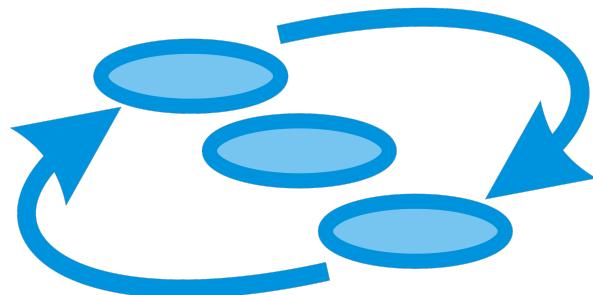




Grant Agreement: 644047

INtegrated TOOl chain for model-based design of CPSs



INTO-CPS

Method Guidelines 1

Deliverable Number: D3.1a

Version: 1.0

Date: December 2015

Public Document

Contributors:

John Fitzgerald, UNEW
Carl Gamble, UNEW
Richard Payne, UNEW
Ken Pierce, UNEW

Editors:

Richard Payne, UNEW

Reviewers:

Christian König, TWT
Andrey Sadovykh, ST
Claes Dühring Jaeger, AI

Consortium:

Aarhus University	AU	Newcastle University	UNEW
University of York	UY	Linköping University	LIU
Verified Systems International GmbH	VSI	Controllab Products	CLP
ClearSy	CLE	TWT GmbH	TWT
Agro Intelligence	AI	United Technologies	UTRC
Softeam	ST		

Document History

Ver	Date	Author	Description
0.1	03-08-2015	Ken Pierce	Draft structure of deliverable and responsibilities
0.2	22-09-2015	Richard Payne	First version of concepts base with initial comments addressed
0.3	28-10-2015	Richard Payne	Deliverable editor comments addressed
0.4	05-11-2015	Ken Pierce	Introduction added
0.5	06-11-2015	Ken Pierce	Workflow material added
0.6	09-11-2015	Ken Pierce	Workflow material revised after comments
0.65	13-11-2015	Ken Pierce	Abstract added
0.9	24-11-2015	Richard Payne	Revised in light of comments
0.95	15-12-2015	Richard Payne	Amended to reflect revised SysML profile
1.0	15-12-2015	Ken Pierce	Finished revisions in light of comments

Abstract

This document is the first of three that will give methods guidance for the INTO-CPS technologies. It is aimed at end users of the technologies. This first version presents an initial concepts base, which describes the terminology used within INTO-CPS; expected workflows for INTO-CPS, focusing on how two existing workflows are enhanced by the technologies; and the use of SysML for holistic architectural modelling for CPSs, complementing the work in Deliverable D2.1a [APCB15].

Contents

1	Introduction	6
2	Concepts and Terminology	7
2.1	Systems	7
2.2	Models	7
2.3	Tools	9
2.4	Analysis	10
2.5	Existing Tools and Languages	10
2.6	Formalisms	11
3	Workflows for INTO-CPS	13
3.1	Workflow Activities	13
3.2	Embedded Systems Waterfall Workflow	15
3.3	Embedded Systems V-Model Workflow	17
4	SysML for Multi-modelling	20
4.1	SysML Modelling Approaches	20
A	List of Acronyms	30
B	Glossary	31

1 Introduction

The material in this document is aimed primarily at new and prospective users of the INTO-CPS technologies. Readers seeking a progress report on the methods work should refer to the companion deliverable, D3.1b Methods Progress Report.

The INTO-CPS technologies bring together a variety of baseline tools and technologies. Each technology has its own culture, abstractions and approaches to problem solving that inform how they are used. Many of these things are tacit, and tend to be discovered only after trying to combine them. The aim of the methods work is to understand how best to use these technologies, to pilot approaches and techniques, and to distill this into a set of guidelines aimed at the users of the technologies—engineers wishing to build cyber-physical systems (CPSs).

As the project continues and the tool chain evolves and matures, this guidance will expand to cover all the engineering processes enabled by the new technology. By the time of the final release of the technologies, the guidelines document will comprehensively cover the INTO-CPS workflow.

In this first version of the guidelines document however, with the combined INTO-CPS technology in its infancy, the guidance focuses on laying the foundations for future. The guidance is broken down into three sections, all aimed at engineers who are new to the INTO-CPS technologies, or who are considering adopting them. These areas are:

Concepts and Terminology (Section 2) This section is an introduction to the concepts and terminology used in INTO-CPS. It explains many terms from the various baseline technologies, as well as other model-based design terminology. In parts this involved reconciling terms used differently in different areas, and finding common, agreed-upon terms for similar concepts. These concepts are applicable for all documents produced by INTO-CPS (this document, user manuals, deliverables, and publications).

