INtegrated TOol chain for model-based design of CPSs

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

Version: 0.8

Date: September, 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.

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+18a] 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 increase the open tool chain with additional tools from its partners. The association provides open source software, training material including tutorials and documents describing the connections and use of the technology. The INTO-CPS Association attempts to ensure that the different tools are kept in sync and that the importable examples expressed in a collection of notations using different tools are kept up to date.

This guide to INTO-CPS starts of with an overview of what we consider the main challenges in the engineering of CPSs in section 2. Afterwards section 3 provides a short overview of the INTO-CPS projects in a nutshell. This is followed by section 4 which gives an overview of the CPS foundations; section 5 which gives and overview of the CPS methodology; and section 7 an overview of the tool chain. Afterwards, the industrial usage of the INTO-CPS technology that have been reported publicly in section 8. Finally, section 9 provide an overview of related work and section 10 presents an overview of potential future directions to take the INTO-CPS technology. At the end of this guide there are three appendices where Appendix A provide a list of acronyms; Appendix B provides a definition of the main terms used in the INTO-CPS documentation and Appendix C provides 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-CPSis simple: that teams of developers from diverse disciplines and organisations are enabled to work together to converge more rapidly than today on system designs that perform optimally. Realising this vision requires methods and tools that 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 support such multidisciplinary design with semantically well-founded approaches in a cost-effective manner? In general we believe that there are a number of areas where there are challenges in the proper engineering of CPSs. These are described below in separate subsections.

2.1 The challenge of time to market

There is a clear need for model-based methods to permit early design space exploration, optimisation and experimentation without delaying the time from initial idea towards the launch of a new product to the market. We feel that the main areas of relevance 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 The challenge of diversity

In the INTO-CPS project we start from the view that disciplines such as software, mechatronic and control engineering have evolved notations and theories that are tailored to their needs, and that it is undesirable to suppress this diversity by enforcing uniform general-purpose models [FGL+15, LFW+16b]. The goal, then, must be to allow the effective federation of highly diverse design models.

2.3 The challenge of collaboration

There is a clear need to provide mechanisms to support collaborative model-base engineering without compromising the independence of contributors. We feel that the main areas of relevance 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.

To the CPS engineer, the system of interest includes both computational and physical elements, so 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 (*co-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. The idea is to enable co-simulation of extensible groups of semantically diverse models, and at the same time the semantic foundations are extended using Unifying Theories of Programming (UTP) to permit analysis using advanced meta-level tools that are primarily

targeted towards academics and thus not considered as a part of the industrial INTO-CPS tool chain.

3 INTO-CPS in a Nutshell

To address the challenges presented above in Section 2, INTO-CPS project has created an integrated "tool chain" for comprehensive model-based design of CPSs. The tool chain supports multidisciplinary, collaborative modelling of CPSs from requirements, through simulation of multiple heterogeneous models that represent the physical elements as well as the computational parts of the system, down to realisation in hardware and software, enabling traceability at all stages of the development as outlined in figure 1.

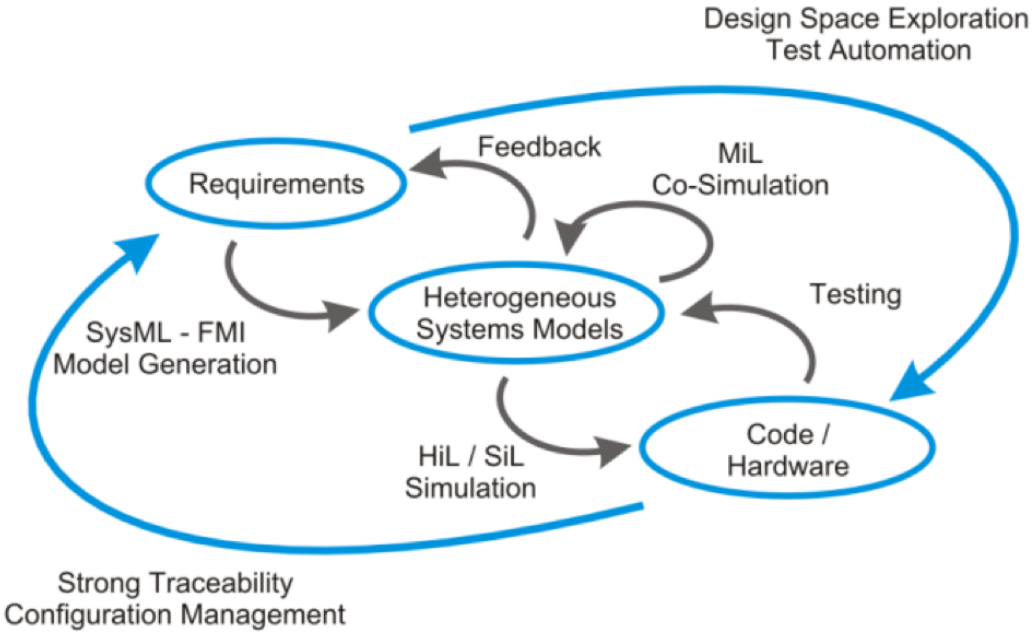


Figure 1: Connections in the INTO-CPS tool chain.

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

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

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

3.1 How INTO-CPS works

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

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

A Co-simulation Orchestration Engine (COE) called Maestro has been built on the baseline technologies and in accordance with requirements driven by the industry case studies outlined below [TLLM18]. This engine combines previous experience from TWT's Co-Simulation engine and the Crescendo tool developed in the project Design Support and Tooling for Embedded

Control Software (DESTECS) [BLV+10]. The goals for the COE include, among others, optimised scalability and performance, and data exchange between the different models facilitated by the Functional Mockup Interface (FMI) [Blo14]. Interfaces to further tools will be provided so that the requirements and the different artefacts will be fully exploited. An INTO-CPS Application acting as a common front-end to the INTO-CPS tool chain has been produced using web-based technologies (on top of Electron). This enables stakeholders without detailed knowledge on the different modelling technologies to experiment with alternative candidate designs and use systematic ways to either explore a large design space or systematically test heterogeneous models. The INTO-CPS Application, the COE and its most important connections are shown in Figure 2.

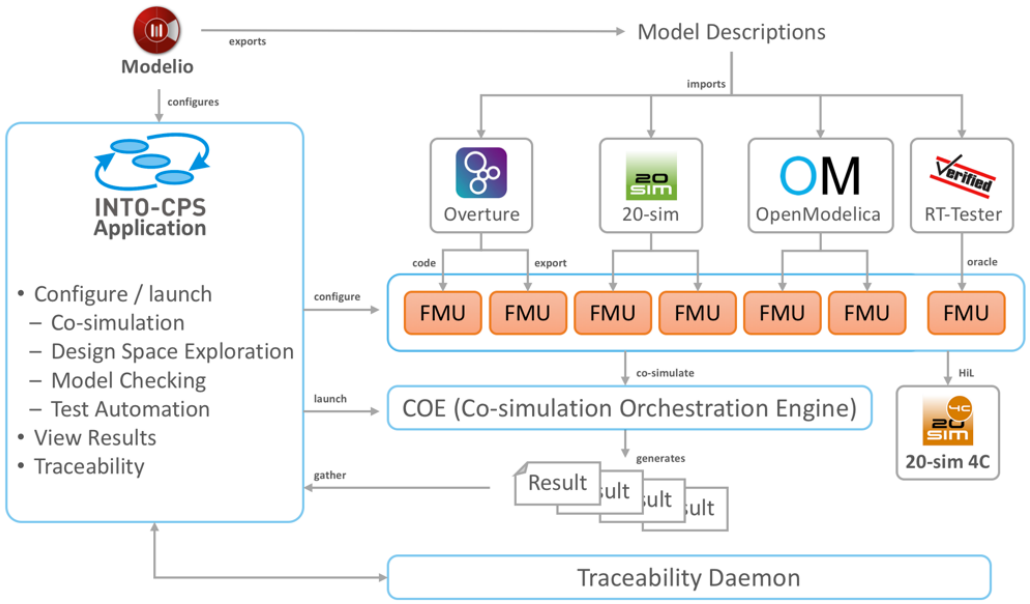


Figure 2: Overview of the INTO-CPS tool chain.

The COE connects multiple diverse models, each in the encapsulated form of a Functional Mock-up Unit (FMU) or running in its native modelling environment, to an overall system model. An algorithm for Design Space Exploration (DSE) enables sweeps through ranges of design parameters, performing co-simulations on each. System robustness can be evaluated by using Test Automation (TA) tools that can manipulate the simulation. Links to models can be kept in a database to allow for versioning and traceability even between artefacts produced by different tools. The INTO-CPS tool chain is described in more detail in Section 7.

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 8. 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 [ZCWO17a], 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+16b] 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+18a], the coupling of subsystem simulations. This constitutes the backdrop of the industrial Functional Mockup Interface (FMI) standard [Int14, BBG+13, CWA16] for co-simulation of components built using distinct modelling tools. The FMI Standard has been proposed to address the challenge of interoperability, coupling different simulators and their high-level control components via a bespoke FMI API.

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

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

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

Our approach is to formally verify the well-formedness and healthiness of SysML CPS architectural designs as a prelude to co-simulation. The designs

are described using INTO-SysML [APC+15], a profile for multi-modelling and FMI co-simulation. The well-formedness checks verify that designs comply with all the required constraints of the INTO-SysML meta-model; this includes connector conformity, which checks the adequacy of the connections between SysML blocks (denoting components) with respect to the types of the ports being wired. The healthiness checks concern detection of algebraic loops, a feedback loop resulting in instantaneous cyclic dependencies; this is relevant because a desirable property of co-simulation, which often reduces to coupling of simulators, is convergence (where numerical analyses approximate the solution), which is dependent on the structure of the subsystems and cannot be guaranteed if this structure contains algebraic loops [KS00, BBG+13]. The work in 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 [Hoa85a] with its accompanying FDR3 refinement-checker [RABR16].

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

4.1.4 Instantiation for robotics applications

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

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

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

It is also relatively direct to wrap existing CSP semantics for UML state machines [DC03, RW05] to allow the use of such models as FMUs in a co-simulation. In this case, traditional UML modelling can be adopted.

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

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

4.1.5 Future work

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

5 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 chapter 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.

The chapter complements the Tool Chain User Manual [BLL+17]— which gives detail on how to use the features of the tool chain— by providing guidance on when and why you might use these features. 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 [MGP+17]. Industrial applications are described in the Case Studies report [OLF+17]. 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, architectural modelling, New users are recommended to:

- Read the introductory material in Sections 5.1–5.3.
- Follow the first tutorial to experience the INTO-CPS Application.
- Import one or two examples from the Examples Compendium[6] into the INTO-CPS Application and interact with them.
- As you start your own multi-modelling, return to this chapter for guidance on particular topics as required.

[6] https://github.com/INTO-CPS-Association/training/tree/master/tutorials

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" [**?**].

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 CPS model, 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.

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*** is they describe computational or physical elements.

We consider both ***holistic*** and a ***design*** architectures (Section 6.3). 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 [APCB15] 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].

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

5.2.3 Tools

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

Central to the INTO-CPS tool chain is the use of the ***Functional Mockup Interface (FMI)*** standard. This is a tool-independent standard to support both model exchange and co-simulation of dynamic models using a combination of XML-files and compiled C-code [Blo14]. Part of the FMI standard for model exchange is specification of a ***model description*** file. This is an XML file that supplies a description of all properties of a model (for example input/output variables). A ***Functional Mockup Unit (FMU)*** is a tool component that implements FMI. Data exchange between FMUs and the synchronisation of all simulation solvers [Blo14] is controlled by a ***Master Algorithm***.

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

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

Code generation is the transformation of a model into generated code suit-

able for compilation into one or more target languages (e.g. C or Java).

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

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

5.2.4 Analysis

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

Test Automation (TA) is defined as the machine assisted automation of system tests. In INTO-CPS, we concentrate on various forms of ***model-based testing*** – centering on testing system models, against the require-

ments on the system. The ***System Under Test (SUT)*** is "the system currently being tested for correct behaviour. An alias for system of interest, from the point of view of the tester" [HIL+14]. The SUT is tested against a collection of ***test cases*** – a finite structure of input and expected output [UPL06], alongside a ***test model***, which specifies the expected behaviour of a system under test [CMC+13]. TA uses a ***test suite*** – a collection of ***test procedures***. These test procedures are detailed instructions for the set-up and execution of a given set of test cases, and instructions for the evaluation of results of executing the test cases [WG-92].

INTO-CPS considers three main types of test automation: ***Hardware-in-the-Loop (HiL)***, ***Software-in-the-Loop (SiL)*** and ***Model-in-the-Loop (MiL)***. In ***HiL*** there is (target) hardware involved, thus the FMU is mainly a wrapper that interacts (timed) with this hardware; it is perceivable that realisation heavily depends on hardware interfaces and timing properties. In ***Software-in-the-Loop (SiL)*** testing the object of the test execution is an FMU that contains a software implementation of (parts of) the system. It can be compiled and run on the same machine that the COE runs on and has no (defined) interaction other than the FMU-interface. Finally, in ***Model-in-the-Loop (MiL)*** the test object of the test execution is a (design) model, represented by one or more FMUs. This is similar to the SiL (if e.g., the SUT is generated from the design model), but MiL can also imply that running the SUT-FMU has a representation on model level; e.g., a playback functionality in the modelling tool could some day be used to visualise a test run.

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

5.2.5 Existing Tools and Languages

The INTO-CPS tool chain uses several existing modelling tools. ***Overture***[7] supports modelling and analysis in the design of discrete, typically, computer-based systems using the ***VDM-RT*** notation. VDM-RT is based upon the ***object-oriented*** paradigm where a model is comprised of one or more ***objects***. An object is an instance of a ***class*** where a class gives a definition of zero or more ***instance variables*** and ***operations*** an object will contain. Instance variables define the identifiers and types of the data stored within an object, while operations define the behaviours of the object.

The ***20-sim***[8] tool can represent continuous time models in a number of ways. The core concept is that of connected ***blocks***. ***Bond graphs*** may implement blocks. Bond graphs offer a domain-independent description of a physical system's dynamics, realised as a directed graph. The vertices of these graphs are idealised descriptions of physical phenomena, with their edges (***bonds***) describing energy exchange between vertices. Blocks may have input and output ***ports*** that allow data to be passed between them. The energy exchanged in 20-sim is the product of ***effort*** and ***flow***, which map to different concepts in different domains, for example voltage and current in the electrical domain.

OpenModelica[9] is an open-source ***Modelica***-based modelling and simulation environment. Modelica is an "object-oriented language for modelling of large, complex, and heterogeneous physical systems" [FE98]. Modelica models are described by ***schematics***, also called ***object diagrams***, which consist of connected components. Components are connected by ports and are defined by sub components or a textual description in the Modelica language.

Modelio[10] is an open-source modelling environment supporting industry standards like UML and SysML. INTO-CPS will make use of Modelio for high-level system architecture modelling using the ***SysML*** language and proposed extensions for CPS modelling. The systems modelling language (SysML) [Sys12] extends a subset of the UML to support modelling of heterogeneous systems.

[7] `http://overturetool.org/`
[8] `http://www.20sim.com/`
[9] `https://www.openmodelica.org/`
[10] `http://www.modelio.org/`

5.2.6 Formalisms

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

A structural operational semantics (SOS) describes how the individual steps of a program are executed on an abstract machine [Plo81]. An SOS definition is akin to an interpreter in that it provides the meaning of the language in terms of relations between beginning and end states. The relations are defined on a per-construct basis. Accompanying the relations are a collection of semantic rules which describe how the end states are achieved. Where an operational semantics defines how a program is executed, a denotational approach defines a language in terms of denotations, in the form of abstract mathematical objects, which represent the semantic function that maps over the inputs and outputs of a program [SS71].

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

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

Several forms of verification are enabled through the use of formally defined languages. ***Refinement*** is a verification and formal development technique pioneered by [BW98] and [Mor90]. It is based on a behaviour preserving relation that allows the transformation of an abstract specification into more and more concrete models, potentially leading to an implementation. ***Proof*** is the process of showing how the validity of one statement is derived from others by applying justified rules of inference [BFL$^+$94]. For the purposes of verification in INTO-CPS, we make use of the Isabelle/HOL theorem prover and the FDR3 refinement checker. These are not considered part of the INTO-CPS tool chain, but were used in the INTO-CPS project primarily to

support the development of foundation work.

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 5.2.5. Those activities in *italics* can be recorded by the traceability features of INTO-CPS, which are described in Section 6.1.

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 6.3. 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 COE.

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 Chapter **??**, 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. These two approaches are covered in the second and third tutorials in Part **??**. Manually editing the JSON configuration is an advanced topic that is not covered in the tutorials, but since JSON is human-readable, not complicated with some experimentation.

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	

5.5 First Steps for Users

In this final section, we consider a how different types of users might approach the INTO-CPS technologies. As described in Section 5.1, all new users are recommended to follow the first tutorial to experience the INTO-CPS Application, import one or two examples from the Examples Compendium (Deliverable D3.6 [MGP+17]) into the INTO-CPS Application and interact with them, returning to this chapter and Chapter 6 as and when you require guidance on a particular area.

After initial familiarisation, the following list provides hints on next steps for different types of users, and where to find further information. As a reminder, updated tutorials supporting newer versions of the tool can be found at `https://github.com/INTO-CPS-Association/training/releases`.

Students Bachelor and Masters students wishing to build multi-models should follow the first few tutorials on adding exporting and adding FMUs. The SysML tutorial can be skipped if desired. Further guidance on exporting FMUs from different tools can be found in the User Manual, Deliverable D4.3a [BLL+17].

Individual Engineers Engineers should follow the first few tutorials on adding and exporting FMUs. The SysML tutorial is also recommended. Further guidance on exporting FMUs from different tools can be found in the User Manual, Deliverable D4.3a [BLL+17]

Engineering Teams Teams requiring traceability must read Chapter **??** first (and Chapter **??** is also recommended), as traceability must be considered from the outset. The SysML tutorial is mandatory, because traceability links begin with requirements and architectural models in Modelio.

Those with Legacy Models A primary goal is to generate an FMU from the tool for your existing models. These can be incorporated into multi-models as described in the second tutorial.

Those wishing to run Design Space Exploration It is necessary to build a multi-model first in order to run a DSE, so the first tutorials should be followed. The SysML tutorial is optional, though useful as the SysML profile includes extensions to help configure DSE analyses. The later tutorials cover DSE, with further guidance in the user manual, Deliverable D4.3a [BLL+17], and Deliverable D5.3e [Gam17] (for more technical details).

Those interested in model checking The User Manual, Deliverable D4.3a [BLL+17], provides useful insight, with in-depth information found in Deliverable D5.3c [BH17].

Those interested in formal semantics and analysis The collection of D2.3deliverables [ZCWO17b, CFT+17, ZFC+17, CFWZ17] provides in-depth information on these aspects of the tool chain, including mechanisation efforts in Isabelle.

6 INTO-CPS Advanced Method Guidelines

Sections 6.1–6.6, covers more advanced topics that require a basic familiarity with the INTO-CPS technologies. Although Sections 6.1–6.6 are ordered based on a start-to-end "work flow", it is not necessary to read them in order. Experienced users may read any section on which they require further guidance.

6.1 Traceability

The technologies in the INTO-CPS tool chain are able to automatically capture traceability information as activities are performed using the various parts in 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 powerful, but require a specific workflow to be followed in order to make best use of them. This chapter explains the steps in this workflow.

This chapter appears first in this advanced material as the following chapters, in particularly Chapters **??** and **??**, provide key guidance on the first part of the workflow that must be followed in order for traceability to be realised. Those not wishing to use the traceability features can read chapters in any order, driven by their needs or interest. This chapter should be used in conjunction with the User Manual (Deliverable D4.3a [BLL+17]), which covers details of how to enable traceability recording in the INTO-CPS Application and baseline tools[11]. Readers interested in detailed specifications of the traceability and provenance features are directed to Methods Progress Report (Deliverable D3.3b [FGP17b]), while the tool implementation is described in Deliverables D4.2d [LNH+16] and D4.3d [KLN+17].

6.1.1 Traceability Workflow

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 too. These root nodes are *requirements*. To use fully the machine-assisted traceability features, it is necessary to initialise the traceability graph

[11] Traceability is turned off by default as it can be intrusive if the right workflow is not followed.

by using Modelio from the beginning of the development process. This means that it is necessary to follow these steps:

1. Define requirements through some requirements process (see guidance in Chapter **??**);
2. Create a Requirements Diagram (RD) in Modelio representing these requirements;
3. Create an Architecture Structure Diagram (ASD) and Connections Diagram (CD) describing the multi-model;
4. Link each requirement to one «`EComponent`» (FMU);
5. Export model descriptions for each «`EComponent`»;
6. Import model descriptions into baseline tools; and
7. Generate a multi-model configuration from the CD.

After these steps, the traceability graph will then be 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. Therefore, by following this workflow it is possible to take advantage of the machine-assisted traceability within INTO-CPS. 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.

6.1.2 What Artefacts are Traced?

Traceability in the INTO-CPS tool chain is based upon a study of the actions performed when using the INTO-CPS tool chain, the artefacts that are used and produced and a combination of two existing standards, the W3C's Prov [12] and the OMG's OSLC [13]. The combination of these resulted in the INTO-CPS traceability ontology that captures in detail all elements in the INTO-CPS workflow and describes the relationships between them. The complete ontology is presented in deliverable D3.3b [FGP17b] and a summary is presented here.

Traceability data is inherently a graph based structure based upon nodes and the connections between them, and Prov provides basic types for those nodes along with list of relationships that may exist between them. The three types of nodes are: Entities, things that may be produced or used during a devel-

[12] `https://www.w3.org/TR/prov-overview/`
[13] `http://open-services.net/`

opment process; Activities, are things that act upon and make use of entities; and Agents, objects that have responsibility for entities and activities. The Prov relations then allow then connection of nodes such as an activity may use an entity, and an entity may be generated by an activity.

The combination of the Prov nodes and relations supports the representation of the processes that lead to the generation of a particular entity, but it does not support connection of those entities to requirements. OSLC contains a set of specifications, each of which defines a list of relations that it supports between entities. In the case of the INTO-CPS traceability, parts of the OSLC architecture management and requirements management specifications are employed, these allow the connection of entities to requirements via a 'satisfies' relation indicating the entity attempts to address the needs of the requirement, additionally it allows the connection of simulation results to requirements via a 'verifies' relation indicating that the requirement has been met.

