953 of *Lecture Notes in Computer Science*, pages 171–186. Springer International Publishing, 2016.

[LFW+17] Peter Gorm Larsen, John Fitzgerald, Jim Woodcock, Carl Gamble, Richard Payne, and Kenneth Pierce. Features of integrated model-based co-modelling and co-simulation technology. In Bernardeschi, Masci, and Larsen, editors, *1st Workshop on Formal Co-Simulation of Cyber-Physical Systems*, Trento, Italy, September 2017. LNCS, Springer-Verlag.

[LFWL16] Peter Gorm Larsen, John Fitzgerald, Jim Woodcock, and Thierry Lecomte. *Trustworthy Cyber-Physical Systems Engineering*, chapter Chapter 8: Collaborative Modelling and Simulation for Cyber-Physical Systems. Chapman and Hall/CRC, September 2016. ISBN 9781498742450.

[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.

[LP95] Peter Gorm Larsen and Wiesław Pawłowski. The Formal Semantics of ISO VDM-SL. *Computer Standards and Interfaces*, 17(5–6):585–602, September 1995.

[LTL+16] Peter Gorm Larsen, Casper Thule, Kenneth Lausdahl, Victor Bandur, Carl Gamble, Etienne Brosse, Andrey Sadovykh, Alessandra Bagnato, and Luis Diogo Couto. Integrated Tool Chain for Model-Based Design of Cyber-Physical Systems. In Peter Gorm Larsen, Nico Plat, and Nick Battle, editors, *The 14th Overture Workshop: Towards Analytical Tool Chains*, pages 63–78, Cyprus, November 2016. Aarhus University, Department of Engineering. ECE-TR-28.

[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.

[Mod14] Modelica Association. Modelica - A Unified Object-Oriented Language for Systems Modeling - Version 3.3 Revision 1. Standard Specification, July 2014.

[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.

[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.

[NWP02] T. Nipkow, M. Wenzel, and L. C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[OCW07] M. V. M. Oliveira, A. L. C. Cavalcanti, and J. C. P. Woodcock. A UTP Semantics for Circus. *Formal Aspects of Computing*, **21**(1):3 – 32, 2007.

[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.

[OSF14] M. V. M Oliveira, A. C. A. Sampaio, and M. S. C. Filho. Model-checking Circus state-rich specifications. In *11th Intl.*

Conf. on Integrated Formal Methods, volume 8739 of *LNCS*, pages 39–54. Springer, 2014.

[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.

[PBM17] Maurizio Palmieri, Cinzia Bernardeschi, and Paolo Masci. Co-simulation of semi-autonomous systems: the Line Follower Robot case study. In *1st Workshop on Formal Co-Simulation of Cyber-Physical Systems*, Trento, Italy, September 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.

[PFG+18] Ken Pierce, John Fitzgerald, Carl Gamble, Martin Mansfield, Richard Payne, and Peter Gorm Larsen. Into-cps method guidelines. Technical report, INTO-CPS Association, September 2018.

[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/.

[Pla18] André Platzer. *Logical Foundations of Cyber-Physical Systems*. Springer, 2018.

[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.

[Pto14] Claudius Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014.

[PVDML+17] Peter Palensky, Arjen A. Van Der Meer, Claudio David Lopez, Arun Joseph, and Kaikai Pan. Cosimulation of Intelligent Power Systems: Fundamentals, Software Architecture, Numerics, and Coupling. *IEEE Industrial Electronics Magazine*, 11(1):34–50, March 2017.

[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.

[SES+18] Gerald Schweiger, Georg Engel, Josef Schoeggl, Irene Hafner, Cláudio Gomes, and Thierry Nouidui. Co-Simulation – an Empirical Survey: Applications, Recent Developments and Future Challenges. In *MATHMOD 2018 Extended Abstract Volume*, pages 125–126, Vienna, Austria, 2018. ARGESIM Publisher Vienna.

[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.

[TWH07] Marija Trcka, Michael Wetter, and Jan Hensen. Comparison of co-simulation approaches for building and HVAC/R system simulation. In *International IBPSA Conference*, Beijing, China, 2007.

[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.

[WC01] J. C. P. Woodcock and A. L. C. Cavalcanti. A Concurrent Language for Refinement. In A. Butterfield and C. Pahl, editors, *IWFM'01: 5th Irish Workshop in Formal Methods*, BCS Electronic Workshops in Computing, Dublin, Ireland, July 2001.