Workflows (Section 3) All groups of engineers using model-based design for CPS development will follow a workflow, either implicitly or explicitly. The linking and combining of technologies in INTO-CPS will enable new workflows and alter existing ones. This section gives an initial vision of the types of workflows for model-based design of CPS that will be enabled by the INTO-CPS technologies as they mature, and how these relate to existing workflows. This section is intended to help engineers decide how INTO-CPS technology may help their design process, by comparing current (“pre-INTO”) workflows with expected (“post-INTO”) enhanced workflows. This is the first step towards guidance on aligning INTO-CPS with existing practice.

SysML for Multi-modelling (Section 4) A profile for modelling CPS in SysML has been developed in the project (see Deliverable D2.1a) and implemented in Modelio (see Deliverable D4.1c). This profile requires that blocks map one-to-one to the elements of a multi-model for FMI simulation—DE and CT models packaged as Functional Mockup Units FMUs. However SysML also has utility at an earlier stage in the modelling process, to capture a holistic view of the system, often before assignment to DE and CT domains can be made. This section describes how SysML can be used in this way, and its relation to the SysML profile embodied in the tools.

2 Concepts and Terminology

This section introduces the basic concepts used in the INTO-CPS project. CPSs bring together domain experts from diverse backgrounds, from software engineering to control engineering. Each discipline has developed their own terminologies, principles and philosophy for years — in places they use similar terms for quite different meanings and different terms that have the same meaning. In addition, the INTO-CPS project aims to produce a tool chain for CPS engineering resulting in the need for common tool-based terminology. INTO-CPS requires experts from diverse fields to work collaboratively, so this section gives some core concepts of INTO-CPS that will be used throughout the project. We divide the concepts into several broad areas in the remainder of this section.

2.1 Systems

A *System* is defined as being “a combination of interacting elements organized to achieve one or more stated purposes” [INC15]. Any given system will have an *environment*, considered to be everything outside of the system. The behaviour exhibited by the environment is beyond the direct control of the developer [BFG⁺12]. We also define a *system boundary* as being the common frontier between the system and its environment. The definition of the system boundary is application-specific [BFG⁺12]. *Cyber-Physical Systems (CPSs)* refer to “ICT systems (sensing, actuating, computing, communication, etc.) embedded in physical objects, interconnected (including through the Internet) and providing citizens and businesses with a wide range of innovative applications and services” [Tho13, DAB⁺15]. A *System of Systems (SoS)* is a “collection of constituent systems that pool their resources and capabilities together to create a new, more complex system which offers more functionality and performance than simply the sum of the constituent systems” [HIL⁺14]. CPSs may exhibit the characteristics of SoSs.

2.2 Models

In the INTO-CPS project, we concentrate on “model-based design” of CPSs. A *model* is a potentially partial and abstract description of a system, limited to those components and properties of the system that are pertinent 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]¹.

In a CPS model, we model systems with cyber, physical and network elements. These components are often drawn from different domains, and are modelled in a variety of languages, with different notations, concepts, levels of abstraction, and semantics, which are not necessarily easily mapped one to another. This heterogeneity presents a significant challenge for simulation in CPSs [HIL⁺14]. In INTO-CPS we use *continuous time*

¹Further discussion is required in the second year of INTO-CPS regarding the definition of aspects of models in particular; environment models, test models in RT-Tester and their correspondence in the INTO-CPS SysML profile.

(*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 are described using either explicit continuous functions of time either implicitly as a solution of differential equations. A DE model has state that can be changed and observed only at fixed, *discrete*, time intervals [vA10]. The approach used in the DESTECS project was to use *co-models* – “a model comprising a DE model, a CT model and a contract” [BFG⁺12]. In INTO-CPS we propose the use of *multi-models* – “comprising multiple *constituent* DE and CT models”.

We cover the main features of the notations used in INTO-CPS in Section 2.5. Here we consider some general terms used in models. 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’. In the INTO-CPS project, we use the simple definition from [PHP⁺14]: “an architecture defines the major elements of a system, identifies the relationships and interactions between the elements and takes into account process. Those elements are referred to as *components*. An architecture involves both a definition of structure and behaviour. Importantly, architectures are not static but must evolve over time to reflect the change in a system as it evolves to meet changes to its requirements”. In a CPS architecture, components may be either *cyber components* or *physical components* corresponding to some functional logic or an entity of the physical world respectively.