The INTO-CPS traceability ontology breaks the INTO-CPS workflow down into activities that, while not atomic if we consider a user's interaction with a particular tool, could be considered atomic when viewing the process of developing a CPS. Figure 3 shows the traceability links recorded during one step in the development of a line following robot. In this example, the requirements, R1 & R2, already exist in the architecture models and the user has created an ASD to decompose the proposed robot into components. The user has, at the same time, associated the blocks within the ASD with the the requirements that each block aims to satisfy. When the user saves the updates architecture model, the Modelio tool records the user's 'Architecture Modelling' activity, along with references to the ASD, the blocks it contains and the newly created links between the blocks and the requirements. Here the *used*, *wgb* (short for 'was generated by'), *assoc* (short for 'associated with') and *attrib* (short for 'attributed to') are links that come from the Prov standard. The *OSLC_Sat* (short for 'satisifies') comes from the OSLC requirements management specification.

A development project will likely consist of many instances of the activities identified in the ontology being performed and together they form a traceability graph. Figure 4 shows a simplified view of a traceability graph with some steps removed for brevity. At the top of the graph we see the architecture modelling step described previously, that produces an architecture model. From the architecture, model description files are exported to start the production of the simulation models. In turn the simulation models are exported as FMUs and the FMUs are used to produce simulation results.

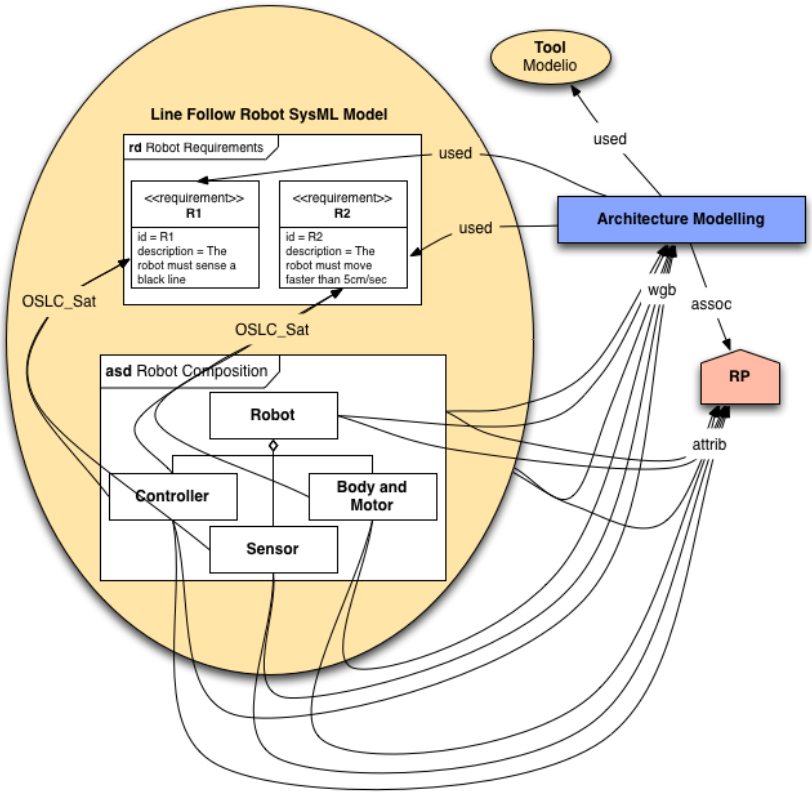


Figure 3: Traceability links captured during the production of an ASD for a line following robot.

Key to the traceability graph then are the 'used' and 'wgb' connections that can be used by a query to determine from where each entity was generated. By following these links back from any entity to the individual blocks within the architecture model, it is possible to determine which requirement(s) each should satisfy. Finally when simulation results are output, these may be linked back to the relevant requirements, stating whether a requirement was verified or violated by that result.

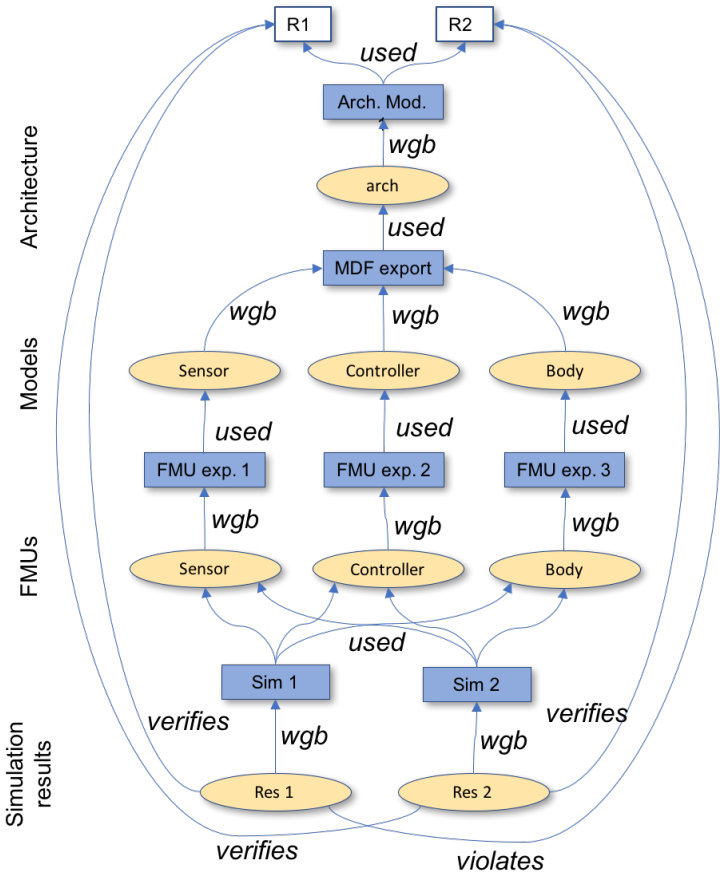


Figure 4: Traceability links captured during the production of an ASD for a line following robot.

The traceability ontology captures the significant activities and entities that form the INTO-CPS workflow. For example a development project might see the following activities recorded in the traceability graph: *Requirements Management*, *Architecture Modelling*, *Architecture Configuration Creation Model Description Export*, *Simulation Modelling*, *FMU Export*, *Configuration Creation*, *Simulation Configuration Creation* and then *Simulation*. These activ-

ities are connected in the workflow by the entities they create and use, so the example would see the traceability graph containing records of: *Architecture Structure Diagram*, *Architecture SubSystem*, *Architecture Connection Diagram*, *Model Description File*, *Simulation Model*, *FMU*, *Multi-model Configuration*, *Simulation Configuration* and *Simulation Result*. Alongside these will be records of the agent(s), who are both associated with activities and have entities attributed to them.

6.1.3 Traceability Queries

The traceability graph created by the INTO-CPS tool chain uses a graph database tool called *Neo4J*. Once a graph has been built, queries can be executed over the graph to perform both forwards and backwards traceability. Below are some types of queries that can be executed over the graphs. The INTO-CPS Application supports some of these queries with the GUI, and the rest through inline access to the Neo4J console.

1. Impact analysis
   - Forward traceability (from requirements to entities)
   - Backwards traceability (from FMU to requirements)
   - Backwards traceability (from components to requirements)
2. Simulation sources
   - Find all simulations
   - Find sources and sinks for a simulation
3. Coverage
   - Requirements without architecture elements
   - Requirements without simulation models
   - Requirements without FMUs
   - Requirements without positive simulation results
   - Requirements without any simulation results
4. Code sources
   - Find all generated source code entities
   - Find the models for a given source code entity
5. User impact
   - Find all users in the database
   - Find all artefacts influenced by a user
   - Find all activities performed by a user

Queries are written in Cypher, a query language built into Neo4J. Advanced users or those developing extensions to INTO-CPS can build their

own queries in Cypher[14] and execute them using Neo4J directly as described in the User Manual (Deliverable D4.3a [BLL+17]).

6.2 Requirements Engineering

In this chapter, we consider the requirements engineering (RE) activities for the design of CPSs. Specifically, we consider the specification and documentation of requirements placed upon a CPS. These requirements may, for example, impose restrictions, define system capabilities or identify qualities of a system. The requirements should indicate some value or use for the different stockholders of a CPS.

As described in the previous chapter, traceability needs requirements to be defined as early as possible in a development process, and these must be recorded in Modelio for the machine-assisted traceability information to be recorded accurately. It is therefore appropriate to consider requirements processes for such developments at this stage.

In this remainder of this chapter, we discuss the needs for requirements engineering in CPS development, in particular based on the experience of the industrial partners for INTO-CPS. We describe one possible approach to RE for CPS, specifically adapting the SoS-ACRE approach for systems-of-systems (SoSs) to CPS. Note however that this approach is not mandatory, and in general RE processes and tools vary widely across organisations and domains. For this reason, tool support for traceability in INTO-CPS begins once requirements have been defined and can be added to Modelio. The diagrams described in the example are not part of INTO-CPS SysML specification. Therefore, this chapter should truly be treated as guidance, primarily serving to highlight the nature of RE for CPS, which may be of use for both new and more experienced CPS teams.

6.2.1 Requirements Engineering and Cyber-Physical Systems

The main issue of concern for RE in CPSs is that of differing domain contexts [WGS+14]. In addition, it has been noted that there are overlaps in challenges in CPSs and SoSs [PE12]— especially independence, evolution and, increasingly, distribution. As described by Lewis et al. [LMP+09], as system architectures become more complex, there is often a need to consider

[14] `https://neo4j.com/developer/cypher-query-language/`

requirements and structural architectures during the RE process. The authors suggest that an engineer should identify the system needs, component interactions and stakeholders, and map those needs onto those interested parties.

As research in RE in CPS is a nascent field, we suggest one approach is to adopt RE processes from the SoS world, rather than defining an approach specifically for CPSs. In chapter, we consider SoS-ACRE (System of Systems Approach to Context-based Requirements Engineering) [HPP+15], as an example. This approach was adapted from standard systems engineering, and tailored for SoSs— enabling the identification and reasoning about requirements across constituent systems of an SoS and understanding multi-stakeholder contexts. We suggest it might be useful to organisations trying to approach RE for CPS.

6.2.2 The SoS-ACRE View of Requirements

We first consider the collection of views defined in SoS-ACRE, and their applicability to CPS engineering and the INTO-CPS tool chain. These views could be represented as diagrams in SysML[15], or as we describe, could equally be represented in other tools where these are already used (e.g. Excel). Examples of each view are shown in Figures 7, 8, 5 and 6.

Source Element View (SEV) The SEV defines a collection of source materials from which requirements are derived. In SoS-ACRE, a SysML block definition diagram is considered. In INTO-CPS, this view could also be represented using an Excel table or Word document (with each source having a unique identifier), or by simply referring to source documents using OSLC traces.

Requirement Description View (RDV) The RDV is used to define the requirements of a system and forms the core of the requirement definition. SoS-ACRE suggests the use of SysML requirements diagram or in tabulated form, such as through the use of Excel. In addition, specifying requirements in Doors would support this view.

Context Definition View (CDV) The CDV is a useful view for CPS engineering in order to explicitly identify interested stakeholders and points of context in the system development, including customers, suppliers and system engineers themselves. In SoS-ACRE, they are defined

[15]Note that SoS-ACRE is not specifically supported as a Modelio plug-in, but other equivalent diagrams could be used.

using SysML block definition diagrams, and could also be represented using an Excel table or Word document (with each context having a unique identifier). This diagram type could be useful when identifying the divide in CT/DE and cyber-physical elements of a system.

Requirement Context View (RCV) In SoS-ACRE, a RCV is defined for each constituent system context identified in CDVs. This is appropriate when there is a set of diverse system owners, which is typical for SoSs and increasingly CPSs. A **Context Interaction View (CIV)** is then defined to understand the overlap of contexts and any common/-conflicted views on requirements. In a CPS, however, there may not be such a clear delineation between the owners of constituent system components. However, if we consider the different domains (e.g. CT/DE or cyber/physical divides) as different contexts, then this approach would be useful. In SoS-ACRE, RCVs and CIVs are both defined with SysML use case diagrams. Excel could be used if unique identifiers are defined for contexts and requirements as described earlier.

Validation View (VV) VVs, defined as SysML sequence diagrams in SoS-ACRE, describe validation scenarios for a SoS to ensure each constituent system context understands the correct role of the requirements in the full SoS. This is not an obvious fit in CPS engineering, and therefore not necessarily required.

6.2.3 The SoS-ACRE RE Process

The SoS-ACRE requirement engineering process may be useful for organisations wishing to better understand requirements for CPSm, particularly across multiple domains. It is a lightweight process, and therefore suitable for small- to medium-sized enterprises. Organisations with established may not feel the need to radically alter their existing practice, but may find it instructive to consider how their current processes might be updated or revised to consider better CPS requirements.

A SoS-ACRE process for CPS should include the following steps:

1. Identify and record source elements. This would be using a SEV, or simply recording paths to relevant files or documents.
2. Record system-level functional and non-functional requirements. Requirements may be derived using RDVs, and we could consider domain-specific requirements (e.g. cyber or physical), or analysis-specific re-

quirement types (e.g. DSE or testing requirements).

3. Model initial System structure using INTO-CPS ASD. This will identify the cyber and physical elements and the domain/phenomena of the CPS. This may also give initial idea of component functionalities, which may lead to a repeat of Step 2 above[16].

4. Define the various contexts in CDVs – both external stakeholders, and if appropriate, contexts for the different components. If only a single system context is defined, then a single RCV is defined. However, if multiple contexts are defined for a CPS, then several RCVs are to be defined, along with a CIV to explore requirements from multiple contexts.

5. Trace the requirements through INTO-CPS tool chain models and results. This was covered in the previous chapter, however we revisit it below in the context of requirements.

6.2.4 Using technologies with SoS-ACRE

As INTO-CPS does not specifically support SoS-ACRE. Indeed INTO-CPS does not mandate and specific approach to RE, because of the wide variety of approaches in industry. We conclude this chapter with an example of how a SoS-ACRE (or other RE process) could be integrated into an INTO-CPS development. We describe a range of permutations of the use of models and documents for recording the requirements engineering process described above. In addition, we include discussions on the links between requirements and architectural models— identified above as a key method for requirements engineering in CPSs.

URI, Excel and SysML We first consider an approach using URIs for the source elements, an Excel document (or a collection of Excel tables) for the RDV, CDV, RCV and CIV of SoS-ACRE. A SysML model in Modelio can be used to define the architecture of the multi-model. Internal tracing in Excel can be achieved using identifiers referenced between sheets. Excel requirements can be replicated in Modelio then traced to elements in the INTO-CPS tool chain automatically. Figure 5 presents an example with URI, Excel and SysML models and OSLC links between the artefacts.

[16]In the process of architectural modelling, it may also be necessary to redefine contexts depending on whether different simulation tools, or indeed different components of a model, are better able to provide the requirements of the CPS.

Excel and SysML The next approach uses Excel to define the SEV and RDV of SoS-ACRE, a SysML model to define the context-oriented views (CDV, RCV and CIV) and a separate architectural model to define the CPS architecture. The Excel requirements can then be mirrored in a Modelio model, and linked to the architectural model. The INTO-CPS traceability features can trace the requirement artefacts to the architectural model. Additional OSLC links could be added manually to link elements of the Excel requirements and context views in a SysML. Figure 6 presents an example with URI, Excel and two SysML models with OSLC links between the artefacts.

Single SysML model The next permutation is to use a single SysML model for both requirements engineering and architectural modelling. Such a model will contain all SoS-ACRE views[17] (SEV, RDV, CDV, RCV and CIV), in addition to diagrams defined using the INTO-CPS profile for the CPS composition and connections. Modelling in this way enables trace links to be defined inside a single SysML model. Figure 7 presents an example SysML model with trace relationships.

SysML requirements and SysML architectural models The final permutation is to use SysML for both requirements engineering and architectural modelling, however to use two separate models for the two activities (one containing the RE views (SEV, RDV, CDV, RCV and CIV) and another for architectural diagrams (ASD and CD)). We consider this permutation with two SysML models in addition to the single SysML model, because the requirements engineering and architectural modelling activities are often considered separately, with different engineering teams comprised of engineers with specialist skills. As such we can assume there are cases where these teams have ownership of different models. Trace links may be used within each individual model (for example, tracing from source elements to requirements in a RE model), and OSLC links defined to trace between requirements elements and architectural elements. Figure 8 presents an example with two SysML models with trace relationships and OSLC links between the models.

[17]Note that Modelio does not currently provide an extension for SoS-ACRE, but these views can be realised using existing SysML stereotypes.