[WCF+12] J. Woodcock, A. Cavalcanti, J. Fitzgerald, P. Larsen, A. Miyazawa, and S. Perry. Features of CML: a Formal Modelling Language for Systems of Systems. In *Proceedings of the 7th International Conference on System of System Engineering*. IEEE, July 2012.

[WCF+14] Jim Woodcock, Ana Cavalcanti, John Fitzgerald, Simon Foster, and Peter Gorm Larsen. Contracts in cml. In *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, pages 54–73. Springer, 2014.

[WD96] Jim Woodcock and Jim Davies. *Using Z – Specification, Refinement, and Proof*. Prentice Hall International Series in Computer Science, 1996.

[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.

[Woo14] Jim Woodcock. Engineering UToPiA - Formal Semantics for CML. In Cliff Jones, Pekka Pihlajasaari, and Jun Sun, editors, *FM 2014: Formal Methods*, volume 8442 of *Lecture Notes in Computer Science*, pages 22–41. Springer International Publishing, 2014.

[WRF+15] David D Walden, Garry J Roedler, Kevin J Forsberg, R Douglas Hamelin, and Thomas M Shortell, editors. *Systems Engineering Handbook. A Guide for System Life Cycle Processes and Activities, Version 4.0.* Wiley, January 2015.

[WWC13] Kun Wei, Jim Woodcock, and Ana Cavalcanti. Circus Time with Reactive Designs. In *Unifying Theories of Programming*, volume 7681 of *LNCS*, pages 68–87. Springer, 2013.

[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.

[ZFF16] F. Zeyda, S. Foster, and L. Freitas. An axiomatic value model for Isabelle/UTP. In *Proc. 6th Intl. Symp. on Unifying Theories of Programming*, volume 10134 of *LNCS*. Springer, 2016. To appear.

A List of Acronyms

ABS	Abstract Behavioral Specification
AI	Agrointelli
API	Application Programming Interface
ASD	Architecture Structure Diagram
AST	Abstract Syntax Tree
AU	Aarhus University
BCS	Basic Control States
BMC	Bounded Model Checker
CD	Connections Diagram
CFD	Computational Fluid Dynamics
CLP	Controllab Products B.V.
COE	Co-simulation Orchestration Engine
CORBA	Common Object Request Broker Architecture
CPS	Cyber-Physical Systems
CSP	Communication Sequential Processes
CT	Continuous-Time
CTL	Computation Tree Logic
DAE	Differential-Algebraic Equation
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
HOL	Higher Order Logic
HVAC	Heating, Ventilation, and Air Conditioning
HW	Hardware
ICT	Information Communication Technology
IDA	Irish Development Agency
IDE	Integrated Design Environment
LTL	Linear Temporal Logic
M&S	Modelling and Simulation
MA	Master Algorithm

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
NFP	Non-Functional Property
ODE	Ordinary Differential Equation
OMG	Object Management Group
OS	Operating System
OSLC	Open Services for Lifecycle Collaboration
PID	Proportional Integral Derivative
PROV-N	The Provenance Notation
RE	Requirements Engineering
ROM	Reduced Order Model
RPC	Remote Procedure Call
RTT	Real-Time Tester
SiL	Software-in-the Loop
SMT	Satisfiability Modulo Theories
SoS	System of Systems
SOS	Structural Operational Semantics
ST	Softeam
SUT	System Under Test
SVN	Subversion
SysML	Systems Modelling Language
TA	Test Automation
TE	Test Environment
TIM	Traceability Information Model
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 Background on the Individual Tools

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

B.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 customisation 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.

B.2 Overture

The Overture platform [LBF+10] is an Eclipse-based IDE for the development and validation of system specifications in three dialects of the specification language of the Vienna Development Method (VDM). 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.
- Visualisation of test coverage information gathered from automated testing.
- Visualisation 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 synchronisation 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 and it enables the description of distributed systems. Examples can be found in the VDM-RT tutorials distributed with Overture. The integration of Overture into the INTO-CPS tool-chain is realised via the FMI standard.