In INTO-CPS we consider both a *holistic architecture* and a *design architecture*. An example of their use is given in Section 4. The aim of a holistic architecture is to identify the main units of functionality of the system reflecting the *terminology and structure of the domain of application*. It describes a conceptual model that highlights the main units of the system architecture and the way these units are connected with each other, taking a holistic view of the overall system. The design architectural model of the system is effectively a multi-model. The INTO-CPS SysML profile [APCB15] is designed to enable the specification of CPS design architectures, which emphasises a decomposition of a system into *subsystems*, where each subsystem is an assembly of cyber and physical components and possibly other subsystems, and modelled separately in isolation using a special notation and tool designed for the domain of the subsystem. *Evolution* refers to the ability of a system to benefit from a varying number of alternative system components and relations, as well as its ability to gain from the adjustments of the individual components’ capabilities over time (Adjusted from SoS [NLF⁺13]).

Considering the interactions between components in a system architecture, an *inter-*

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

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

The INTO-CPS SysML profile comprises two diagram types. 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.

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 to be orchestrated by the COE. Central to the INTO-CPS tool chain is the use of the Functional Mockup Interface (FMI) standard.

The **Functional Mockup Interface (FMI)** 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 (MA)**.

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 and scales them to the CPS level, allowing CPS multi-models to be evaluated through co-simulation. 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. The **co-simulation configuration** is the configuration that the COE needs to initialise a co-simulation. It contains paths to all FMUs, their inter connection, parameters and step size configuration. When this is combined with a start and end time, a co-simulation can be performed.

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

The INTO-CPS project considers 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) and specifically 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].

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**. 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.

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

INTO-CPS considers three main types of TA: **Hardware-in-the-Loop (HiL)** testing with DE models running on target hardware components; **Software-in-the-Loop (SiL)** testing with software running on CT model simulator; and **Model-in-the-Loop (MiL)** testing with co-simulated CT/DE models.

2.5 Existing Tools and Languages

The INTO-CPS tool chain uses several existing modelling tools. **Overture**² 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

²<http://overturetool.org/>

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*³ 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*⁴ 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*⁵ 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.

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] is a technique to for describing language semantics in a unified framework. A theory of a language is composed of an *alphabet*, a *signature* and a collection of *healthiness conditions*.

³<http://www.20sim.com/>

⁴<https://www.openmodelica.org/>

⁵<http://www.modelio.org/>

The Communicating Sequential Processes *CSP* notation [Hoa85] is a formal process algebra for describing communication and interaction. *INTO-CSP* is a version of CSP, which will be used to provide a model for the SysML-FMI profile, FMI, VDM-RT and Modelica semantics. It is a front end for a UTP theory of reactive concurrent continuous systems customised for the needs of INTO-CPS. *Hybrid-CSP* is a continuous version of CSP defined originally by He Jifeng [Jif94]. It will be used as a basis to inform the design of INTO-CSP.

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

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

3 Workflows for INTO-CPS

As described in the concepts base (Section 2), a workflow is a sequence of activities performed during the development of a CPS. The workflow should have some defined goal, with the activities in the workflow being steps towards that goal. These steps may be repeated, and feed back into each other, as part of an iterative development.

Engineering teams will, either explicitly or implicitly, follow some workflow, either prescribed within their team or based on experience. Many of these existing workflows may already involve model-based design, while others may not. In both cases the INTO-CPS technologies could enhance their existing workflows, but the use of the technologies will differ in each case.

This section is intended to help engineers decide how INTO-CPS technology may help their design process, in particular in how they align to existing practice. The fundamental question that we hope to answer in the workflows work is: *how does the INTO-CPS approach, and its supporting technologies, affect the development process for cyber-physical systems?* In answering this question, we hope to provide insight and guidance on when companies should adopt the INTO-CPS approach. This should take into account their existing workflows, the makeup of their teams skills, and so on.

At this stage of INTO-CPS work, the first version of the technologies are just emerging, so guidance given in this section is speculative, setting out a vision for INTO-CPS. As the technologies mature, the guidance will be updated to take into account real experience of using the technologies. In this first version, we focus on identifying how current workflows could be enhanced by INTO-CPS technologies.

Existing embedded systems developers are an obvious candidate to become developers of CPS. Either moving into this area proactively, or simply finding themselves building increasing complicated, networked systems. This is an area in which guidelines for model-based design have been published, from baseline projects such as DESTECS [FLV14]. For these reasons, in this first version of the guidance, we focus on how embedded systems workflows can be enhanced by INTO-CPS technologies. Further releases of the guidance will be more comprehensive as real experience is gained.

3.1 Workflow Activities

In this section, we list activities that appear in the later workflows. The choice of granularity for defining an activity will affect the size of such a list, however we have tried to select a level that is instructive for describing workflows, but one that does not make the described workflows overly long. Activities are grouped into broad categories. Note that these include both existing, embedded systems activities and activities enabled by INTO-CPS. Since we consider a variety of workflows, both existing and expected, there is overlap between them. For example, those under the **Design** will often be supported by **Modelling** activities, but not necessarily.