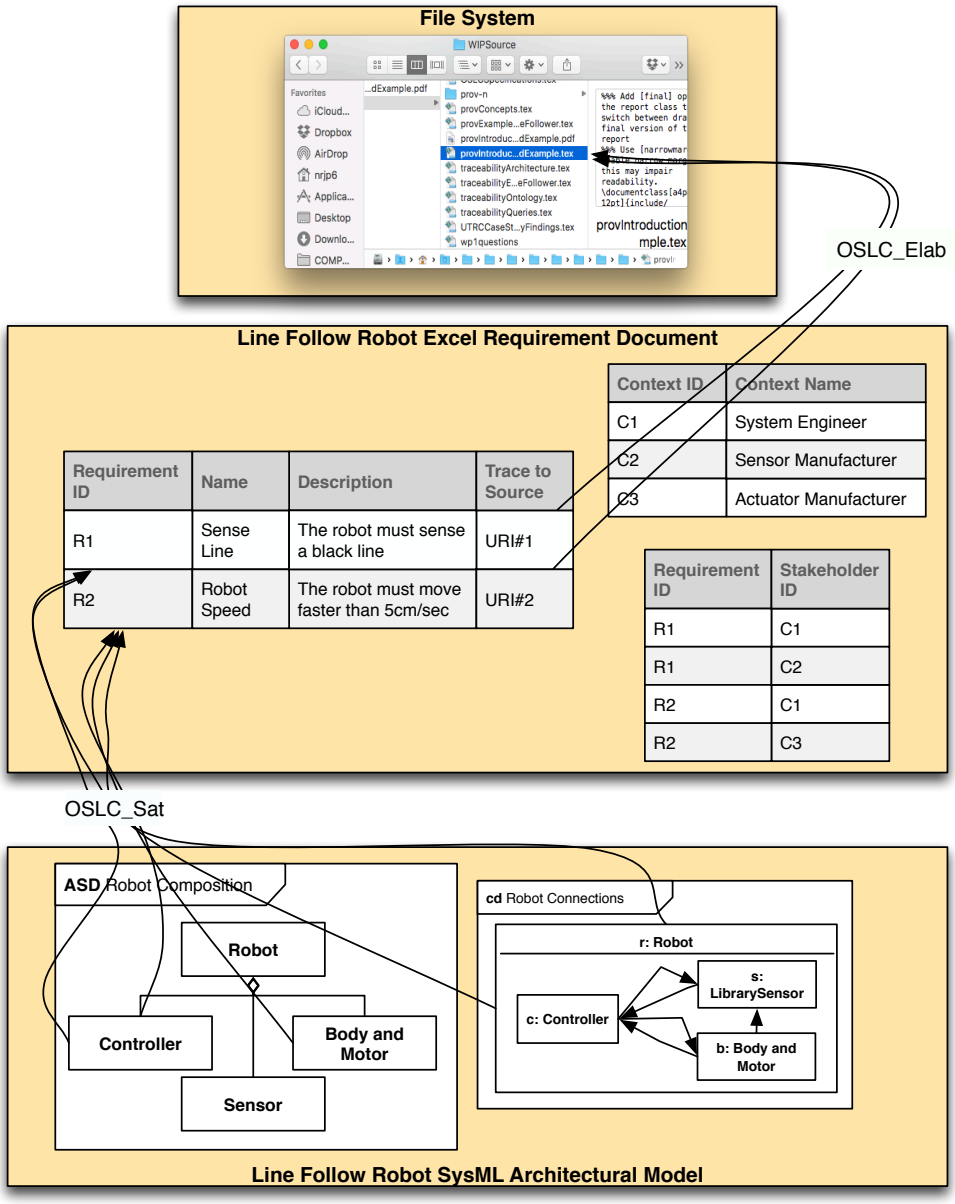


Figure 5: URI, Excel and SysML – model overview

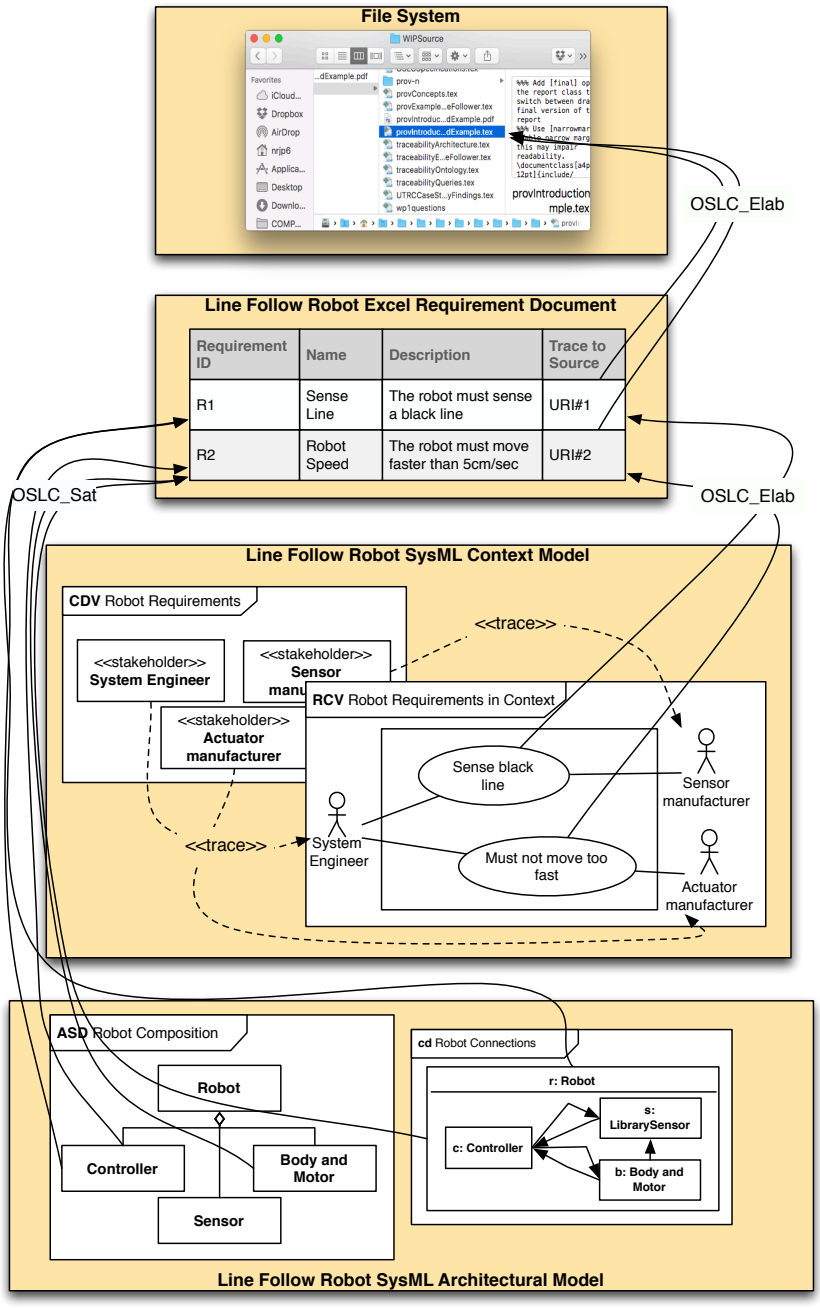


Figure 6: Excel and SysML – model overview

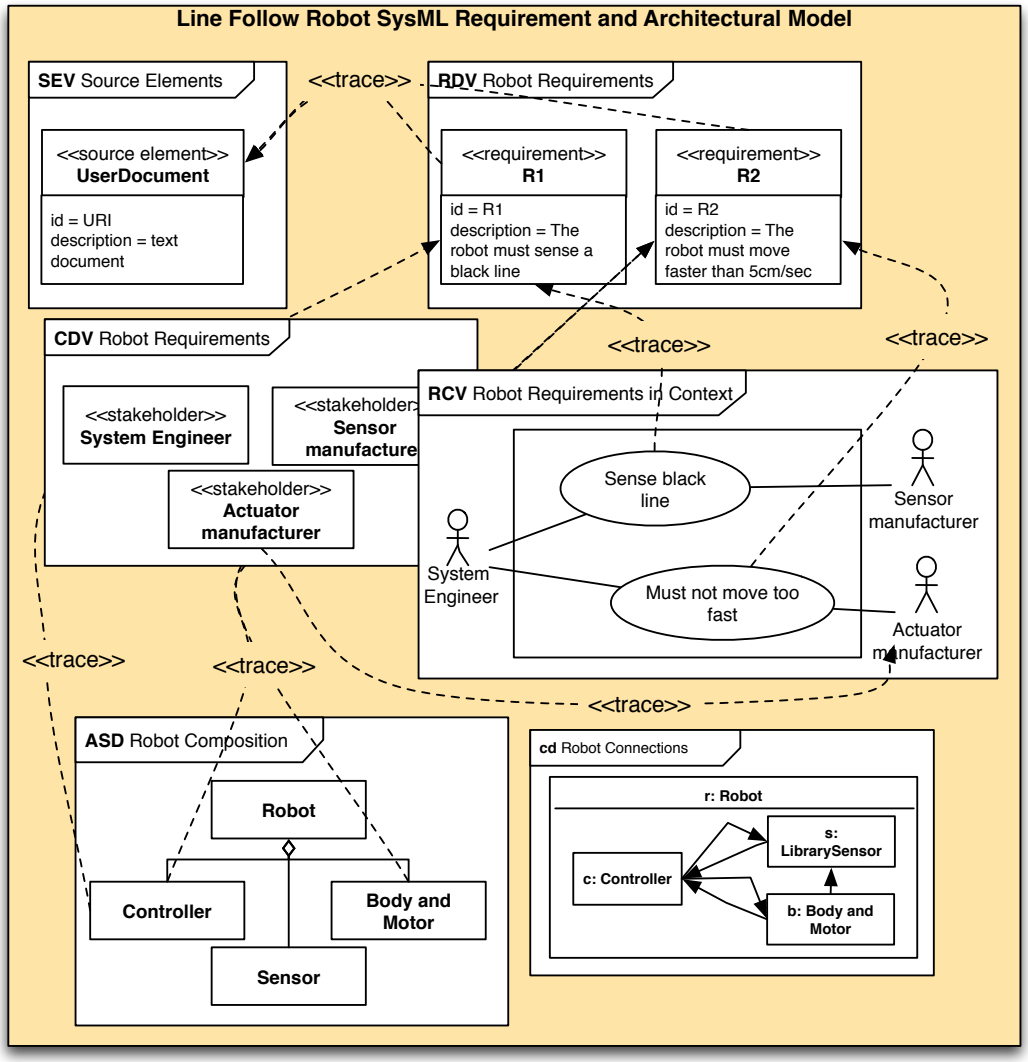


Figure 7: Single SysML model – model overview

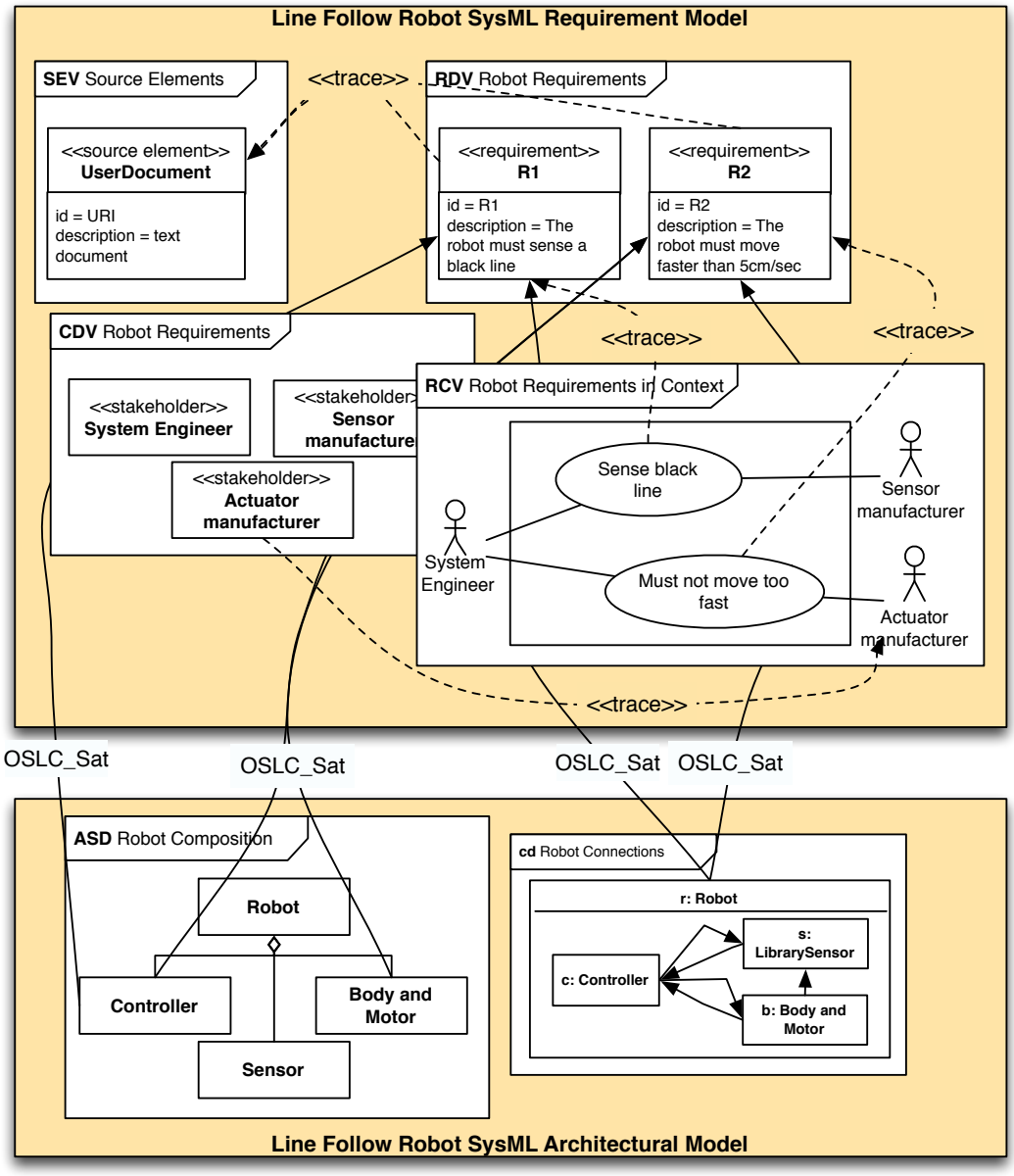


Figure 8: SysML requirements and SysML architectural models – model overview

6.3 SysML and Multi-modelling

This chapter describes the use of SysML with the INTO-CPS tool chain. As described previously in Chapter **??**, 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 help users to *configure multi-models for co-simulation* and *configure design space exploration (DSE) analysis* [APCB15, ACM+16a, ZCWO17b, BQ15, BQ16, Bro17]. For ease explanation, we describe these separately below, however all the diagrams described are part of a single extended SysML profile.

This chapter summarises the diagrams provided in the two profiles and describe their use in Sections 6.3.1 and 6.3.2. The diagrams presented are illustrative, showing the main elements of a diagram; they are not full definitions of the meta-model, which can be found in the documents cited above. All diagrams are supported by the Modelio tool, and we refer readers to the user manual, Deliverable D4.3a [BLL+17], for further information on how to use Modelio to draw these diagrams and generate configurations for use in the INTO-CPS Application.

The chapter concludes with an example of the relationship between a *holistic* model created using standard SysML and a *design* model using the INTO-CPS profile, and concludes with a discussion on how to represent non-design elements (such as FMUs that only perform visualisation) in the INTO-CPS profile in Section 6.3.4.

6.3.1 SysML Diagrams Describing Multi-models

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, using the JSON format to describe the FMUs, their parameters and connections between them. These can be created manually in a text editor, or from the INTO-CPS Application itself. Alternatively, a configuration can be generated by Modelio from the diagrams defined in this profile. There are two types diagram, the *Architectural Structure Diagram* describing the static structure of FMUs, and the *Connections Diagram* describing their instantiation and connections. These are shown in Figure 9.

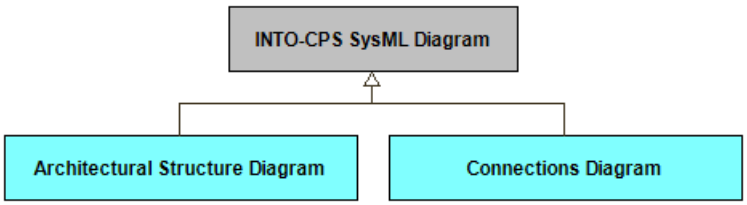


Figure 9: Diagrams in the multi-modelling SysML profile

Architectural Structure Diagram The *Architecture Structure Diagram* (ASD) specialises SysML block definition diagrams (BDDs) to support the specification of a multi-model architecture described in terms of a systems components, which will be represented by FMUs. As shown in Figure 10 this diagram must include a «`System`» which is then broken down into zero or more «`Component`» blocks.

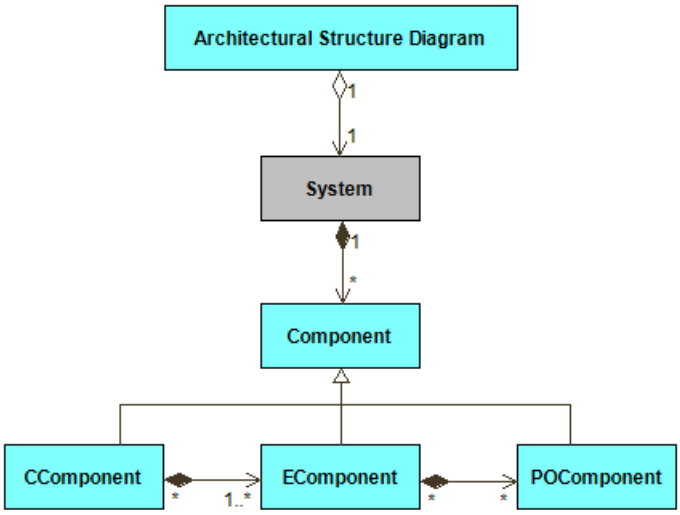


Figure 10: *Architectural Structure Diagram* describing FMUs (EComponents) and their hierarchies

There are three types of component block. The «`EComponent`» (encapsulating component) represents a part of a system that will be represented by a single FMU. These blocks have properties indicating which modelling language and tool will be used: `modelType` (*discrete* or *continuous*) and `platform` (*VDMRT*, *TwentySim*, *OM*, and *other*).

An «`EComponent`» can be broken down logically into «`PComponent`» (part-of component) representing an internal element of an «`EComponent`».

Both «EComponent» and «PComponent» blocks can define *variables* and *FlowPorts* that an FMU will have.

The third type of component is a «CComponent» (collection component) that allows other components to be grouped logically (it has no ports or behaviours). These can be used to separate design elements within a diagram, as described in Section 6.3.4. All component blocks have a *kind* that marks their purpose in the model (*cyber*, *physical*, *environment*, *visualisation*).

FMUs are connected by *ports*, and may also present internal state through externally visible *variables*, which can be monitored on a live graph, for example. Both «EComponent» and «PComponent» blocks can define *FlowPort* and *Variable* attributes, as shown in Figure 11, which will form the interface of the FMU and are added to the "model description" exported by Modelio.

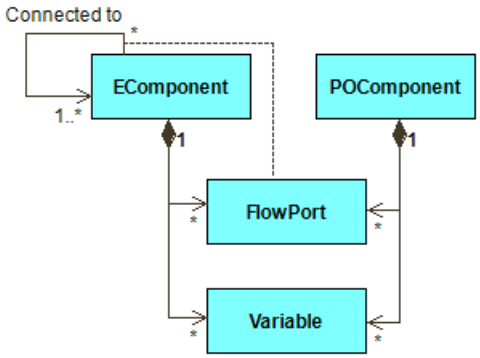


Figure 11: Component blocks may define variables and ports

Connections Diagram The *Connections Diagram* (CD) specialises SysML internal block diagrams to convey the internal configuration of the systems components. Specifically, it describes which FMUs are instantiated (i.e. which «EComponent»s form the ASD), and how the ports are connected. This diagram is used by Modelio to generate multi-model configurations.

6.3.2 SysML Diagrams Describing Design Space Exploration

The design space exploration (DSE) SysML profile is an addition to the multi-modelling SysML profile described above. As with single co-simulation, the

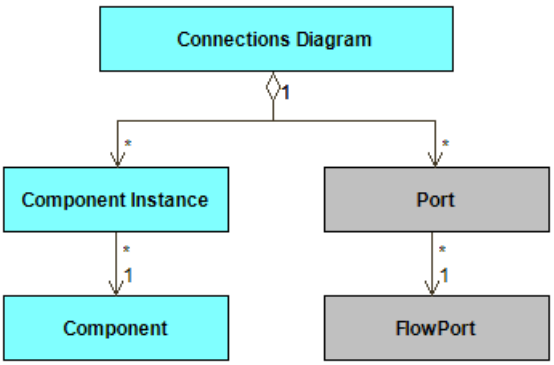


Figure 12: *Connections Diagram* describing the static structure of FMUs

INTO-CPS Application can run a DSE based on a JSON configuration file. These can be created manually in a text editor or edited in the INTO-CPS Application. Alternatively, a configuration can be generated by Modelio, from a set of diagrams defined in the profile. There are five diagram types, which are described below. Further guidance on DSE can be found in Chapter **??**.

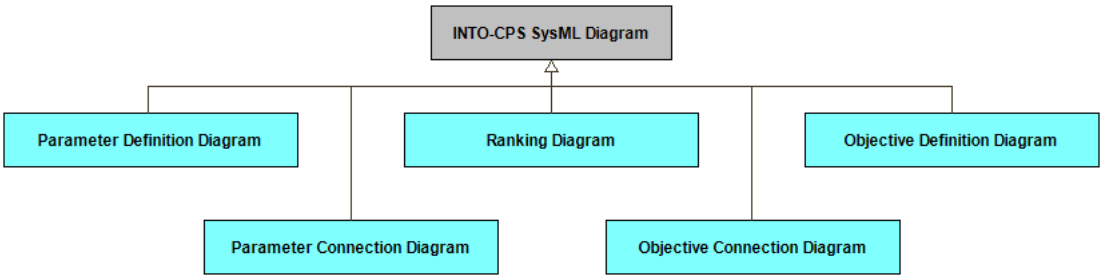


Figure 13: Diagrams in the DSE SysML profile

Objective Definition Diagram The *Objective Definition Diagram* is used to define the objectives for use during a DSE. Objectives are characterising measures of performance that may be used to determine the relative benefits of competing designs. They are defined as metrics over the results of a co-simulation of a specific design and are used to judge its quality for use in later processing e.g. ranking.

Objectives are described in terms of a name, a script file that will be used to compute them, and the ports that will provide the data they require. As with the *Architectural Structure Diagram* above (Section 6.3.1), this diagram

gives the static structure of the objectives; instances of these definitions are created using the *Objective Connection Diagram* below (Section 6.3.2).

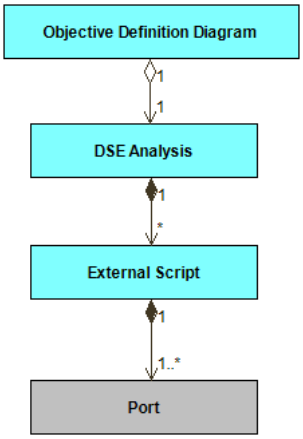


Figure 14: *Objective Definition Diagram* describing objectives in a DSE

Objective Connection Diagram The *Objective Connection Diagram* is used to instantiate objectives defined in the *Objective Definition Diagram* above (Section 6.3.2). The diagrams allow the ports of each instance of the objective to be linked to a data source: either a static value, or a value from data exchanged in the multi-model.

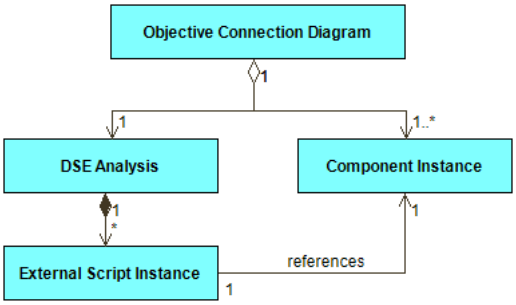


Figure 15: *Objective Connections Diagram* linking objectives to data sources

Parameter Definition Diagram The *Parameter Definition Diagram* is used to define the parameters that will changed for each co-simulation in a DSE. Parameters are described in terms of a name, and a set of values that we wish to test. The product of the cardinalities of the set of values for each parameter gives the size of the design space— the total number of simulation required for an exhaustive search. As with the *Architectural Structure*

Diagram above (Section 6.3.1), this diagram gives the static structure of the parameters; instances of these definitions are created using the *Parameter Connection Diagram* below (Section 6.3.2).

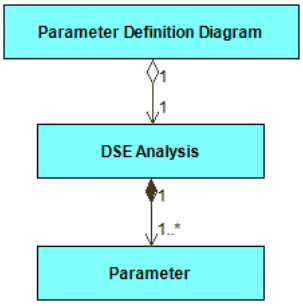


Figure 16: *Parameter Definition Diagram* defining parameters and their values

Parameter Connection Diagram The *Parameter Connection Diagram* is used to instantiate parameters defined in the *Parameter Definition Diagram* above (Section 6.3.2). The diagram allows the parameters to be linked to those provided by the FMUs in the multi-model.

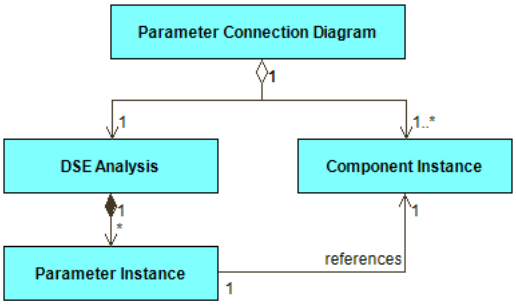


Figure 17: *Parameter Connections Diagram* linking parameters to FMUs

Ranking Diagram The *Ranking Diagram* is used to declare which of the objectives should be used to compare competing designs, and whether lower or higher values for each the objectives is better (i.e. whether to maximise or minimise a value).

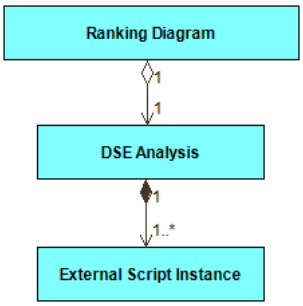


Figure 18: *Ranking Diagram* defining how to rank designs based on objectives

6.3.3 Holistic and Design Architectural Modelling

A system architecture defines the major components of a system, and identifies 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. Following on from Chapter **??**, this section uses a smart grid example to show both holistic and design architectural modelling approaches, and provide 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.