B.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 analysed 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 analyse models, build control systems and improve system performance. Figure 11 shows 20-sim with a model of a controlled

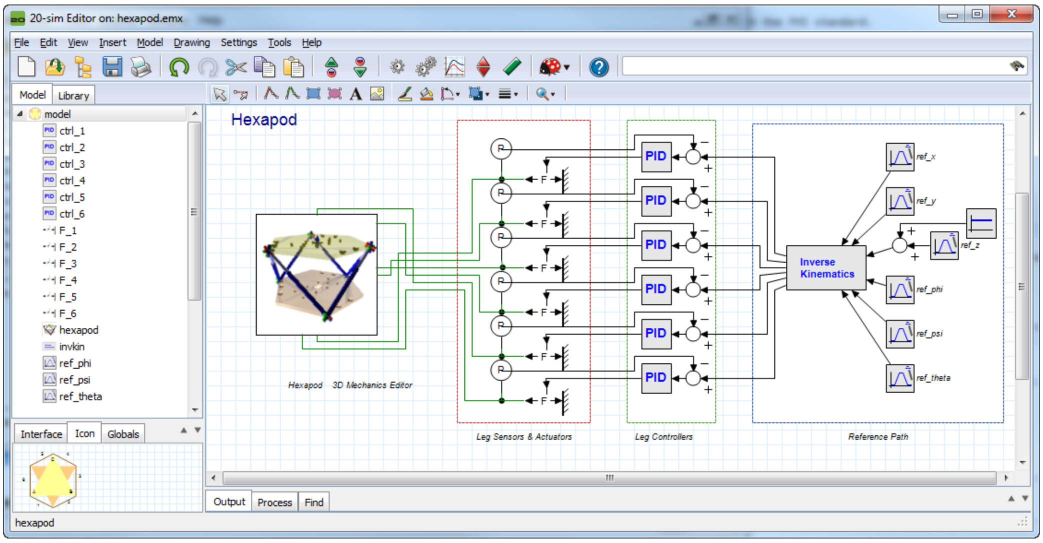


Figure 11: Example of a hexapod model in 20-sim.

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 realised via the FMI standard.

B.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 realised via compliance with the FMI standard, and is described in Deliverable D4.3b [PBL+17].

B.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.

B.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.

B.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 12, 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].

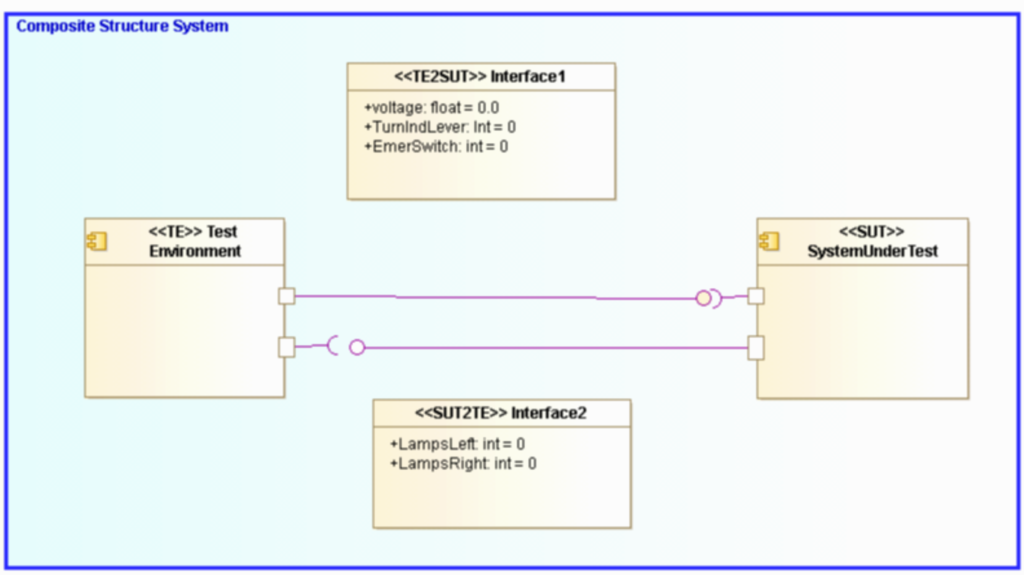


Figure 12: Simple model that highlights interfaces between the environment and the system-under-test.