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.

 The Crescendo tool.

 OM OpenModelica.

 20-sim.

Descriptions of these tools can be found in the concepts base at the beginning of this document in Section 2.5.

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 4. 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* (  ) 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* ( ) means capturing the “physical” component of the system, e.g. in 20-sim or OpenModelica. Collectively these can be referred to as *Simulation Modelling* (  ) 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 tools 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* (  ) 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.

The above activities are summarised in Table 1. Terms in *italics* correspond to INTO-CPS activities that produce traceable artifacts, as described in the traceability ontology in Deliverable D3.1b [FGPP15].

3.2 Embedded Systems Waterfall Workflow

Current Workflow In this first pre-INTO workflow we consider an engineering team that already uses model-based design, following a waterfall approach. They predominantly build models of physical and control systems. This is because design and low-level control are most critical to their processes. In Fitzgerald et al. [FLV14] this type of workflow is called “continuous-time first” (CT-first) as this is the most common modelling paradigm used in modelling physical systems. In an INTO-CPS context however we coin the term *physical-first* for clarity.

The steps in this example workflow are as follows:

- Requirements gathering
- Plant modelling
- Validation
- Control loop design
- Validation
- Software-in-the-loop (SiL) simulation
- Software design
- Hardware-in-the-loop simulation
- Integration and integration testing

The steps above are visualised in Figure 1a. Requirements are gathered from the customer and an initial plant model is built in order to clarify these requirements and decide on the physical parameters of the device to be built. Existing model components can be reused as appropriate. Validation at this stage involves discussions with the customer to ensure their requirements are met by the design at this stage. The low level control laws are then developed based on the plant model, and another step of validation occurs. Where the system being designed is sufficiently similar to previous products, software-in-the-loop simulation (SiL) can be used to test the model against existing software. Software design can then begin properly, taking into account the control laws previously developed. This can be tested against the plant model as a hardware-in-the-loop simulation (HiL). The plant model can then inform creation of a prototype plant and this can be integrated with the software.

Table 1: Activities in existing embedded systems design workflows or enhanced INTO-CPS workflows. Entries in italics correspond to traceable artifacts in INTO-CPS (see Deliverable D3.1b [FGPP15])

Requirements and Traceability	
<i>Design Notes</i>	
Requirements	
Validation	
Architectural Modelling	
Holistic Architectural Modelling	
Design Architectural Modelling	
<i>Export Model Descriptions</i>	
Modelling	
<i>Import a Model Description</i>	  
Physical Modelling (<i>Simulation Modelling</i>)	 
Cyber Modelling (<i>Simulation Modelling</i>)	
<i>Co-modelling</i>	
<i>Multi-modelling</i>	
Design	
Supervisory Controller Design	
Low Level Controller Design	
Software Design	
Hardware Design	
Analysis	
<i>Create Tests</i>	 
<i>Model Checking</i>	
<i>Create Test Oracle</i>	
<i>Create a Configuration</i>	
<i>Define Design Space Exploration Configurations</i>	
<i>Export FMU</i>	  
Co-simulation	 
Prototyping	
<i>Generate Code</i>	  
Hardware-in-the-Loop (HiL) Simulation	
Software-in-the-Loop (SiL) Simulation	
Manual Code Writing	

Validation meetings can be lengthy, particularly later in the development process as large amounts of information produced by various parts of the team need to be assessed in order to validate each design step. Such workflows are also highly at risk from late-stage discovery of design errors and are not robust to requirements changes from the customer.

Enhanced Workflow In an enhanced workflow using the INTO-CPS technologies we can imagine steps as follows. The double vertical lines indicate that cyber- and physical-modelling occur in parallel:

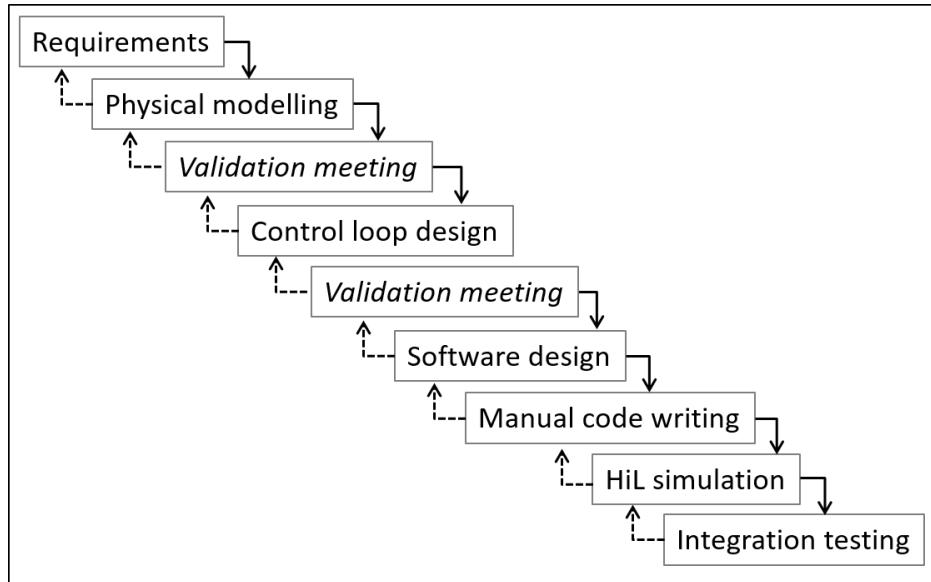
- Requirements gathering
- Architectural Modelling
- Physical modelling || Cyber modelling
- Co-simulation
- Hardware-in-the-loop testing
- Code generation

The steps above are visualised in Figure 1b. Using SysML, engineers are able to build a model of the system that captures the users requirements, deferring choice of “cyber” or “physical” elements if necessary. Guidance on this can see found in Section 4 of this document. The SysML profile (Deliverable D2.1a) can then be used to describe the system in terms that allow model descriptions to be exported to supported modelling tools. Crucially, modelling of the physical elements and control laws can proceed in parallel with modelling of the software. Co-simulation can be used to test these models against each other. Simulation with existing software or hardware (HiL / SiL) is still possible, for example if one modelling activity produces results faster than the other, or before potentially expensive real-world integration of hardware and software.

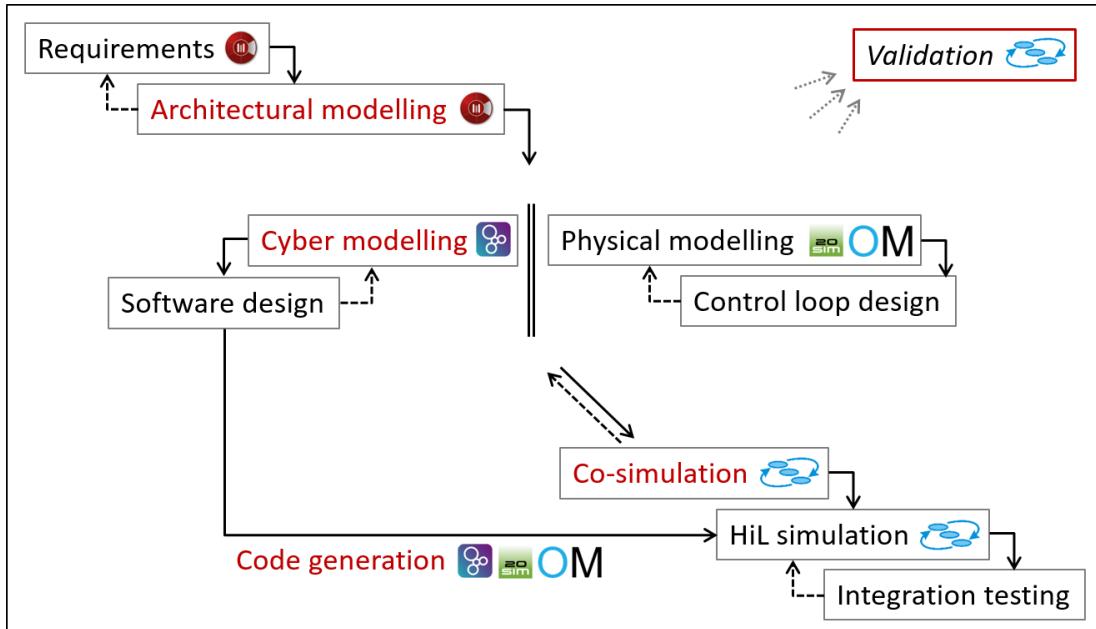
At each step in the workflow, engineers can store and retrieve artifacts using the INTO-CPS App. Design rationale and design notes can be attached to artifacts, as well as information about which engineer created or modified them. This data can be retrieved to reconstruct evidence for validation at any time, reducing the time required in face-to-face meetings decoupling the validation process from the design process. Initial work to define the services required to support this has been carried out and is reported in Deliverable D3.1b [FGPP15]. It is not currently targeted at end users, but may be of interest.