Example Introduction A smart grid is an electricity power grid where integrated ICT systems play a role in the control and management of the electricity power supply. Such ICT elements include distributed control in households, control of renewable energies and networked communications. In this section we outline a Smart Grid model to explore different design decisions in the cyber control of an electricity power grid. The model presented here is a small illustrative example, which omits complexities of a real Smart Grid. For example, the change from three-phase AC power to one-phase DC power allowing us to use simpler physical models. A second simplification is in the number of houses present in the grid model. We model only 5 houses,

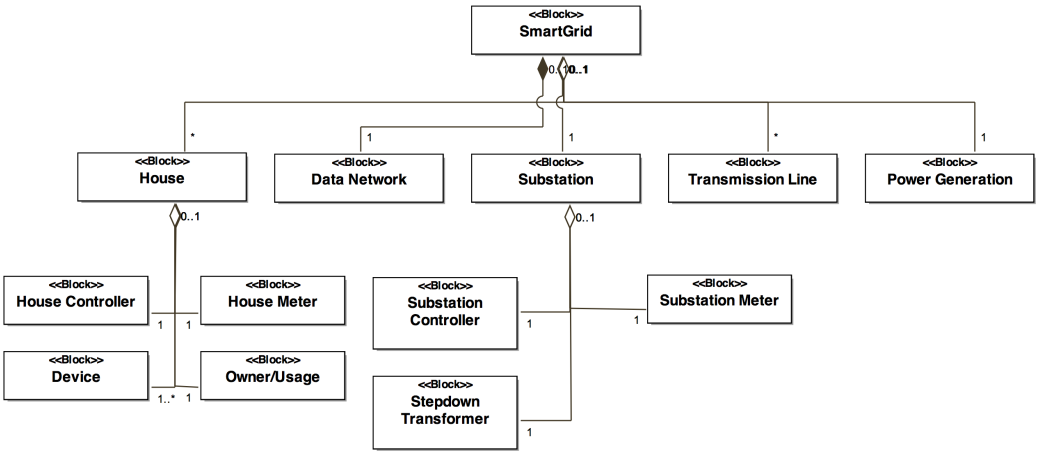


Figure 19: Block Definition Diagram of Smart Grid

assumed to be in a small local area supplied by a single substation. We do not consider the remainder of the grid. To ensure that any effect due to changes in the power consumption by those properties are observed by the other houses, we skew the resistance of the transmission lines between the power generation and substation, and substation to houses.

Holistic Architectural Model A Block Definition Diagram (BDD) of the Smart Grid is given in Figure 19. The figure shows that the *Smart Grid* system comprises two top-level physical elements: *Power Generation* and *Transmission Lines*; a single top-level cyber elements: the *Data Network*; and two cyber-physical systems: a *Substation* and several *Houses*. The two elements may be further decomposed. The *Substation* elements is composed of a cyber *Substation Controller* and physical *Substation Meter* and *Stepdown Transformer*. The *House* element comprises: a cyber *House Controller*, physical *House Meter* and *Devices*, and an *Owner/Usage Profile*.

An Internal Block Diagram (IBD) of the Smart Grid is given in Figure 20. The diagram shows there are two main connection types in the model, corresponding to the physical power connections and the cyber data connections. The model also shows the connections between the cyber and physical parts of the models – currently modelled using data-type connections.

The first type of connection —the physical power connections— show a flow of *Power* from the Power Generation, through the Transmission Lines to the Houses, via the Substation. In the Substation, the Stepdown Transformer is connected to the Substation Meter. Similarly, in each House (only one is shown in the figure), the Power flows through the House Meter to each

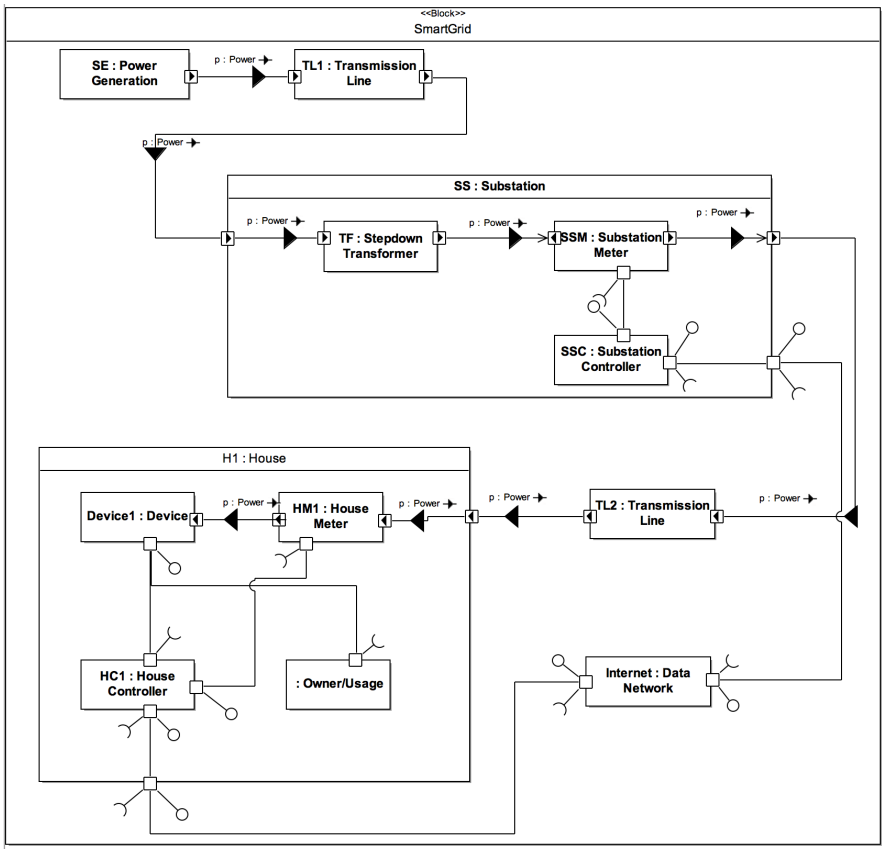


Figure 20: Internal Block Diagram of Smart Grid

Device (again only one is shown for readability). The data connections exist between the Substation Controller and House Controllers. The Data Network is explicitly modelled and links the various controllers. Finally, there are links between the cyber controllers and the physical systems. In this model, the Substation Controller is connected to the Substation Meter, and the House Controller is linked to the House Meter and Devices.

Design Architectural Model Looking at the holistic architecture defined in Figures 19 and 20 and moving towards a multi-model, we use the INTO-CPS SysML profile to define the architecture of the Smart Grid from the perspective of multi-model. This yields the ASD in Figure 21. This structure removes all subsystem structures such that each component is to be realised in a single FMU. Each element is defined as either a *physical* or *cyber* component, with the model type and platform identified.

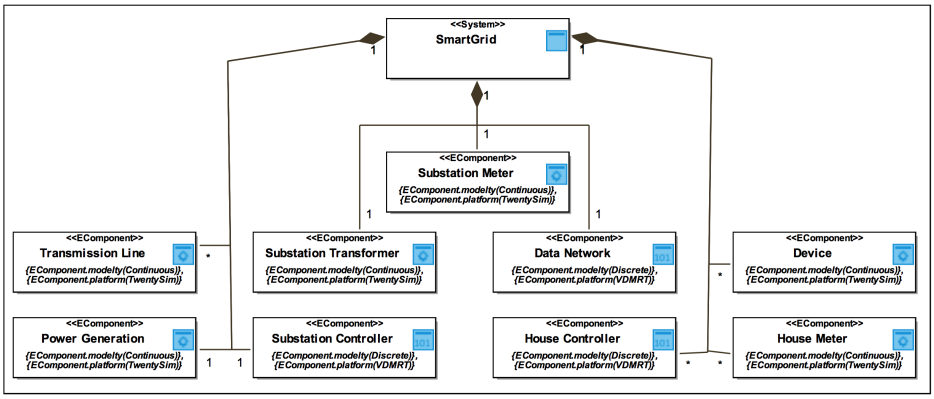


Figure 21: Architecture Structure Diagram for multi-model of Smart Grid

The connections between the components are defined in the Connections Diagram (CD) in Figure 22. The interface between subsystems is defined as the interaction points between cyber and physical components (FMUs).

Discussion Contrasting the architectures shown in the initial model (Figures 19 and 20) to that in the multi-model (Figures 21 and 22), whilst the same base components are present in both, some of the intuitive domain-specific structures are lost when moving to a multi-model. For example, it is now not clear where the *substation* or *house* elements are in the multi-model.

An important issue here is in the reason behind producing different architectural models. Using SysML diagrams in a *holistic* approach, a CPS engineer

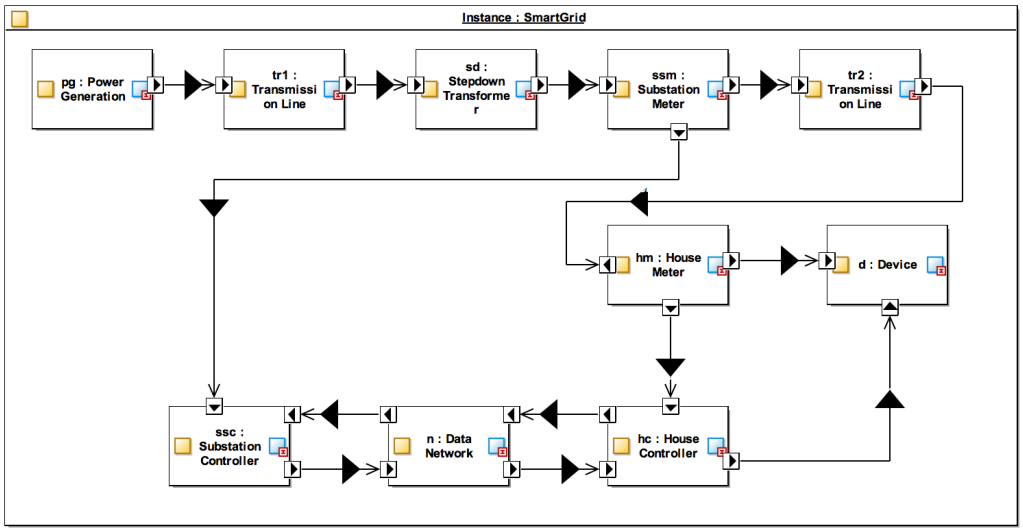


Figure 22: Connections Diagram for multi-model of Smart Grid

describes the model using a structure natural to the application domain. As such, the *reason* for modelling is not in the ultimate analysis to perform, but to define and understand the structure and behaviour of a system. In contrast, the *design* approach is necessary to configure INTO-CPS multi-models from SysML.

Figure 23 presents an overview of the relationships between the different types of models. The figure shows that the 'real' system may be modelled in different forms: the holistic and design architectures and the multi-model.

As illustrated in the figure, one approach can inform another. In some cases this may be a natural process; for example in the Smart Grid example, isolating each of the lowest level components in Figure 19 to be individual FMUs in a multi-model is an evolution which will likely result in a feasible model. By creating a domain-specific holistic architecture first, then transforming these models into a design architecture for multi-modelling, design teams will likely gain the most benefit.

6.3.4 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

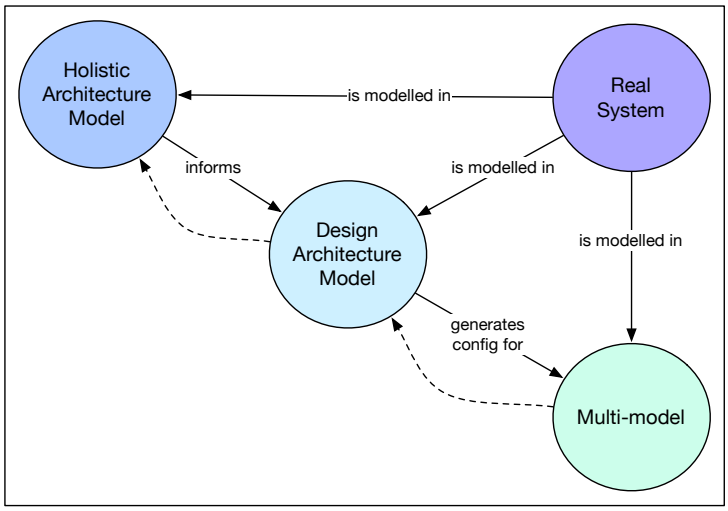


Figure 23: Relating holistic and design architectures

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.

Here we present a small example of the use of these extensions, using a simple robot example (based on the line-following robot pilot study, see Deliverable D3.6 [MGP+17]) to illustrate the use of «`CComponent`»s and the *kind* of components (*cyber*, *physical*, *environment*, *visualisation*) described in Section 6.3.1 above.

The architecture structure diagram in Figure 24 shows: a *System_Env* block, an «`EComponent`» defined as an `Environment` FMU; a *3D_View* block an «`EComponent`», defined as an `Visualisation` FMU; and an *Example_Robot* block, an «`EComponent`» defined as an `composition` of two FMUs.

The example has two connection diagrams. The first is shown in Figure 25, it contains only those connections with respect to the system and its constituent components . This diagram shows a block instance *cps1* containing the environment (e) and the example robot (r) which contains two the controller and hardware components.

The second is shown in Figure 26, it depicts the use of the block instance *3D* of type *3D_View*. In this diagram, we show additional ports of the original block instances to output internal model details and connect these

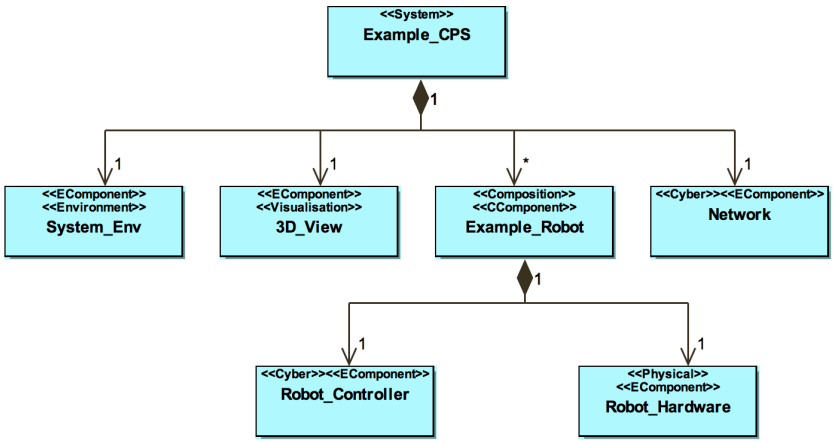


Figure 24: Example Architecture Structure Diagram of robot system

to the *3D* instance. The diagram includes the system connectors as shown in Figure 25.

Initial Multi-Modelling using a Discrete-Event Notation: VDM In this section we provide guidance on producing initial multi-models from architectural descriptions produced using the INTO-CPS SysML profile. We focus on using discrete-event (DE) models to produce initial, abstract FMUs that allow integration testing through co-simulation before detailed modelling work is complete. This is called a "DE-first" approach [FLPV13, FLV13]. We describe the use of VDM and the Overture tool, with FMI export plug-in installed, for this approach. The principles outlined in this section can be applied in other modelling tools. This approach can work with or without the SysML profile.

6.4 The DE-first Approach

After carrying out requirements engineering (RE), as described in Chapter **??**, and design architectural modelling in SysML, as described in Chapter **??**, the engineering team should have the following artifacts available:

- One or more Architecture Structure Diagrams (ASDs) defining the composition of
  «`EComponent`»s (to be realised as «`Cyber`» or «`Physical`» FMUs) that will form the multi-model.
- Model descriptions exported for each «`EComponent`».

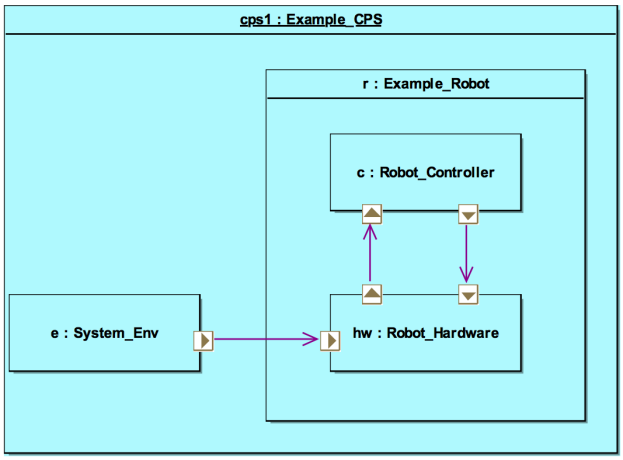


Figure 25: Connections Diagram for robot showing only system and environment connectors

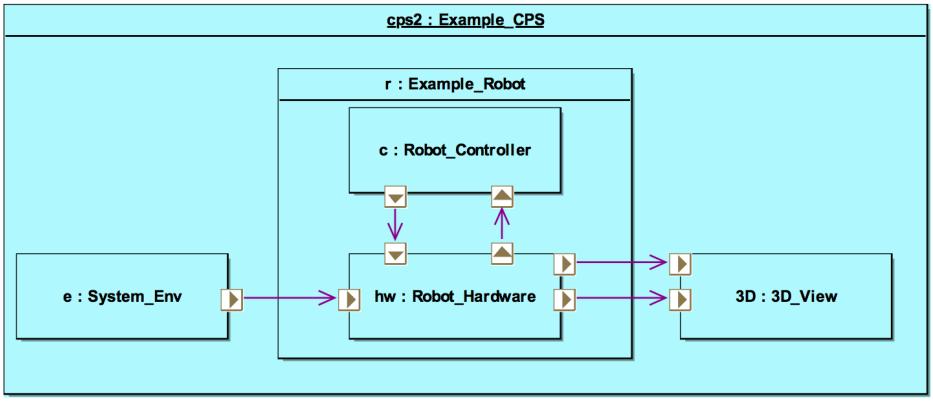


Figure 26: Connections Diagram for the robot system showing the system *and* visualisation components

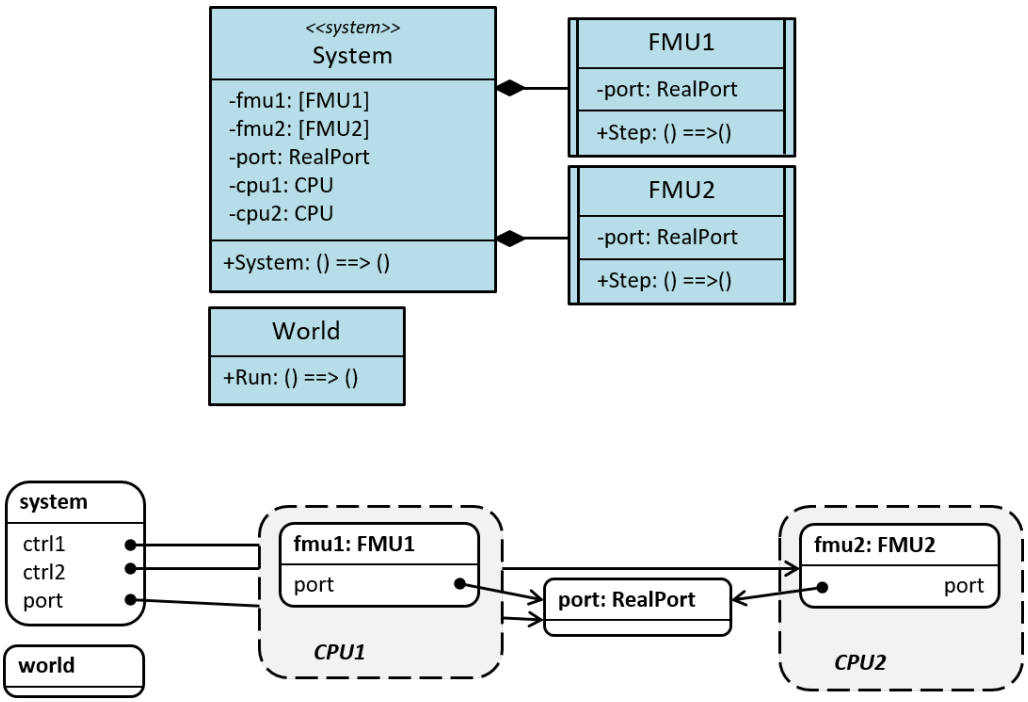


Figure 27: Class diagram showing two simplified FMU classes created within a single VDM-RT project, and an object diagram showing them being instantiated as a test.

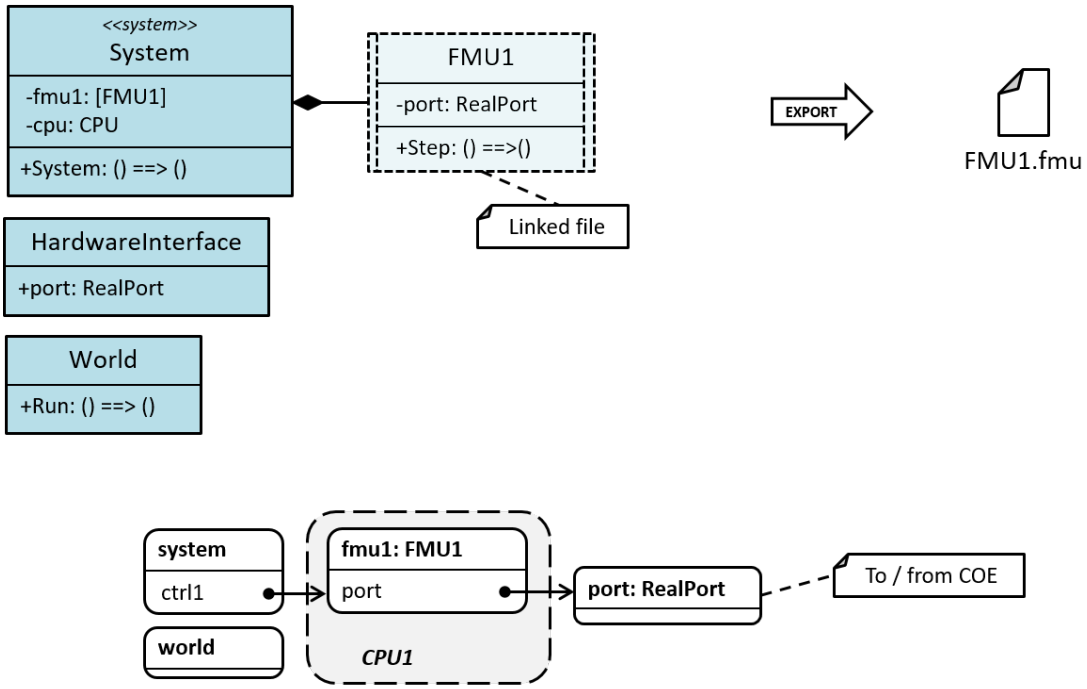


Figure 28: Class and object diagrams showing a linked class within its own project for FMU creation.

- One or more Connections Diagrams (CDs) that will be used to 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 «`EComponent`» 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].

Guidance on how to produce DE approximations for use in multi-modelling, and in particular approximations of CT behaviour, can be found in material describing the Crescendo baseline technology [FLV13], which is also available via the Crescendo website[18].