B.6 Eclipse 4diac™

Eclipse 4diac™ (4diac from now on) provides an open source infrastructure for distributed industrial process measurement and control systems based on the IEC 61499 standard. IEC 61499 defines a domain-specific modeling language for developing distributed industrial control solutions. IEC 61499 extends IEC 61131-1 by improving the encapsulation of software components for increased re-usability, providing a vendor-independent format, and simplifying support for controller-to-controller communication. Its distribution functionality and the inherent support for dynamic reconfiguration provide the required infrastructure for Industrie 4.0 and industrial IoT applications. 4diac allows the development of distributed control systems compliant to the IEC 61499 standard and three of its main projects are:

- 4diac FORTE: The runtime environment is a small portable C++ implementation of an IEC 61499 runtime environment, which supports the execution of distributed control programs on small embedded devices. 4diac FORTE runs above a device's OS. It is a multi-threaded and low memory consuming runtime environment. The runtime environment has been tested on the following systems:
  - Windows Cygwin on i386, ppc and xScale

  - Linux on i386, ppc and xScale
  - NetOS
  - RTOS on IPC@chip
  - eCos ARM7
  - VxWorks
  - freeRTOS

- 4diac IDE: This is the IDE written in Java and based on the Eclipse framework and provides an extensible engineering environment for modeling distributed control applications compliant to the IEC 61499 standard. The user uses 4diac IDE to create FBs, applications, configure the devices and all related to IEC 61499 and also download this to devices running 4diac FORTE.

- Function Block Library: contains Function Blocks which are available in 4diac FORTE and can, therefore, be used to create IEC 61499 compliant control applications.

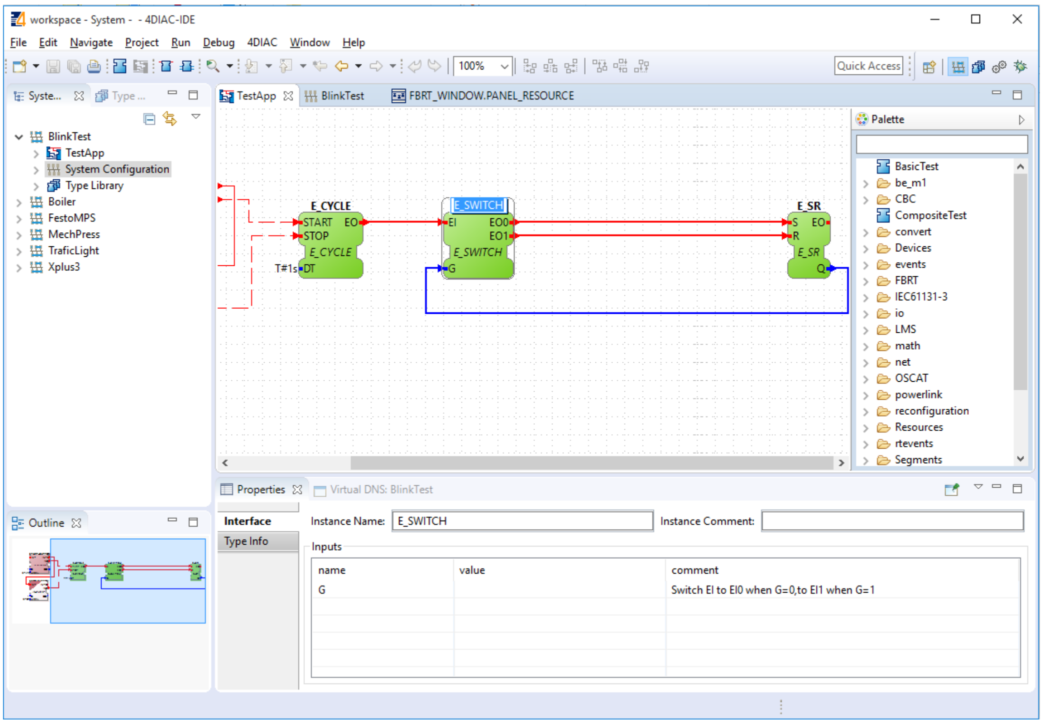


Figure 13: Overview of the 4diac IDE

From version 1.10, 4diac has the capability to export each device of a system as an FMU (FMI 2.0) in order to test the behaviour of the controller against another FMU of the controlled system in the co-simulation environment. More detailed information about 4diac can be found in the official website:

http://www.eclipse.org/4diac/en_help.php