3.3 Embedded Systems V-Model Workflow

Current Workflow In this second pre-INTO workflow, we consider an engineering team that follows the V-model in designing embedded systems. They predominantly worked in software before and use discrete-event modelling in their controller design. The V-model can be seen as an extension to the waterfall model, where the system is designed as a whole, components are identified, designed, tested and finally integrated back to a full realisation. At each stage of the design and implementation, a corresponding validation is carried out. While discrete-event modelling can help for design and validation of cyber



(a) Waterfall workflow for a physical-first design



(b) Enhanced workflow using INTO-CPS technologies

Figure 1: Two workflows for model-based design of CPSs based around an iterative waterfall approach. The first (top) uses single-domain modelling, focusing on the physical plant and defers software design. The second (bottom) shows enhancements by the INTO-CPS technologies in red. The double vertical lines indicate that cyber- and physical-modelling can now occur in parallel.

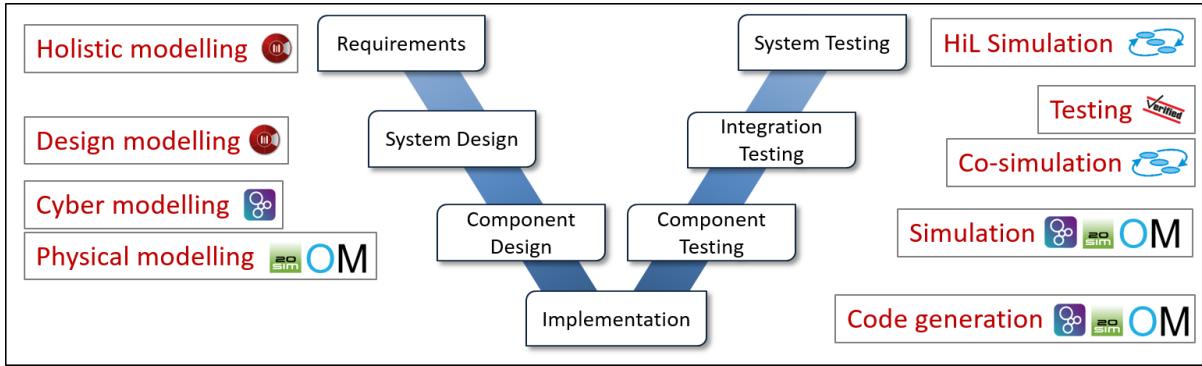


Figure 2: V-model enhanced with INTO-CPS-enabled activities

elements, an enhanced workflow using multi-modelling will allow for improved validation against both models of hardware, and real hardware through HiL simulation.

Enhanced Workflow A representation of the V-model, marked with these enhanced activities, is given in Figure 2. Note that the steps in the V-model here are abstract and do not map to those in Table 1, while the annotated activities do.

Requirements gathering and analysis are enhanced through holistic architectural modelling in SysML, allowing aspects of the system to be captured and understood. *System design* is enhanced through design architectural modelling in SysML, which, informed by the holistic model, allows components to be assigned to the cyber and physical domains. From the design architectural model, model descriptions can be exported to the domain-specific modelling tools (Overture, 20-sim, and OpenModelica). Component design is enhanced through cyber and physical modelling in these tools. Code generation enhances the *implementation* phase. These component models can be tested through simulation, and tested together through co-simulation, enhancing both *Component Testing* and *Integration Testing*. HiL simulation enhances *system testing*, along with test cases that can be generated from the design model and run against the component models, multi-models, and the final realisation of the system.

4 SysML for Multi-modelling

A system architecture defines the major elements (components) of that 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, or capturing the system components in a way that targets multi-modelling. These different goals may give rise to very different models.

In this section we consider different methods for constructing SysML models for use with the INTO-CPS tool chain, in particular the relation between *holistic* and *design* architectural models.

4.1 SysML Modelling Approaches

The INTO-CPS SysML profile, defined in Deliverable D2.1a [APCB15], proposes the structuring of a CPS architectural model into ‘subsystems’. These subsystems may be composed of cyber or physical components. Using the INTO-CPS tool chain, each subsystem corresponds to a FMI model description, and therefore an individual model in a multi-model.

Thinking of a system in this way is not necessarily natural – especially when designing a system ab initio and with systems comprising entities across different domains requiring diverse domain expertise. Whilst both *holistic* and *design* architecting advocate modelling a system in a modular way, it may not be natural to split model elements by the notations they are to be ultimately modelled in.

In this section we use a smart grid example to show the different 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 to a multi-modelling approach.

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, 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 Architecture

The first diagram, the Block Definition Diagram (BDD) of the Smart Grid, is given in Figure 3. 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 *Step-down Transformer*. The *House* element comprises: a cyber *House Controller*, physical *House Meter* and *Devices*, and a *Owner/Usage Profile*.