6.4.1 DE-first within INTO-CPS

Given an architectural structure diagram, connections diagram and model descriptions for each «`EComponent`», the suggested approach is to begin by building a single VDM-RT project in Overture with the following elements:

- A class for each «`EComponent`» representing an FMU. Each class should define port-type instance variables (i.e. of type `IntPort`, `RealPort`, `BoolPort`, or `StringPort`) corresponding to the model description and a constructor to take these ports as parameters. Each

[18] See `http://crescendotool.org/documentation/`

FMU class should also define a thread that calls a `Step` operation, which should implement some basic, abstract behaviour for the FMU.
- A `system` class that instantiates port and FMU objects based on the connections diagram. Ports should be passed to constructor of each FMU object. Each FMU object should be deployed on its own CPU.
- A `World` class that starts the thread of each FMU objects.

Class and object diagrams giving an example of the above is shown in Figure 27. In this example, there are two «`EComponent`»s (called *FMU1* and *FMU2*) joined by a single connection of type real. Such a model can be simulated within Overture to test the behaviour of the FMUs. This approach can be combined with the guidance in Chapter **??** to analyse more complicated networked behaviour. Once the behaviour of the FMU classes has been tested, actual FMUs can be produced and integrated into a first multi-model by following the guidance below.

6.4.2 FMU Creation

The steps outlined below assume a knowledge of FMU export in Overture, which can be found in the User Manual, Deliverable D4.3a [BLL+17], in Section 5.1. To generate FMUs, a project must be created for each «`EComponent`» with:

- One of the FMU classes from the main project.
- A `HardwareInterface` class that defines the ports and annotations required by the Overture FMU export plug-in, reflecting those defined in the model description.
- A `system` class that instantiates the FMU class and passes the port objects from the `HardwareInterface` class to its constructor.
- A `World` class that starts the thread of the FMU class.

The above structure is shown in Figure 28. A skeleton project with a correctly annotated `HardwareInterface` class can be generated using the model description import feature of the Overture FMU plug-in. The FMU classes can be linked into the projects (rather than hard copies being made) from the main project, so that any changes made are reflected in both the main project and the individual FMU projects. These links can be created by using the *Advanced* section of the *New > Empty VDM-RT File* dialogue, using the `PROJECT-1-PARENT_LOC` variable to refer to the workspace directory on the file system (as shown in Figure 29). Note that if the FMU classes need to share type definitions, these can be created in a class called `Types` in the

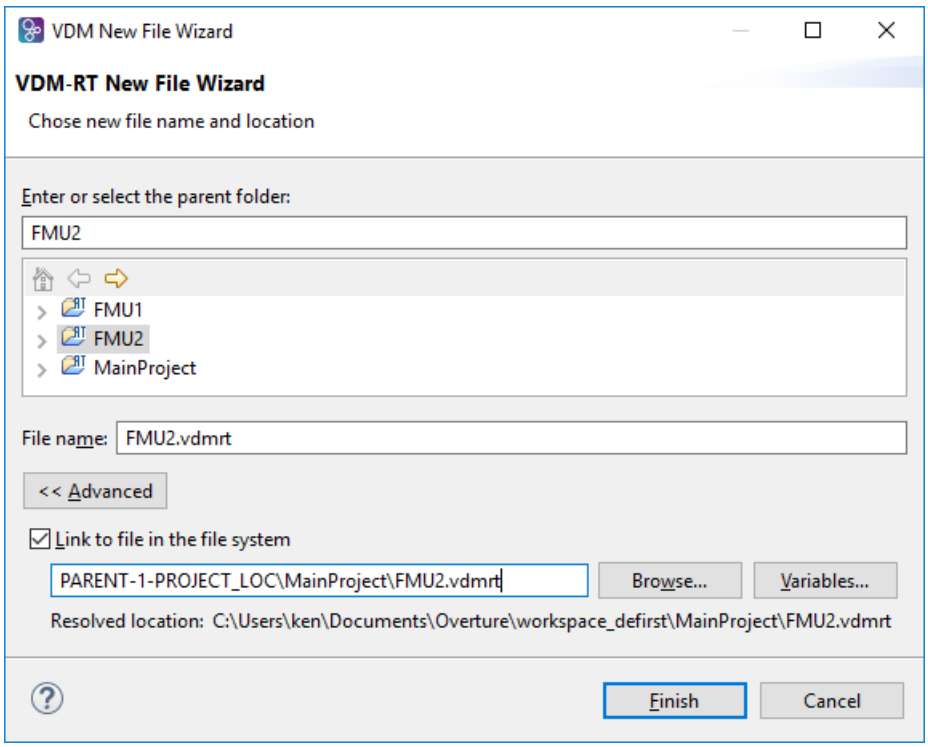


Figure 29: Linking files in the *New > Empty VDM-RT File* dialogue.

main project, then this class can be linked into each of the FMU projects in the same way.

From these individual project, FMUs can be exported and co-simulated within the INTO-CPS tool. These FMUs can then be replaced as higher-fidelity versions become available, however they can be retained and used for regression and integration testing by using different multi-model configurations for each combination.

6.5 Modelling Networks with VDM in Multi-models

In this section, we address the problem of modelling networked controllers in multi-models, presenting a solution using VDM. 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. For example, consider the case of five controllers depicted in Figure 31. In order to connect each con-

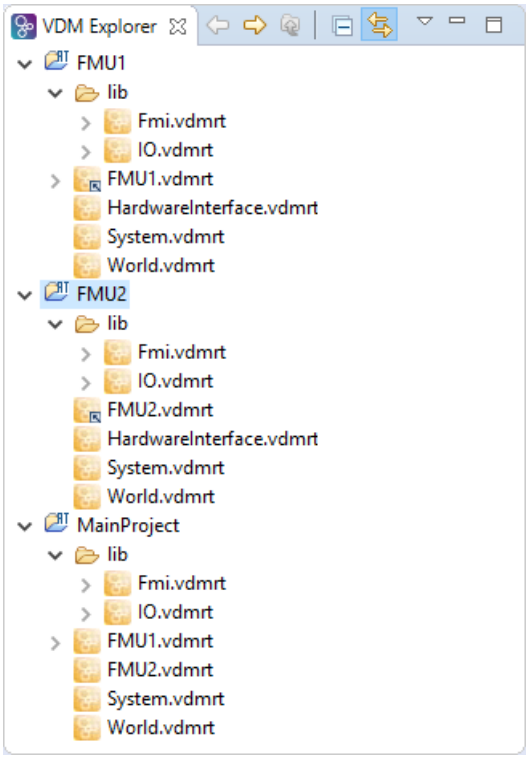


Figure 30: Project structure of an Overture workspace showing a main project and two projects used for generating FMUs from linked class files.

troller together, 20 connections are needed (i.e. for a complete bidirected graph). Even with automatic generation of multi-model configurations, this is in general not a feasible solution.

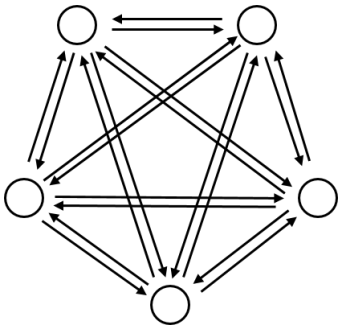


Figure 31: Topology of five controllers connected to each other

We suggest employing a pattern described initially as part of the Crescendo technology [FLV14], in which a representation of an abstract communications medium called the 'ether' is introduced. 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. For five controllers, only 10 connections are needed, as shown in Figure 32.

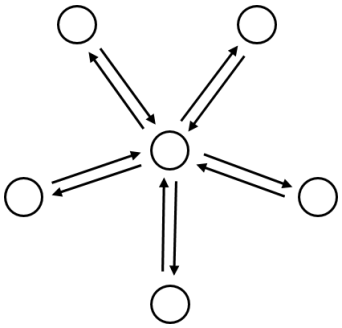


Figure 32: Topology of five controllers connected via a shared medium

In the remainder of this section, 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. An example multi-model, called *Case Study: Ether*, is available from the INTO-CPS Application. It is also described in the Examples Compendium, Deliverable D3.6 [MGP+17].

6.5.1 Representing VDM Values as Strings

Connections between FMUs are typically numerical or Boolean types. This works well for modelling of discrete-time (DT) controllers and continuous-time (CT) physical models, however one of the advantages of VDM is the ability to represent more complex data types that better fit the abstractions of supervisory control. Therefore, in a multi-modelling setting, it is advantageous if VDM controllers can communicate with each other using data types that are not part of the FMI specification.

This can be achieved by passing strings between VDM FMUs (which are now supported by the Overture FMU export plug-in) and the `VDMUtil` standard library included in Overture, which can convert VDM types to their string representations and back again.

The `VDMUtil` library provides a (polymorphic) function called `val2seq_of_char`, that converts a VDM type to a string. It is necessary to tell the function what type to expect as a parameter in square brackets. For example, in the following listing, a 2-tuple is passed to the function, which will produce the output `"mk_(2.4, 1.5)"`:

```
VDMUtil`val2seq_of_char[real*real](mk_(2.4, 1.5))
```

The above can be used when sending messages as strings. In the model receiving message, the inverse function `seq_of_char2val` can be used. This function returns two values, a Boolean value indicating if the conversion was successful, and the value that was received:

```
let mk_(b,v) = VDMUtil`val2seq_of_char[real*real](msg) in
  if b then ...
```

In the first few steps of co-simulation, empty or invalid strings are often passed as values, so it is necessary to check if the conversion was successful (as in the above listing) before using the value.

Note that currently (as of Overture 2.4.0), the `VDMUtil` library is called in the default scope, meaning that it does not know about custom types defined in the model. Therefore, it is recommended to pack values in a tuple (as in the above example) for message passing, then convert to and from any custom types in the sending and receiving models.

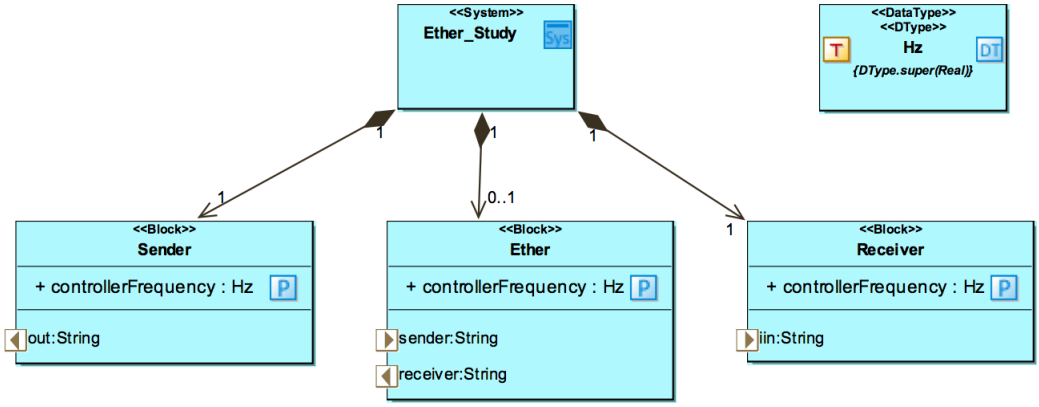


Figure 33: *Case Study: Ether* example

6.5.2 Using the Ether FMU

By encoding VDM values as strings, it is possible to define a simple broadcast ether that receives message strings on its input channel(s) and sends them to its output channel(s). As a concrete example, we consider the *Case Study: Ether* (see Deliverable D3.5 [PGP+16]), which contains a `Sender`, a `Receiver` and an `Ether`, as depicted in Figure 33. In this example, the three FMUs have the following roles:

Sender Generates random 3-tuple messages of type `real * real * real`, encodes them as strings using the `VDMUtil` library and puts them on its output port.

Receiver Receives strings on its input port and tries to convert them to single messages of type `real * real * real` or to a sequence of messages of type of type `seq of (real * real * real)`.

Ether Has an input port and output port, each assigned a unique identifier, i.e. as a `map Id to StringPort`. It also has a mapping of input to output ports as a set of pairs: `set of (Id * Id)`. It has a list that holds messages for each output destination, because multiple messages might arrive for one destination. It gathers messages from each input and passes them to the outputs defined in the above mapping.

In this simple example, the sender and receiver are asymmetrical, but in more complicated examples controllers can be both senders and receivers by implementing both of the behaviours described above.

The *Case Study: Ether* example contains two multi-models that allow the

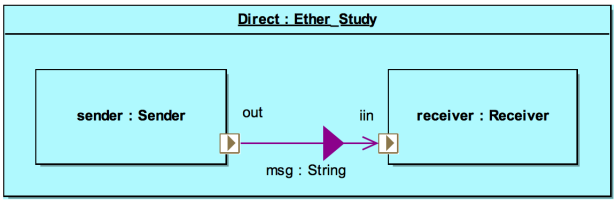


Figure 34: Connection diagram of the *Direct* multi-model in the *Case Study: Ether* example

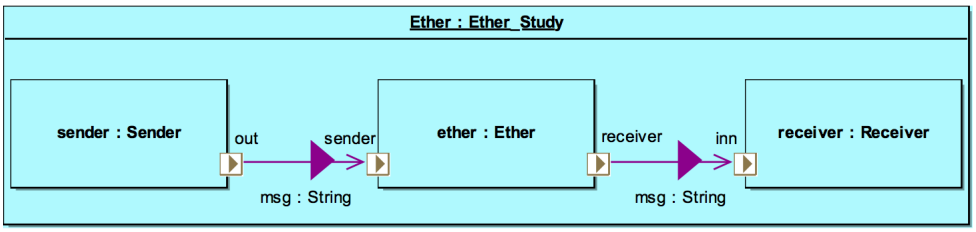


Figure 35: Connection diagram of the *Ether* multi-model in the *Case Study: Ether* example

sender and receiver to be connected directly (connection diagram shown in Figure 34), or to be connected via the ether (connection diagram shown in Figure 35). The description in the Examples Compendium, Deliverable D3.5 [PGP+16], explains how to run the two different multi-models. This approach shows that the use of string ports and the `VDMUtil` library can be useful even without the ether for message passing between controllers in simple topologies.

For the sender, this connection is transparent, it does not care whether it is connected to the ether or not. For the receiver, in the direct connection it will receive single messages, whereas when receiving from the ether it will receive a list of messages (even for a single value). So the receiver is able to deduce when it is directly connected or connected via the ether.

The ether defined in this example is intended to be generic enough that it can be used in other case studies that need a simple broadcast ether without guarantees of delivery. To use it, you can:

1. Import the *Ether* model from the *case-study_ether/Models* directory into Overture;
2. Update the `HardwareInterface`[19] class to provide input and/or output ports for all controllers that will be connected to the ether.

[19] A class that provides annotated definitions of the ports for a VDM FMU.

3. Update the `System` class to assign identifiers to all input and output ports; and
4. Update the set of identifier pairs that define connections.

6.5.3 Consequences of Using the Ether

The ether as presented above is fairly basic. In each update cycle, it passes values from its input variables to their respective output variables. This essentially replicates the shared variable style of direct FMU-FMU connections, which means that the relative update speeds of the FMUs may lead to the following:

Values may be delayed The introduction of an intermediate FMU means that an extra update cycle is required to pass values from sender to ether and ether to receiver. This may delay messages unless the ether updates at least twice as fast as the receiver.

Values may not be read If a value is not read by the receiver before it is changed, then that value is lost.

Values may be read more than once If a value is not changed by the sender before the receiver updates, then the value is read twice. In the simple ether, the receiver cannot distinguish an old message from a new message with the same values.

In the Examples Compendium, Deliverable D3.5 [PGP+16], the *Case Study: Ether* example is described along with some suggested experiments to see the effects of the above examples by changing the controller frequency parameters of the sender, ether and receiver. In the final part of this section we outline ways to overcome such problems if it is necessary to guarantee that messages arrive and are read during a co-simulation.

6.5.4 Modelling True Message Passing and Quality of Service

The key to achieving a true message-passing is to overcome the problem of distinguishing old messages from new messages with the same values. This can be done by attaching a unique identifier to each message, which could be, for example, an identifier of the sender plus a message number:

```
instance variables

id: seq of char := "a";
seqn: nat1 := 1;

...

VDMUtil`val2seq_of_char[seq of char*real*real](
  mk_(id ^ [seqn], 2.4, 1.5));
seqn := seqn + 1
```

The advantage of assigning an identifier to each controller is that messages could also contain destination addresses, instead of the broadcast model presented above. In order to achieve these, some changes are needed to allow for acknowledging receipt of messages. Controllers should:

1. Send a queue of messages on their output channel along with message identifiers of (recently received) messages;
2. Expect to receive a queue of messages along with message identifiers of successfully sent messages; and
3. Senders should remove messages from their output queue once their receipt has been acknowledged.

The `Ether` class must be extended to:

1. Inspect the message identifier (and destination if required) using `VDMUtil`;
2. Pass message identifiers back to senders to acknowledge receipts; and
3. Listen for message identifiers from receivers to know when to remove messages from the queue.

A dedicated channel for acknowledging messages could also be introduced, which would simplify the above. Therefore, each controller would have four connections to the ether: send and acknowledge, receive and acknowledge, as depicted in Figure 36.

The advantages of guaranteed message delivery as described here are that realistic and faulty behaviour of the communication medium can be studied. An ether can be produced that provides poorer quality of service (delay, loss, repetition, reordering). These behaviours could be parameterised and explored using DSE (see Chapter **??**). By controlling for problems introduced by the nature of co-simulation, any reduction in performance of the multi-model can be attributed to the realistic behaviour introduced intentionally into the model of communications.

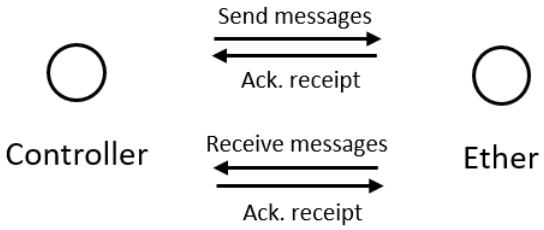


Figure 36: Topology of controller to *Ether* connection with dedicated channels for messages and acknowledgements

6.6 Design Space Exploration

In this section, we outline guidelines for DSE over co-models of CPSs that: (a) support decision management by helping engineers to articulate clearly the parameters, objectives and metrics of a DSE analysis (Section 6.6.1); and (b) enable the tuning of DSE methods for given domains and systems of interest (Section 6.6.2).

6.6.1 Guidelines for Designing DSE in SysML

Rationale Designing DSE experiments can be complex and tied closely to the multi-model being analysed. The definitions guiding the DSE scripts should not just appear with no meaningful links to the any other artefacts in the INTO-CPS Tool chain. There are two main reasons for this, firstly there is no traceability back to the requirements from which we might understand why the various objectives (measures) were being evaluated or why they were included in the ranking definition. Secondly, if DSE configurations are created manually for each new DSE experiment it is easy to imagine that the DSE analysis and ranking might not be consistent among the experiments.

Engineers need, therefore, to be able to model at an early stage of design how the experiments relate to the model architecture, and where possible trace from requirements to the analysis experiments. Here we describe the first step towards this vision: a SysML profile for modelling DSE experiments. The profile comprises five diagrams for defining *parameters*, *objectives* and *rankings*.

We take the same approach to defining the SysML profile for DSE as that used to define the INTO-SysML profile. A metamodel is defined (see Deliver-

able D3.2b [FGPP16]) and the collection of profile diagrams that implement this metamodel are defined in Deliverable D4.2c [BQ16].

In this section, we present an illustrative example of the use of the DSE-SysML profile – from requirements engineering through defining parameters and objectives in the DSE-SysML profile to the final DSE JSON configuration files. We present result of the execution of DSE for the defined configuration.

As an example, we use the line follower robot pilot study. More details can be found in Deliverable D3.5 [PGP+16].

Requirements We propose the use of a subset of the SoS-ACRE method detailed in Chapter **??** (as this section concentrates on the application of the DSE-SysML profile, we don't consider the full SoS-ACRE process). In the Requirements Definition View in Figure 37, the following five requirements are defined:

1. The robot shall have a minimal cross track error
2. The cross track error shall never exceed **X** mm
3. The robot shall maximise its average speed
4. The robot shall have a minimum average speed of **X** ms^{-1}
5. The robot sensor positions may be altered to achieve global goals

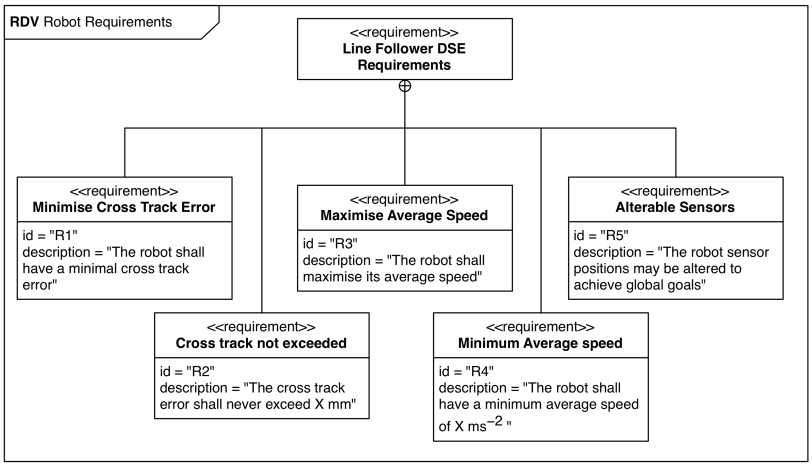


Figure 37: Subset of the Requirements Definition View for requirements of the Line Following Robot

Objectives from Requirements Based upon the requirements above, we define two objectives: the calculation of deviation from a desired path, and the speed of the robot.

Deviation The deviation from a desired path, referred to as the cross track error, is the distance the robot moves from the line of the map, as shown in Figure 38.

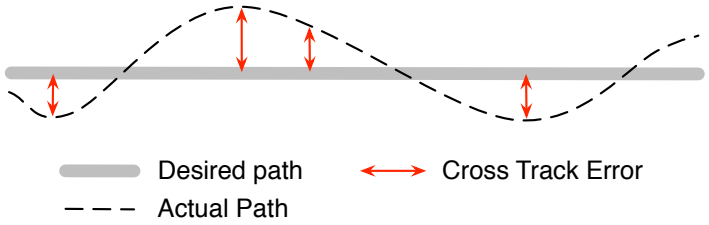


Figure 38: Cross track error at various points for a robot trying to follow a desired line

To compute cross track error we need some model of the desired path to be followed and the actual path taken by the robot. Each point on the actual path is compared with the model of the desired path to find its distance from the closest point, this becomes the cross track error. If the desired path is modelled as a series of points, then it may be necessary to find shortest distance to the line between the two closest points.