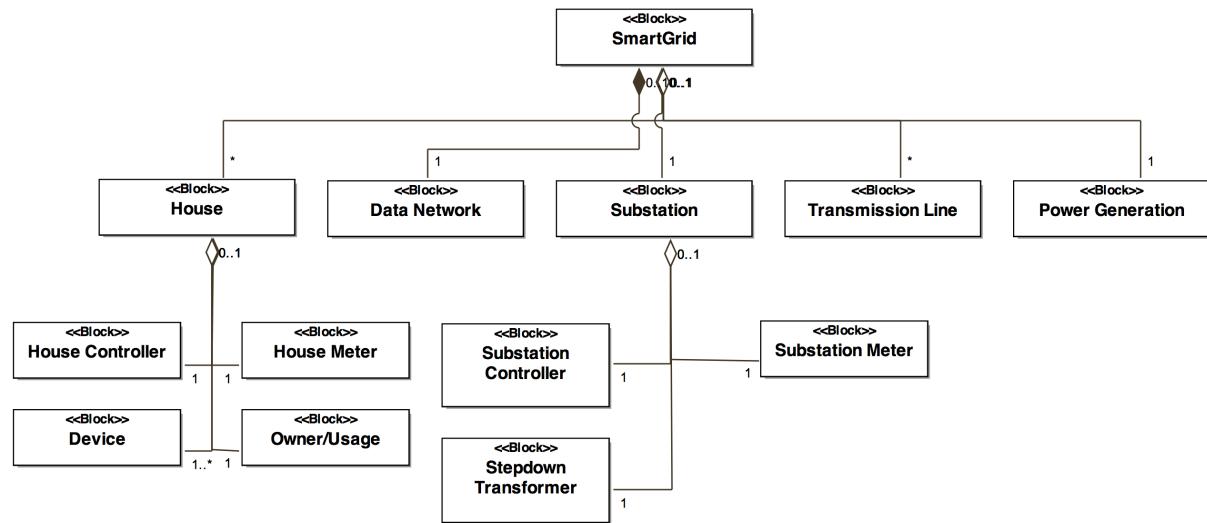


Figure 3: Block Definition Diagram of Smart Grid

The second SysML diagram in Figure 4 is the Internal Block Diagram (IBD) of the smart grid. 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 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.