Speed The speed may be measured in several ways depending on what data is logged by the COE and what we really mean by speed, indicated in Figure 39.

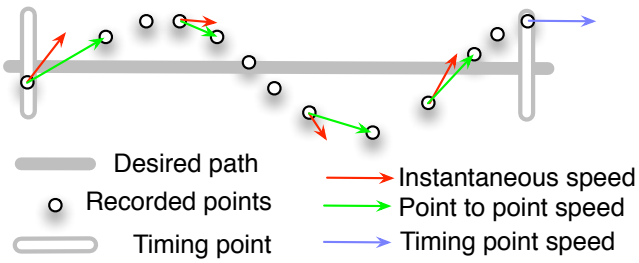


Figure 39: Cross track error at various points for a robot trying to follow a desired line

Inside the CT model there is a bond graph flow variable that represents the forwards motion of the robot. This variable is not currently logged by the COE but it could be and this would result in snapshots of the robot speed being taken when simulation models synchronise. In this example, we take the view that speed is referring to the time taken to complete a lap.

SysML Representation of Parameters, Objectives and Ranking We next consider the use of the upcoming DSE profile to define the DSE parameters, objectives and desired ranking function. In the following SysML diagrams, we explicitly refer to model elements as defined in the architectural model of the line follower study, presented in Deliverable D3.5 [PGP+16].

Parameters In the requirements defined above, we see that the position of the line follower sensors may be varied. In real requirements, we may elicit the possible variables allowed. Figure 40 is a DSE Parameter Definition Diagram and defines four parameters required: *S1_X*, *S1_Y*, *S2_X* and *S2_Y*, each a set of real numbers. The DSE experiment in this example is called *DSE_Example*.

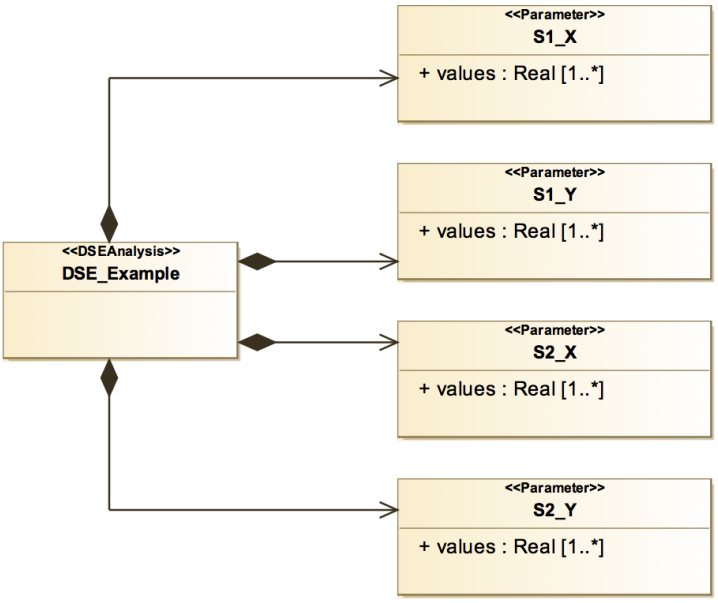


Figure 40: DSE-SysML Parameter Definition Diagram of Line Following Robot example

Figure 41 identifies the architectural model elements themselves (the `lf_position_x` and `lf_position_y` parameters of *sensor1* and *sensor2*) and the possible

values each may have (for example the `lf_position_x` parameter of *sensor1* may be either 0.01 or 0.03). The diagram (or collection of diagrams if there is a large number of design parameters) should record all parameters for the experiment.

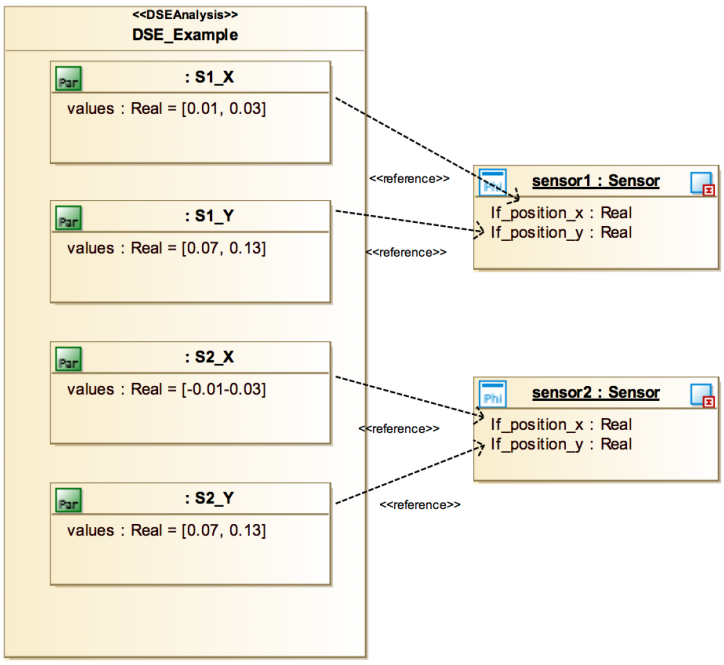


Figure 41: DSE-SysML Parameter Connection Diagram of Line Following Robot example

Objectives The objectives follow from the requirements as mentioned above. Figure 42 shows the DSE Objectives Definition Diagram with four objectives: *meanSpeed*, *lapTime*, *maxCrossTrackError* and *meanCrossTrackError*. Each have a collection of inputs – defined either as constants (e.g. parameter *p1* of *meanSpeed*), or to be obtained for the multi-model.

The objective definitions are realised in Figure 43. The *meanSpeed* requires the step-size of the simulation (this is obtained from the co-simulation results, rather than defined here) and the `robot_x` and `robot_y` position of the robot body. The *lapTime* objective requires the time at each simulation step (again, obtained directly from the co-simulation output), the `robot_x` and `robot_y` position of the robot body and the name of the map. Both the *maxCrossTrackError* and *meanCrossTrackError* objectives require only the `robot_x` and `robot_y` position of the robot body.

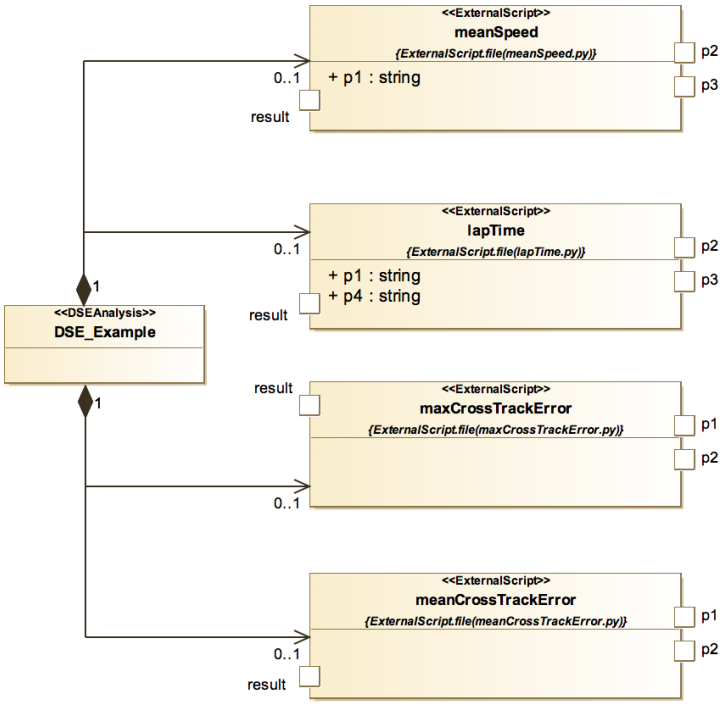


Figure 42: DSE-SysML Objective Definition Diagram of Line Following Robot example

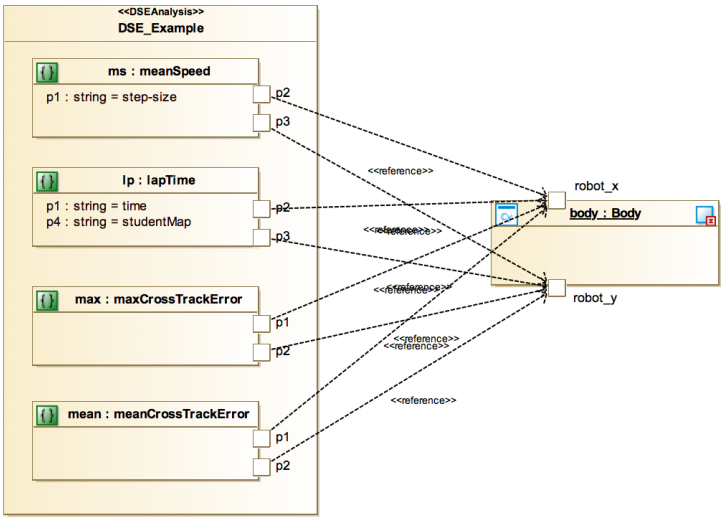


Figure 43: DSE-SysML Connection Objective Diagram of Line Following Robot example

Ranking Finally, the DSE Ranking Diagram in Figure 44 defines the ranking to be used in the experiment. This diagram states that the experiment uses the Pareto method, and is a 2-value Pareto referring to the *lapTime* and *meanCrossTrackError* objectives.

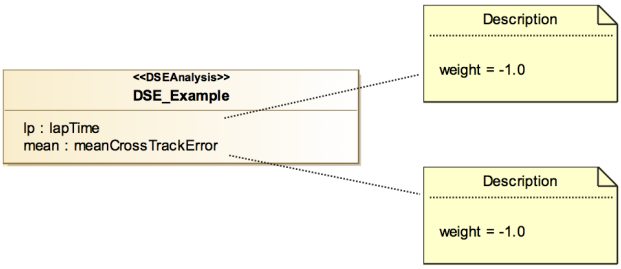


Figure 44: Example DSE-SysML Ranking Diagram of Line Following Robot example

DSE script These diagrams may then be translated to the JSON config format required by the DSE tool. The export of the configuration is performed in the Modelio tool and the subsequent movement of the resulting configuration file is performed in the INTO-CPS application (see the INTO-CPS User Manual, Deliverable D4.3a [BLL+17] for more details). Figure 45 shows the corresponding DSE configuration file for the line follower experiments. Note that where we refer to model elements of the architecture (such as model parameters), we now use the same conventions used in the co-simulation orchestration engine configuration.

DSE results DSE is performed in the DSE tool (again, see the INTO-CPS User Manual, Deliverable D4.3a [BLL+17] for more detail) 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.

Figure 46 shows an example of the DSE results from the line follower robot

where the lap time and mean cross track error were the objectives to optimise. These results contain two representations of the data, a graph plotting the objective values for each design, with the Pareto front of optimal trade-offs between the key objectives highlighted, here in blue. The second part of the results presents the data is tables, indexed by the ranking position of each result. This permits the user to determine the precise values for both the measured objectives and also the design parameters used to obtain that result.

6.6.2 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. The initial approach we took was to implement an algorithm to exhaustively search the design space, and evaluate and rank each design. Whilst this approach is acceptable on small-scale studies, this quickly becomes infeasible as the design space grows. For example, varying n parameters with m alternative values produces a design space of m^n alternatives. In the remainder of this paper, we present an alternative approach to exploring the design space in order to provide guidance for CPS engineers on how to design the exploration of designs for different classes of problems.

A Genetic Algorithm for DSE 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. Figure 47 represents the structure of a genetic algorithm used for DSE. Several activities are reused from exhaustive DSE: simulation; evaluation of objectives; rank simulated designs; and generate results. The remaining activities are specific to the genetic approach and are detailed in this section.

Generating initial population: Two methods for generating an initial population of designs are supported: randomly, or uniformly across the design space. Generating an initial design set which is distributed uniformly could allow parts of the design space to be explored that would otherwise not be explored with a random initial set. This could give us greater confidence that the optimal designs found by the genetic algorithm are consistent with the optimal designs of the total design space.

Selecting parents: Two options for parent selection are supported: random

and distributed. Random selection means that two parents are chosen randomly from the non-dominated set (NDS). There is also a chance for parents to be selected which are not in the NDS, potentially allowing different parts of the design space to be explored due to a greater variety of children being produced.

An intelligent approach involves calculating the distribution of each design's objectives from other designs in the NDS. One of the parents chosen is the design with the greatest distribution, enabling us to explore another part of the design space which may contain other optimal designs. Picking a parent that has the least distribution suggests that this parent is close to other optimal designs, meaning that perhaps it is likelier to produce optimal designs.

Figure 48 shows the fitness roulette by which how much a design solution in Figure 49 satisfies the requirements. It can be seen that there exists a relationship where the greater the fitness value a design has, the more likely it is to be selected as a parent. The probability P of design d being selected as a parent can be calculated by:

Breeding children: After the parents are selected, the algorithm creates two new children using a process of crossover. Figure 50 shows this process. Mutation could also occur, where a randomly chosen parameter's value is replaced by another value defined in the initial DSE configuration, producing new designs to explore other parts of the design space.

Checking current progress: Progression is determined by the change in the NDS on each iteration. It is possible to tune the number of iterations without progress before termination.

Measuring Effectiveness To provide guidance on selection and tuning of a specific algorithm to a DSE situation it is necessary that there is a means for experimenting with the algorithm parameters and also means for evaluating the resulting performance. To this end an experiment was devised that supports exploration of these parameters using a range of design spaces as the subject. The experiment is based upon generating a ground truth for a set of design spaces such that the composition of each Pareto front is known and we may assess the cost and accuracy of the genetic algorithm's attempt to reach it. A limiting factor for these design spaces is that they must be exhaustively searched and so there are current four of these all based upon the line follow robot: an 81-point and a 625-point design space where the sensor positions are varied and a 216-point and 891-point design spaces

where the controller parameters are varied. There are three measures applied to each result that target the tension between trading off the cost of running a DSE against the accuracy of the result

Cost: The simplest of the measures is the cost of the performing the search and here it is measured by the number of simulations performed to reach a result. For the purposes of comparison across the different design spaces, this cost is represented as a proportion of the total number of designs

$$cost = \frac{|Simulations\ \ Run|}{|Design\ \ Space|}$$

Accuracy: The ground truth exhaustive experiments provide us with the Pareto Front for that design space and each DSE experiment returns a non-dominated set of best designs found. Here the accuracy measure considers how many of the designs in the genetic non-dominated set are actually the best designs possible. It is measured by finding the proportion of points in the genetic NDS that are also found in the ground truth Pareto front.

$$accuracy = \frac{|GeneticNDS \cap ExhaustiveNDS|}{|GeneticNDS|}$$

Generational Distance: The accuracy measure tells us something about the points in the genetically found NDS that are also found in the exhaustive NDS (Van Veldhuizen & Lamont, 2000). The generational distance gives us a figure indicating the total distance between the genetic NDS and the exhaustive NDS. It is calculated by computing the sum of the distance between each point in the genetic NDS and its closest point in the exhaustive NDS and dividing this by the total number of points.

$$generational\ distance = \frac{\sqrt{(\sum_{i=1}^{n} d_i^2)}}{n}$$

Genetic DSE Experiments and Results The DSE experiments involved varying three parameters of the genetic algorithm and repeating each set of parameters with each design space five times. The parameters of the genetic algorithm varied were:

Initial population size: The initial population size took one of three values. All design spaces were tested using an initial population of 10 designs, they were also tested with initial populations equal to 10% of the design space and 25% of the design space. These are represented on the left hand graphs by the 10, 10% and 25% lines.

Progress check conditions: The number of rounds the genetic algorithm would continue if there was no progress observed was tested with three values, 1, 5 and 10. These are represented on the right hand graphs with the 1, 5 and 10 lines.

Algorithm options: There are two variants of the genetic algorithm, phase 1 with random initial population and random parent selection, and phase 3 which give an initial population distributed over the design space and where parent selection is weighted to favour diverse parents. The phase one experiments are on the left hand side of the graphs, with points labelled '<design space size>-p1' while the phase three experiments are on the right labelled '<design space size>-p3'.

The results of the simulations are shown in graph form below. Each point graphed is the averaged result of the five runs of each set of parameters. Figure 51, shows the graphs of cost of running the DSEs. Encouragingly there is a slight trend of the cost of DSE reducing as a proportion of the design space as the design space size increases. As expected the cost was greater with larger initial populations but the cost did not vary when changing the progress check condition as much as expected.

Figure 52 shows the graphs of DSE accuracy. There is again a slight downward trend as design space size increases, meaning that there is a slight increase in the number of points in the genetic NDS that are not truly optimal. As expected the larger initial population generally resulted in more accurate NDS, this was also true of using the largest value for the progress check condition.

Figure 53 presents the generational distance results. Here we find that the results are generally low, with the exception of the 891-point design space which is significantly worse, the reason for this is still to be determined. The largest initial design space resulted in the lowest (best) values as did using a progress check condition value of five.

Selecting Approaches based on Design Space The choice of which search algorithm to use when performing DSE is dependant on one factor, and that is a comparison of the 'simulations required' compared to the 'simulation budget', before describing what to do with the comparison, it is first necessary to explain those terms.

The number of 'simulations required' is dependant on the number of different design alternatives that the DSE is supposed explore, but there also other factors, specifically repetitions and scenarios. The 'design' part of the number

of simulations required is determined by multiplying together the number of values each parameter may adopt, since this gives the number of unique designs. If the DSE configuration includes parameter constraints then the number of valid designs will be lower since some parameter combinations will fail to meet the constraints. For example, A line follower robot with two sensors, where each sensor has three possible x position values and three possible y position values, would have a design space size of 81, however if those parameters are constrained so designs must be symmetrical, i.e. the x and y values of each sensor must be identical, then the design space only has nine points.

If a simulation model contains random elements, such a noisy inputs to sensors or models of dropped messages on a network, then this leads to the simulation results being non-deterministic. In this case, it will be necessary to perform repeated simulations of the same design with the same starting conditions to account for the random variation.

'Scenarios' refer to the environment around the actual system-under-test in the simulation, for example, in the case of a line following robot, the environment could include the map that the robot is to follow along with other factors such as the intensity of the ambient light. If there is a desire to perform simulations under different scenarios, then the number of scenarios must also be taken into account when determining the required number of simulations. The final required number of simulations then is the product of the design space size, the number of repetitions and the number of scenarios.

The simulation budget term refers to the maximum number of simulations that a user may perform as part of DSE experiment. It is a matter for the user to determine the value for this budget, but it could be determined by determining the amount of CPU time allocated to the DSE and dividing it by the time to run a simulation and compute the objective values.

The decision of whether to use the exhaustive search algorithm or a closed loop search can be made by comparing the number of simulations required with the simulation budget. If the simulation budget is greater than or equal to the required number of simulations, then an exhaustive search should be used as this guarantees to find the optimal designs given the design parameter values, if the budget is less than the required number of simulations required then a closed loop approach is needed.

Parameters for a Genetic Search If the decision is made to perform a genetic search, then it is important to note that there are two parameters

that affect how the algorithm behaves and will have an effect on the outcome. The first of these parameters is the initial population size, this defines how many random designs are generated at the start of the search as a seed for the process. A general rule for this initial population size is that it should be 10 times the number of dimensions (parameters) [DGH07], with the caveat that as the number of dimensions increases, this multiplier must also also increase.

The second parameter is the termination condition, or the number generations the algorithm will continue without seeing progress before it terminates. The genetic algorithm measures progress by looking at the designs that make up the non-dominated set of the Pareto analysis. The only way membership of this set can change between two generations is if better designs, according to the objective measures, have been found, so if membership changes then the search is making progress towards finding better designs. It is not unusual for the algorithm to breed new designs that are not better than those currently in the non-dominated set and so to have a generation that does not show progress, but then to make progress in a subsequent generation. Thus, the number of generations without progress parameter is used to relax the termination condition to permit generations without progress without stopping. Increasing this value will increase the probability that the search will not become stuck in some local optima in the results and may progress to find better designs. There is a cost associated with increasing this value since, as the algorithm produces two new designs per generation, there will be two times the number of generations without progress simulations run at the end of the process that do not lead to better results [FGPL17].

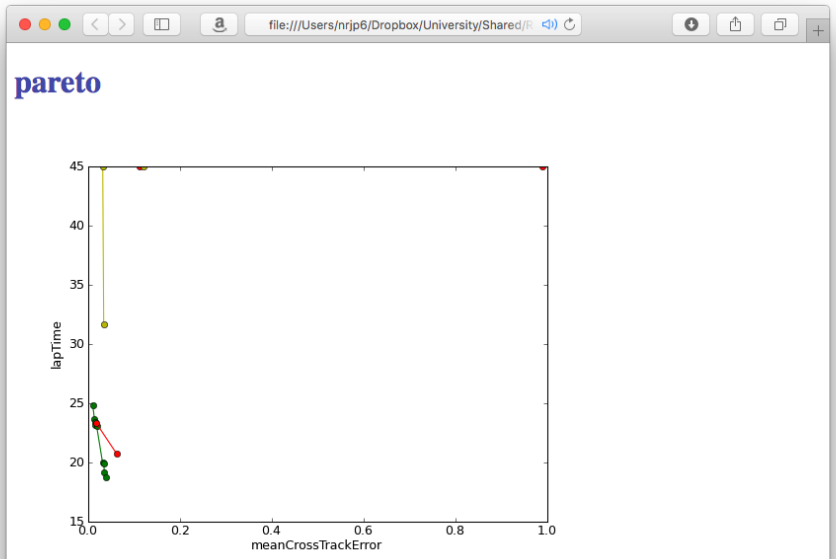
Iterative Exhaustive Search An alternative to the genetic search, which is automated, is to use repeated exhaustive searches to home in on better regions of the design space. In this approach the user would plan to perform multiple DSE experiments, each using some portion of their total simulation budget. 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. Figure 54 shows an example initial search, with the blue dots indicating simulations performed and the green areas giving the best results. The user then divides their remaining

simulation budget among the three green areas of interest, searching each with a higher resolution in an attempt to extract the best results from each, Figure 55.

```
{
    "algorithm": {},
    "objectiveConstraints": {},
    "objectiveDefinitions": {
        "externalScripts": {
            "meanSpeed": {
                "scriptFile": "meanSpeed.py",
                "scriptParameters": {
                    "1": "step-size",
                    "2": "{bodyFMU}.body.robot_x",
                    "3": "{bodyFMU}.body.robot_y"
                    }
                },
            "lapTime": {
                "scriptFile": "lapTime.py",
                "scriptParameters": {
                    "1": "time",
                    "2": "{bodyFMU}.body.robot_x",
                    "3": "{bodyFMU}.body.robot_y",
                    "4": "studentMap"
                    }
                },
            "maxCrossTrackError": {
                "scriptFile": "maxCrosstrackError.py",
                "scriptParameters": {
                    "1": "{bodyFMU}.body.robot_x",
                    "2": "{bodyFMU}.body.robot_y"
                    }
                },
            "meanCrossTrackError": {
                "scriptFile": "meanCrosstrackError.py",
                "scriptParameters": {
                    "1": "{bodyFMU}.body.robot_x",
                    "2": "{bodyFMU}.body.robot_y"
                    }
                }

        },
        "internalFunctions": {}
    },
    "parameters": {
        "{sensor1FMU}.sensor1.lf_position_x": [
            0.01,
            0.03
        ],
        "{sensor1FMU}.sensor1.lf_position_y": [
            0.07,
            0.13
        ],
        "{sensor2FMU}.sensor2.lf_position_x": [
            -0.01,
            -0.03
        ],
        "{sensor2FMU}.sensor2.lf_position_y": [
            0.07,
            0.13
        ]
    },
    "ranking": {
        "pareto": {
            "lapTime": "-",
            "meanCrossTrackError": "-"
        }
    },
    "scenarios": [
        "studentMap"
    ]
}
```


Figure 45: A complete DSE configuration JSON file for the line follower robot example



Rank	maxCrossTrackError	meanCrossTrackError	lapTime	meanSpeed	controller.backwardRotate	contr
1	0.11212158579	0.00972601327368	24.87	0.0534613909175	0.1	0.5
1	0.11212158579	0.0135839416528	23.66	0.0535217189374	0.1	0.5
1	0.11212158579	0.0149825071391	23.41	0.0530730992742	0.1	0.5
1	0.11212158579	0.0158818400743	23.14	0.0536836889558	0.1	0.5
1	0.11212158579	0.0183768999898	23.05	0.0531333542011	0.1	0.5
1	0.11212158579	0.0312131106813	20.0	0.0530649424606	0.1	0.5
1	0.11212158579	0.0338998655889	19.94	0.0531259947624	0.1	0.5
1	0.11212158579	0.0351908945857	19.13	0.0535092206699	0.1	0.5
1	0.11212158579	0.0383207684581	18.78	0.0527153104611	0.1	0.5
2	0.11212158579	0.0170376693167	23.35	0.0534644763037	0.1	0.5
2	0.123687486858	0.0616770378122	20.75	0.0535117177187	0.1	0.5
3	0.122323135307	0.0315978498604	45	0.0533721883477	0.1	0.5
3	0.129135864318	0.0337310165873	31.65	0.0534171637801	0.1	0.5

Figure 46: DSE results

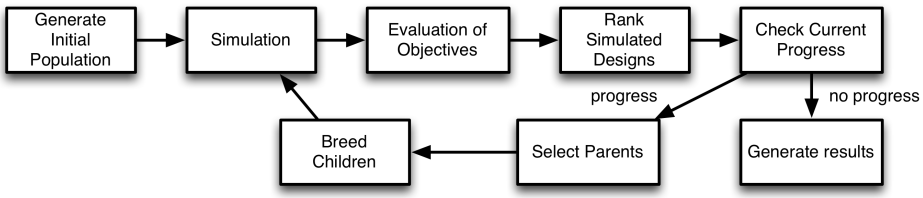


Figure 47: High-level process for DSE Genetic Algorithm

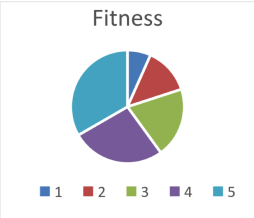


Figure 48: Example fitness roulette

Solution	Fitness
1	25
2	5
3	40
4	10
5	20

Figure 49: Genetic Algorithm fitness selection

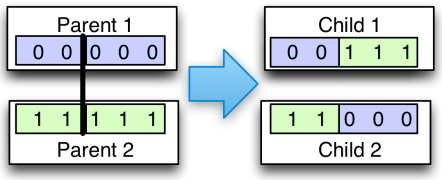


Figure 50: Depiction of genetic crossover

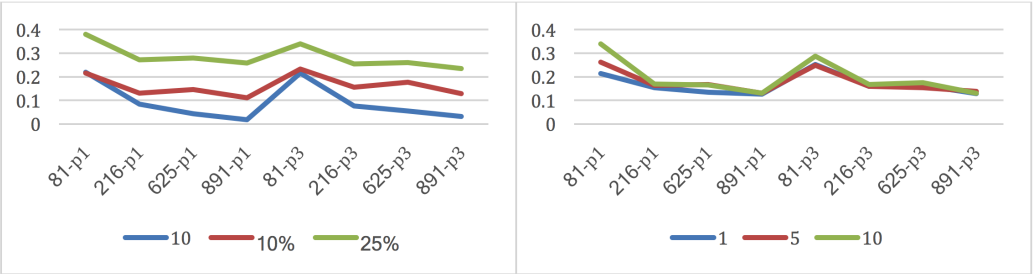


Figure 51: Cost of DSE, number of simulations run as proportion of the total design space.

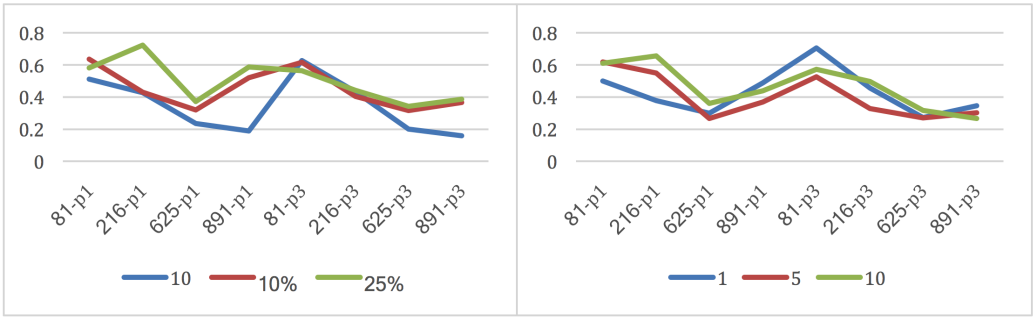


Figure 52: Accuracy of DSE, proportion of genetic NDS found in exhaustive NDS.

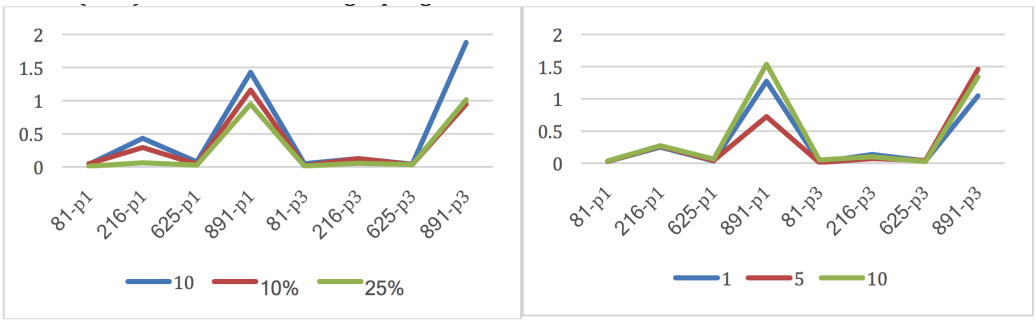


Figure 53: Gap between Genetic NDS and Exhaustive NDS. The vertical axis has no meaningful units, a smaller number is better.

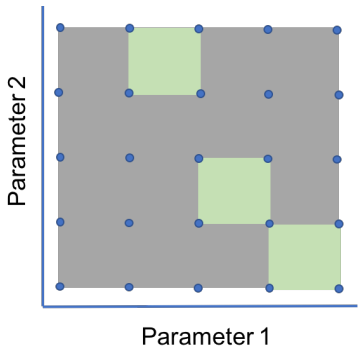


Figure 54: Step 1 of an iterative search. The best results being found in the green regions

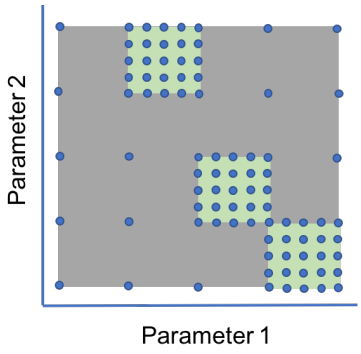


Figure 55: Step 2 of an iterative search. The green regions are searched with a higher resolution to find the best results

7 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 C). 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 56, where the red boxes indicate the different sections of this chapter.

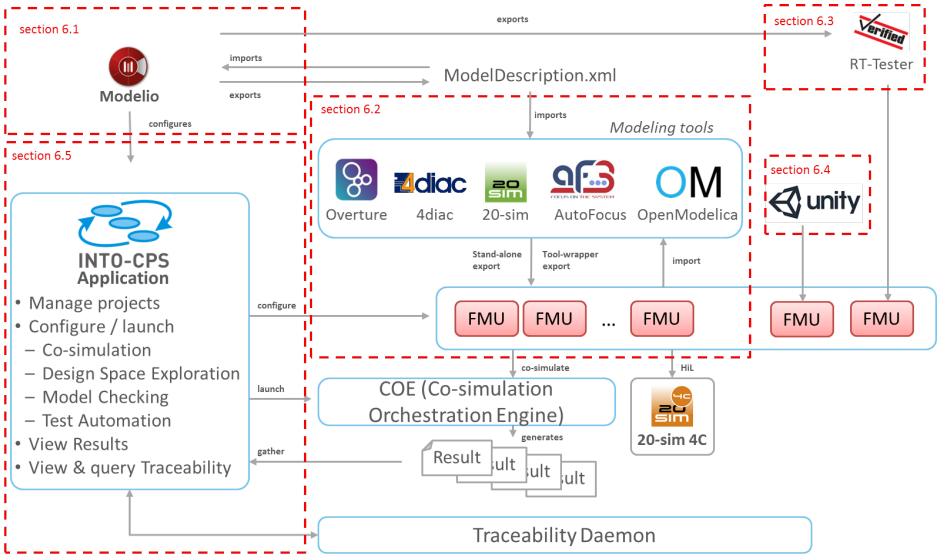


Figure 56: Overview of the different tools and their arrangement in a tool-chain.

7.1 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. [FGP17a]), 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 [20]. 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 7.2.

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 subsection C.1.

7.2 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-

[20] 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 subsection C.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 subsection C.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 subsection C.2.

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

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

ABS is a language for Abstract 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, 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

7.3 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 subsection C.5.

7.3.1 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].

7.3.2 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].

7.4 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 7.1) 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.

7.5 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 7.3). Furthermore, traceability data can be viewed in the Application, either in an expert view, using the Neo4J visualisation[21], or by pre-configured queries.

[21] `https://neo4j.com/`

8 The INTO-CPS Industrial Case Studies

The industrial domains benefited from the INTO-CPS technologies in different ways, as expected due to their different multidisciplinary engineering activities. Nonetheless, certain aspects of the tool chain were of benefit to all industrial partners. We thus argue that these are the most broadly applicable benefits of INTO-CPS and the ones that can most successfully be transferred to other domains (fig. 57). Modelling and simulation is heavily used in industry today, to evaluate performance and robustness of the system with regards to requirements. Aspects such as simulation speed, and model fidelity are of high importance. Going beyond model-based design and single model simulation, co-simulation enables the analysis of physical interactions of systems that were previously not captured, due to the different domains at which the physics were modelled. Multi-physics analysis enables early analysis and detection of issues that were only uncovered at the physical prototyping stage, thus saving time and money.

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

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

Figure 57: INTO-CPS Industrial Case studies

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

8.1 The Automotive Case Study

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

The case study can be considered a CPS because it contains local intelligence and autonomy in the vehicle. This is assisted by information about its environment typically derived from a cloud context (here, information on weather and traffic / route) and the logic depends upon the physical dynamics of the electric vehicle (Fig. 58).

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

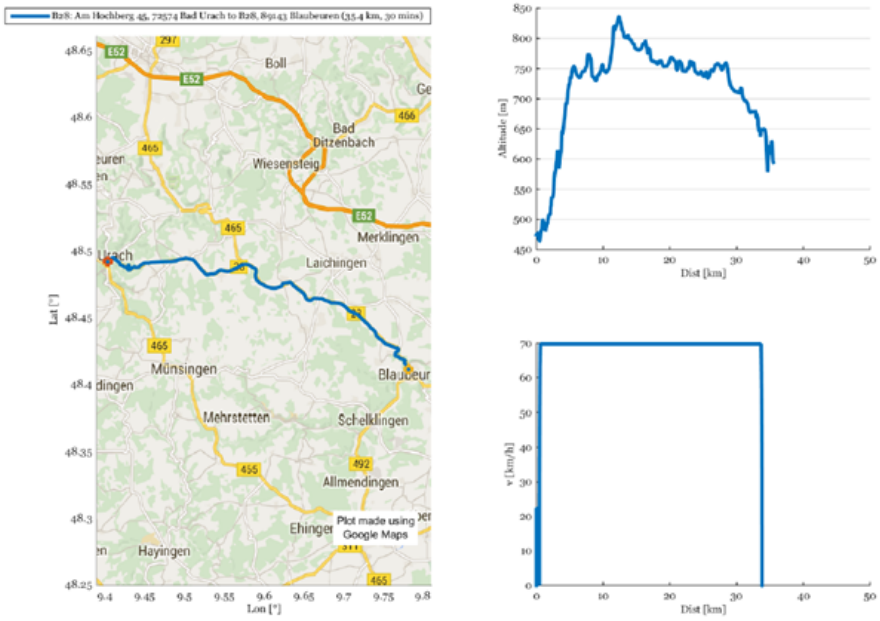


Figure 58: Automotive case study using INTO-CPS: Velocity and altitude profile for a route of 35 km in the vicinity of Stuttgart. The route consists here of a country road only, and thus the velocity is stable at 70 km/h, while the altitude varies between 450m and 850m above sea level.

real hardware (here, as Raspberry Pi) and simulated with the remainder of the system. Since the case study was developed as part of the INTO-CPS project [FGL+15, LFW+16a, LTL+16, LFW+16b, LFW+17], one aim was to evaluate the INTO-CPS tools and methods. Here this is in particular the Co-simulation orchestration engine (COE), which is a FMI 2.0 compliant master algorithm that allows coupling of continuous-time (CT) and discrete event (DE) models in a Co-Simulation setup [4,5]. furthermore, the system was modelled in SysML, using the CPS-extension of the Modelio tools [BBQS15]. The models themselves were created using Matlab[22], 20-sim[23] [Kle06], C++ and Overture[24] [LBF+10].

[22] https://www.mathworks.com/products/matlab.html
[23] http://www.20sim.com/
[24] http://overturetool.org/

8.2 The Agricultural Case Study

The focus of the case study is the development of the agricultural field robot, Robotti using the INTO-CPS technology. Models of the machine dynamics and controllers have been developed using the baseline tools and Co-simulated using the COE. The DSE feature is applied in the development and assessment of the steering controller which is crucial to the performance of the robot. The agricultural case study has been extended with an additional industrial application, namely the AI-Mower. The dynamics of the mower is modelled and co-simulated to assess and optimise a new approach for steering the mower.

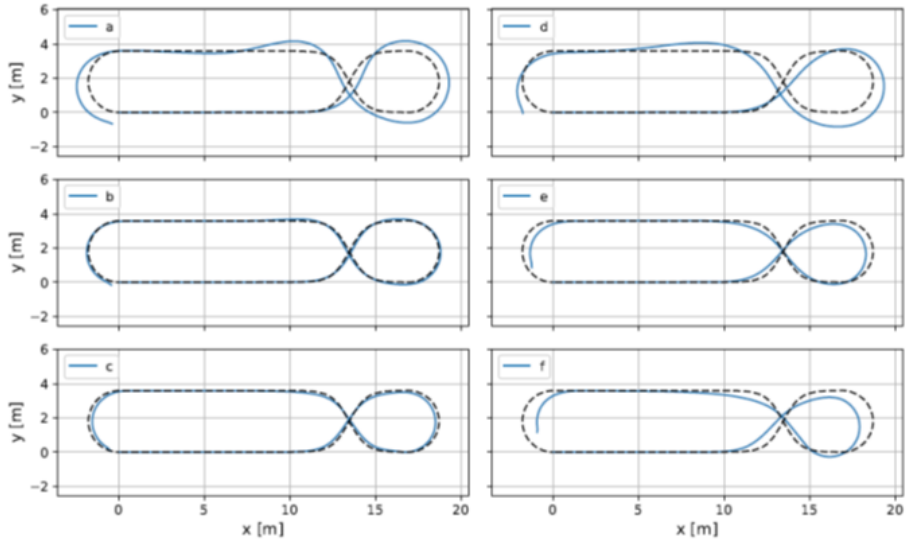


Figure 59: Simulated trajectories of the six controller configurations (a,b,...,f)

Additionally, the 3D FMU feature has been applied to visualise the machine based on the models of the kinematics and dynamics. The details on the mower are kept confidential contrary to the Robotti project which is public. Through the model development using INTO-CPS, the work related to Robotti have been focused towards DSE-based optimisation of the steering controller which is crucial to the performance of the machine. Several scenarios of different steering controller configurations are simulated using the COE and the DSE feature is applied to estimate the optimal controller configuration of the Robotti. The influence of the controller parameters are shown in Fig. 59 where the six simulated trajectories are shown corresponding to six combinations of two controller parameters. The dashed line represents

the desired route and the blue line represents the simulated trajectory of the Robot.

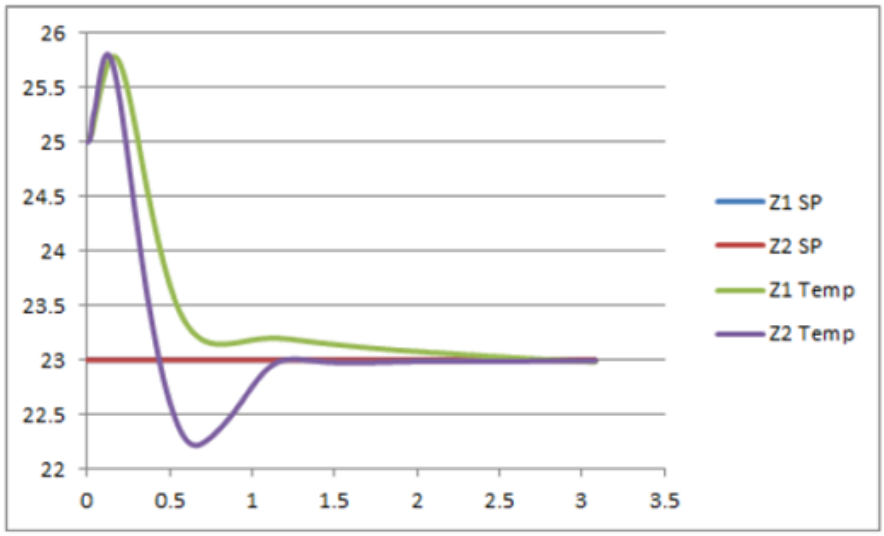
Related Publications can be found at [FLG17, FBG+18].

8.3 The Building Case Study

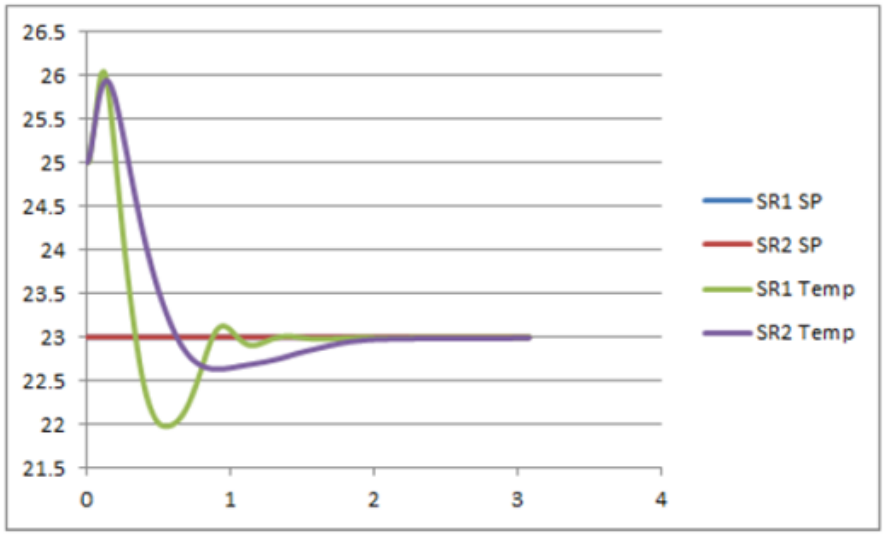
The building case study focuses on modelling and analysis of energy and comfort for Heating, Ventilation and Air Conditioning (HVAC) systems that control the temperature of connected areas inside building premises. The case study models various concepts shown in fig. 60 such as: a) Fan Coil Unit (FCU) and control; b) Supervision and fault detection of FCUs; c) Communication between master-slave FCUs; d) Communication between FCUs and supervisor; e) Air Handling Unit (AHU) and control; f) Chiller load and control; g) Physical rooms and air flow; h) Water and air pipe connections.

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

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



(a) Zone Temperature



(b) Single Rooms Temperature

Figure 60: INTO-CPS Co-Simulation Results for Room temperature in building zones

Related Publications can be found at [FGP+16, CBM+17].

8.4 The Railway Case Study

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

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

Related Publications can be found at [LFWL16]

8.5 The Aerospace Case Study

IDA2- Aerospace - MISSION

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

The case study involved the virtual design and validation of a CPS-based manufacturing system for assembling USB sticks, inspired from Continental's

real manufacturing and testing processes in a production line. It is a representative example of distributed heterogeneous systems in which products, manufacturing resources, orders and infrastructure are all cyber-physical. In this setting, several features (such as asynchronous communication, messages flow, autonomy, self-adaptation, etc.) could be investigated at design time, for example using a collaborative modelling approach. Consequently, the case study offered a balance between being sufficiently simple to be easily followed as a production line example, including generating a tangible output, and at the same time being sufficiently general to allow the study of the co-simulation complexity. Furthermore, by choosing a USB stick, the example opened the (unexplored) possibility of extending the purpose of the study to interactions between generated hardware and generated software solutions in the production line.

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

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

tem.

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

The detailed model of each simulation covered a continuous-time model realised in 20-Sim for the warehouse and robotic arm, a discrete-time model in 4DIAC for the transportation system and test station, and a discrete-time model in Overture for the infrastructure. All units modelled and tested by the heterogeneous co-simulation were then deployed in a demo stand for fine tuning under real-life conditions (Table 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.

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

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

Table 3: Technologies used for different system components

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[25]

[25]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`.

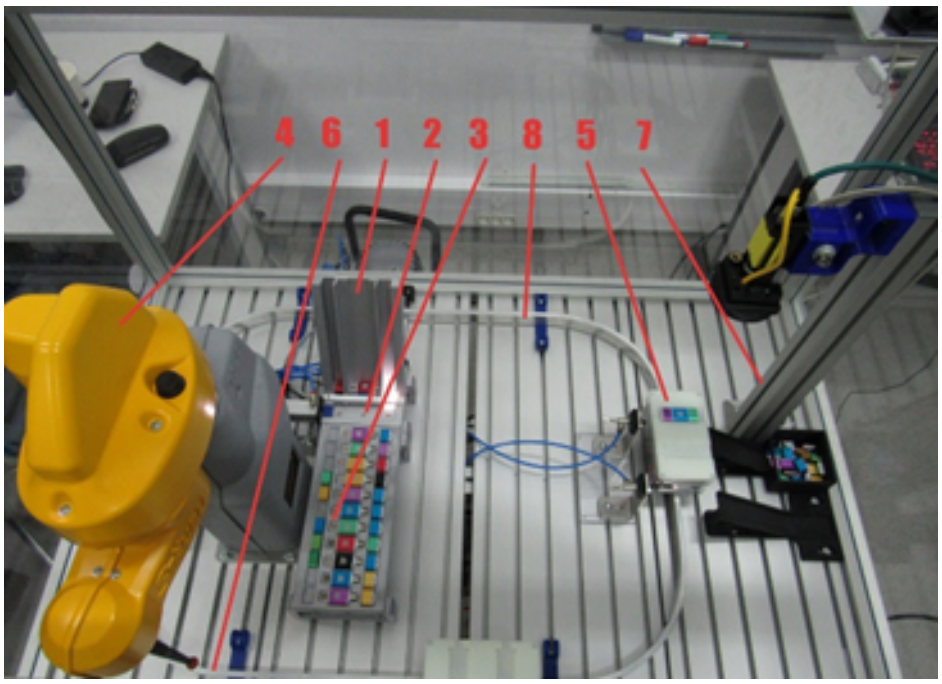


Figure 61: Demo stand for deployment of the co-simulated units, containing: 1) the warehouse stacks; 2) the assembly box at the base of the warehouse stacks; 3) the memory boxes of the warehouse unit; 4) the robotic arm for moving parts around the warehouse; 5) wagons on different locations of the track; 6) the loading station; 7) the test station; 8) the circular track for the wagons..

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

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

These systems require advanced algorithms to control the complexity of the physical dynamics of large engines. MDT is divided into different departments with different responsibilities in the same way as many other large

organisations. In the control department at MDT, control algorithms are created directly in the target software framework with the possibility of performing Software In the Loop (SIL) simulation during development. Models of the physical behaviour are created in other departments of MDT using the tools most suitable for the specific constituent system.

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

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

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.

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

At the European Space Agency (ESA) many systems fall into the CPS category. As reported in [FAVL17], the Mars Rover case study, which was developed in Crescendo, a previous project, had been restricted to the combination of two models: one Discrete Event (DE) model expressed using the Vienna

Development Method (VDM) [FLV08] using the Overture tool [LBF+10] and one Continuous-Time (CT) model expressed using bond graphs [KR68] and the 20-sim tool [Kle06].

Moreover, the CT model would have to be kept confidential. Thus, it was problematic to share this co-model with other parties. As reported in [FAVL17], this could only be circumvented by running the co-simulation over the Internet, with the proprietary constituent CT model running at ESA. While technically feasible, this of course causes many other problems, such as allowing remote access through corporate firewalls, poor simulation performance, etc.

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

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

9 Related Work

Extensive work has been done 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 behavior 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.

Recognizing that co-simulation is not a new concept and that it has been applied in wildly different fields, [GTB+18b] 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 **Claudio: (An extended version is being published at the American Modelica Conference)** 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.

Claudio: Should we survey also co-simulation frameworks, or are the above surveys on the topic enough? I guess that if someone wants to learn about co-simulation, the surveys are a better way to start...

10 Future Directions

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

In the subsections below candidate future directions are proposed.

10.1 Adapting FMUs Easily to Ones Needs

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

10.2 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 7 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+18a] 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.

10.3 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.

10.4 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.

10.5 Increased Support for Dynamic Evolution Scenarios

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

10.6 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].

10.7 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 8.6). 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.

10.8 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.

10.9 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 lear 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.

10.10 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 Reduced Order Models (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+16a] Nuno Amalio, Ana Cavalcanti, Alvaro Miyazawa, Richard Payne, and Jim Woodcock. Foundations of the SysML for CPS modelling. Technical report, INTO-CPS Deliverable, D2.2a, December 2016.

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

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

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

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

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

[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.

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

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

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

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

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

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

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

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

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

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

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

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

[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.

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

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

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

[CFT+17] Ana Cavalcanti, Simon Foster, Bernhard Thiele, Jim Woodcock, and Frank Zeyda. Final Semantics of Modelica. Technical report, INTO-CPS Deliverable, D2.3b, December 2017.

[CFWZ17] Ana Cavalcanti, Simon Foster, Jim Woodcock, and Frank Zeyda. Multi-Model Linking Semantics. Technical report, INTO-CPS Deliverable, D2.3d, December 2017.

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

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

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

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

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

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

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

[CWA16] Ana Cavalcanti, Jim Woodcock, and Nuno Amálio. Behavioural models for FMI co-simulations. In Augusto Sampaio and Farn Wang, editors, *Theoretical Aspects of Computing - ICTAC 2016 - 13th International Colloquium, Taipei, Taiwan,*

ROC, October 24-31, 2016, Proceedings, volume 9965 of *Lecture Notes in Computer Science*, pages 255–273, 2016.

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

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

[DGH07] Pedro A Diaz-Gomez and Dean F Hougen. Initial population for genetic algorithms: A metric approach. In *GEM*, pages 43–49, 2007.

[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.

[FGP17a] John Fitzgerald, Carl Gamble, and Ken Pierce. Method Guidelines 3. Technical report, INTO-CPS Deliverable, D3.3a, December 2017.

[FGP17b] John Fitzgerald, Carl Gamble, and Ken Pierce. Methods Progress Report 3. Technical report, INTO-CPS Deliverable, D3.3b, December 2017.

[FGPL17] John Fitzgerald, Carl Gamble, Richard Payne, and Benjamin Lam. Exploring the Cyber-Physical Design Space. In *Proc. INCOSE Intl. Symp. on Systems Engineering*, volume 27, pages 371–385, Adelaide, Australia, 2017.

[FGPP16] John Fitzgerald, Carl Gamble, Richard Payne, and Ken Pierce. Methods Progress Report 2. Technical report, INTO-CPS Deliverable, D3.2b, December 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.

[FLPV13] John Fitzgerald, Peter Gorm Larsen, Ken Pierce, and Marcel Verhoef. A Formal Approach to Collaborative Modelling and Co-simulation for Embedded Systems. *Mathematical Structures in Computer Science*, 23(4):726–750, 2013.

[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.

[FLV13] John Fitzgerald, Peter Gorm Larsen, and Marcel Verhoef, editors. *Collaborative Design for Embedded Systems – Co-modelling and Co-simulation*. Springer, 2013.

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

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

[Gam17] Carl Gamble. Comprehensive DSE Support. Technical report, INTO-CPS Deliverable, D5.3e, December 2017.

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

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

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

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

[GTB+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, February 2017.

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

[GTB+18b] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. Co-simulation: A Survey. *ACM Computing Surveys*, 51(3):Article 49, April 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.

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

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

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

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

[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.

[HPP+15] Jon Holt, Simon Perry, Richard Payne, Jeremy Bryans, Stefan Hallerstede, and Finn Overgaard Hansen. A model-based approach for requirements engineering for systems of systems. *IEEE Systems Journal*, 9(1):252–262, 2015.

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

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

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

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

[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.

[KLN+17] Christian König, Kenneth Lausdahl, Peter Niermann, Jos Höll, Carl Gamble, Oliver Mölle, Etienne Brosse, Tom Bokhove, Luis Diogo Couto, and Adrian Pop. INTO-CPS Traceability Implementation. Technical report, INTO-CPS Deliverable, D4.3d, December 2017.

[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.

[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.

[LMP+09] Grace A. Lewis, Edwin Morris, Patrick Place, Soumya Simanta, and Dennis B. Smith. Requirements Engineering for Systems of Systems. In *Systems Conference, 2009 3rd Annual IEEE*, pages 247–252. IEEE, March 2009.

[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.

[LNH+16] Kenneth Lausdahl, Peter Niermann, Jos Höll, Carl Gamble, Oliver Mölle, Etienne Brosse, Tom Bokhove, Luis Diogo Couto, and Adrian Pop. INTO-CPS Traceability Design. Technical report, INTO-CPS Deliverable, D4.2d, December 2016.

[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.

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

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

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

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

[NGL+14] Himanshu Neema, Jesse Gohl, Zsolt Lattmann, Janos Sztipanovits, Gabor Karsai, Sandeep Neema, Ted Bapty, John Batteh, Hubertus Tummescheit, and Chandrasekar Sureshkumar. Model-based integration platform for fmi co-simulation

and heterogeneous simulations of cyber-physical systems. In *The 10th International Modelica Conference 2014*, Lund, Sweden, March 2014. Modelica Association.

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

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

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

[OLF+17] Julien Ouy, Thierry Lecomte, Frederik Forchhammer Foldager, Andres Villa Henriksen, Ole Green, Stefan Hallerstede, Peter Gorm Larsen, Luis Diogo Couto, Pasquale Antonante, Stylianos Basagiannis, Sara Falleni, Hassan Ridouane, Hajer Saada, Erica Zavaglio, Christian König, and Natalie Balcu. Case Studies 3, Public Version. Technical report, INTO-CPS Public Deliverable, D1.3a, December 2017.

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

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

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

[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.

[PE12] Birgit Penzenstadler and Jonas Eckhardt. A Requirements Engineering content model for Cyber-Physical Systems. In *RESS*, pages 20–29, 2012.

[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.

[PGP+16] Richard Payne, Carl Gamble, Ken Pierce, John Fitzgerald, Simon Foster, Casper Thule, and Rene Nilsson. Examples Compendium 2. Technical report, INTO-CPS Deliverable, D3.5, December 2016.

[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.

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

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

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

[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.

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

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

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

[TLLM18] Casper Thule, Kenneth Lausdahl, Peter Gorm Larsen, and Gerd Meisl. Maestro: The INTO-CPS Co-Simulation Orches-

tration 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.

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

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

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

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

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

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

[WGS+14] Stefan Wiesner, Christian Gorldt, Mathias Soeken, Klaus-Dieter Thoben, and Rolf Drechsler. Requirements engineering for cyber-physical systems - challenges in the context of "industrie 4.0". volume 438 of *IFIP Advances in Information and Communication Technology*, pages 281–288. Springer, 2014.

[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.

[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, 4 edition, January 2015.

[ZCWO17a] 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.

[ZCWO17b] Frank Zeyda, Ana Cavalcanti, Jim Woodcock, and Julien Ouy. SysML Foundations: Case Study. Technical report, INTO-CPS Deliverable, D2.3a, December 2017.

[ZFC+17] Frank Zeyda, Simon Foster, Ana Cavalcanti, Jim Woodcock, and Julien Ouy. A Mechanised FMI Semantics. Technical report, INTO-CPS Deliverable, D2.3c, December 2017.

A List of Acronyms

20-sim	Software package for modelling and simulation of dynamic systems
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
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 Underlying Principles

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

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

B.1 Co-simulation

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

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

B.2 Design Space Exploration

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

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

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

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

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

B.3 Model-Based Test Automation

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

B.4 Code Generation

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

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

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

C Background on the Individual Tools

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

C.1 Modelio

Modelio is a comprehensive MDE [Fav05] workbench tool which supports the UML2.x standard. Modelio adds modern Eclipse-based graphical environment to the solid modelling and generation know-how obtained with the earlier Softeam MDE workbench, Objecteering, which has been on the market since 1991. Modelio provides a central repository for the local model, which allows various languages (UML profiles) to be combined in the same model, abstraction layers to be managed and traceability between different model elements to be established. Modelio makes use of extension modules, enabling the 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.

C.2 Overture

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

Overture supports the following activities:

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

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

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

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

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

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

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

Finer details can be specified, such as execution synchronisation and mu-

tual exclusion on shared resources. A VDM-RT specification has the same structure as a VDM++ specification, only the conventional system class of VDM++ is mandatory in VDM-RT. Examples can be found in the VDM-RT tutorials distributed with Overture.

C.3 20-sim

20-sim [Con13, Bro97] is a commercial modelling and simulation software package for mechatronic systems. With 20-sim, models can be created graphically, similar to drawing an engineering scheme. With these models, the behaviour of dynamic systems can be 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 62 shows 20-sim with a model of a controlled

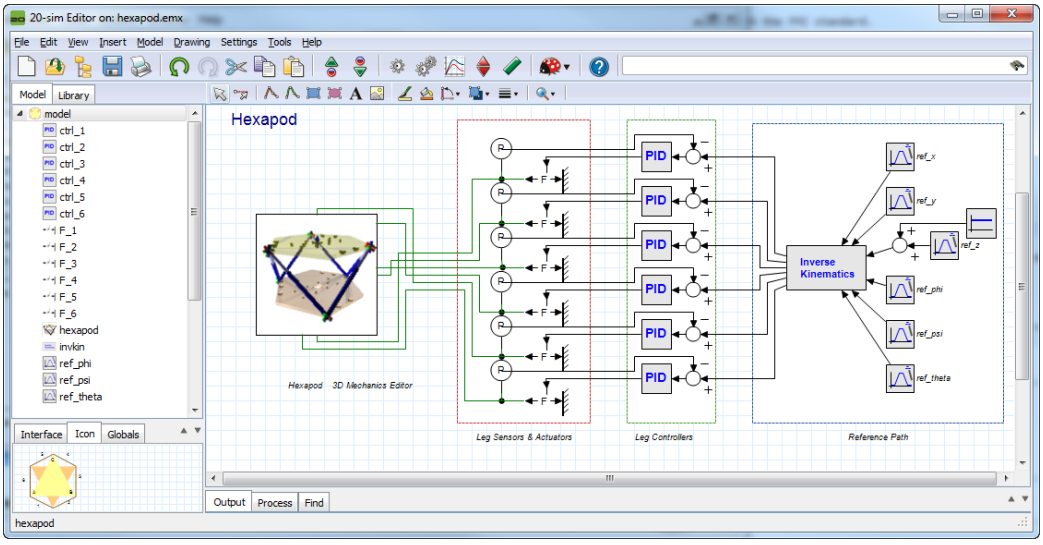


Figure 62: Example of a hexapod model in 20-sim.

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.

C.4 OpenModelica

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

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

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

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

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

C.5 RT-Tester

The RT-Tester [Ver15a] is a test automation tool for automatic test generation, test execution and real-time test evaluation. Key features include a strong C/C++-based test script language, high performance multi-threading, and hard real-time capability. The tool has been successfully applied in avionics, rail automation, and automotive test projects. In the INTO-CPS tool chain, RT-Tester is responsible for model-based testing, as well as for model checking. This section gives some background information on the tool from these two perspectives.

C.5.1 Model-based Testing

The RT-Tester Model Based Test Case and Test Data Generator (RTT-MBT) [Ver15b] supports model-based testing (MBT), that is, automated generation of test cases, test data, and test procedures from UML/SysML models. A number of common modelling tools can be used as front-ends for this. The most important technical challenge in model-based test automation is the extraction of test cases from test models. RTT-MBT combines an SMT solver with a technique akin to bounded model checking so as to extract finite paths through the test model according to some predefined criterion. This criterion can, for instance, be MC/DC coverage, or it can be requirements coverage (if the requirements are specified as temporal logic formulae within the model). A further aspect is that the environment can be modelled within the test model. For example, the test model may contain a constraint such that a certain input to the system-under-test remains in a predefined range. This aspect becomes important once test automation is lifted from single test models to multi-model cyber-physical systems. The derived test procedures use the RT-Tester Core as a back-end, allowing the system under test to be provided on real hardware, software only, or even just simulation to aid test model development.

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

C.5.2 Model Checking of Timed State Charts

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

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

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

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

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

A simple example is depicted in Figure 63, 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].

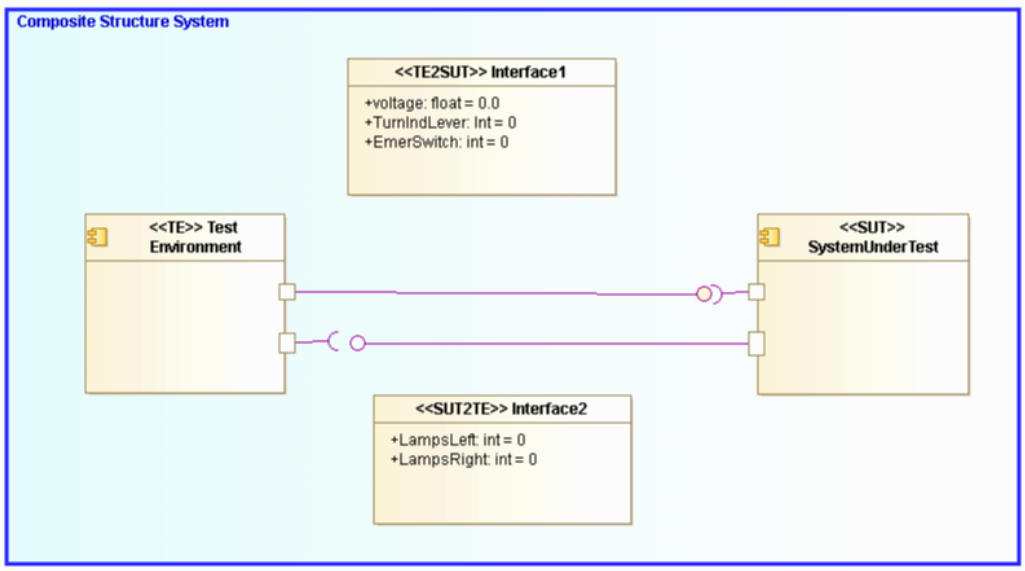


Figure 63: Simple model that highlights interfaces between the environment and the system-under-test.

C.6 4DIAC

4diac 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. The 4DIAC framework allows the development of distributed control systems compliant to the IEC 61499 standard and three of its main projects are:

- 4DIAC-RTE (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. 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 (Integrated Development Environment) 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 to create FBs, applications, configure the devices and all related to IEC 61499 and also download this to devices running FORTE.
- Function Block Library: contains Function Blocks which are available on the 4DIAC-RTE and can, therefore, be used to create IEC 61499 compliant control applications.

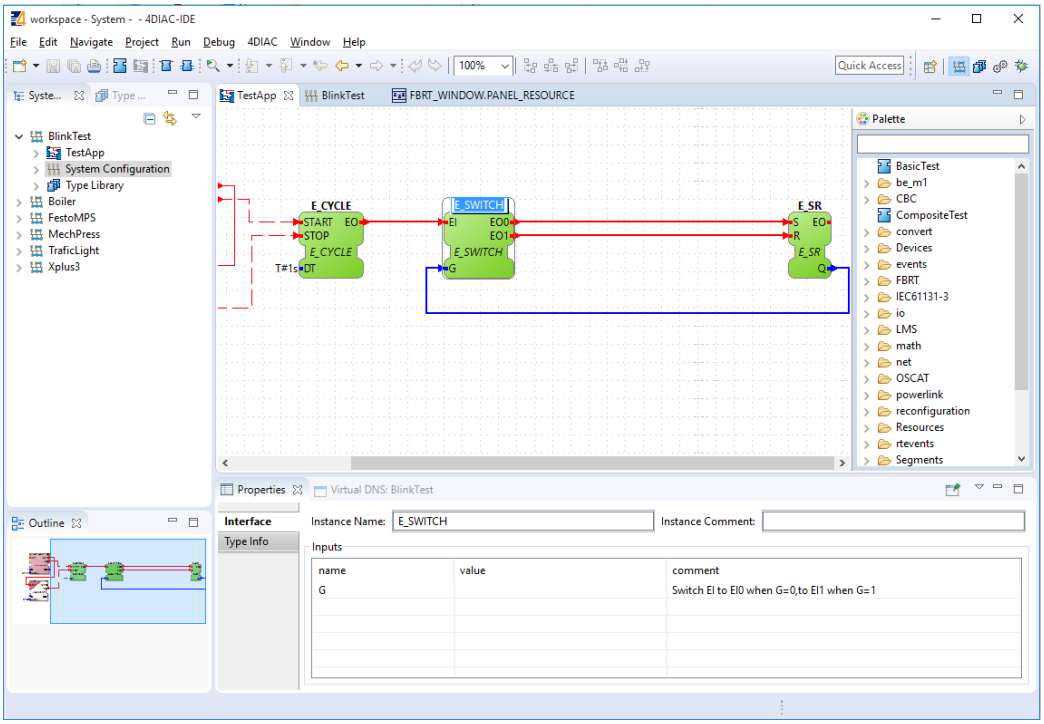


Figure 64: Overview of the 4DIAC-IDE

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`

C.7 AutoFOCUS-3

Hernan Ponce de Leon