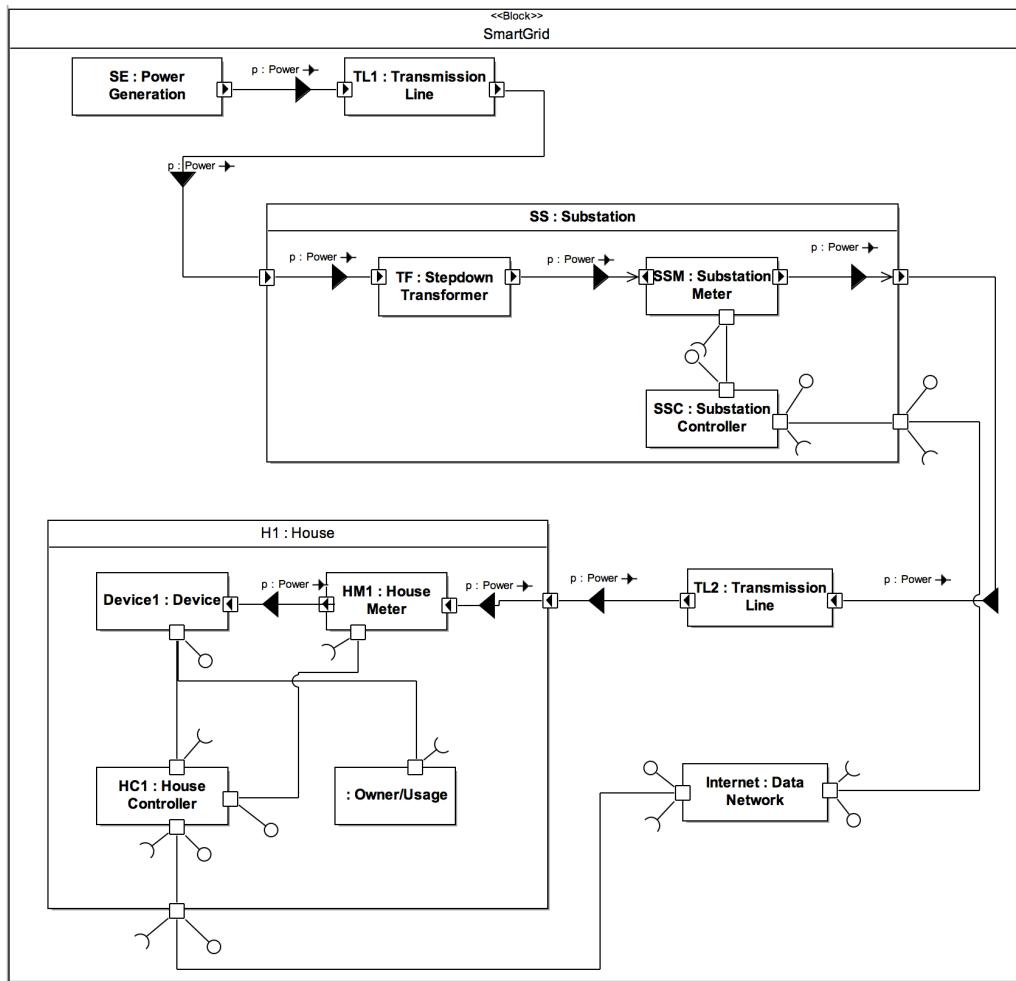


Figure 4: Internal Block Diagram of Smart Grid

A Co-modelling Approach

There are degrees to the extent of splitting the modelling elements into separate subsystems in a design architecture. To highlight the different ways in which the INTO-CPS profile may be used to model the architecture, we first consider how this model could be represented as a co-model – that is a multi-model with one DE and one CT model defined in 20-sim and VDM-RT respectively. In this section, we use the INTO-CPS SysML profile to define the architecture of the Smart Grid from the perspective of informing a co-model.

In the Architecture Structure Diagram (ASD) in Figure 5 we decompose the Smart Grid system into two subsystems: *Physical Elements* and *Cyber Elements*, which contain the physical and cyber elements respectively. This corresponds to two models – the *Physical Elements* subsystem is modelled in 20-sim and the *Cyber Elements* subsystem is modelled in VDM-RT.

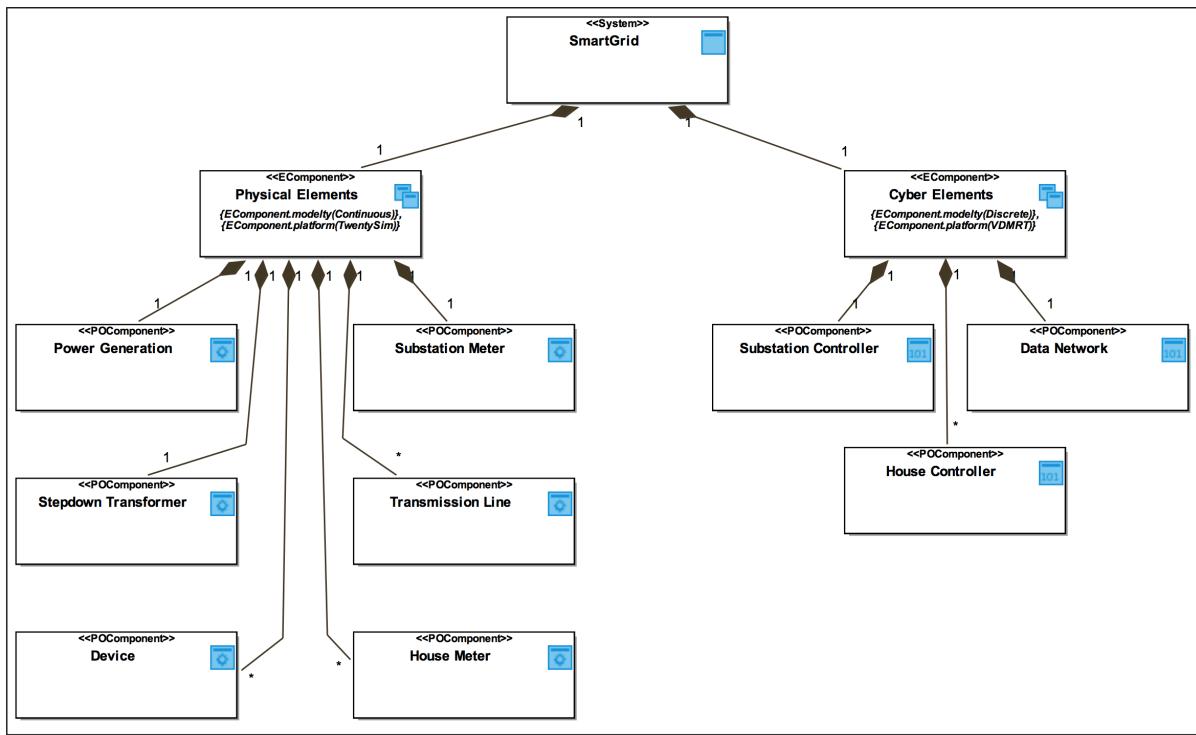


Figure 5: Architecture Structure Diagram for co-model of Smart Grid

The connections between the components are defined in the Connections Diagram (CD) in Figure 6, with the interface between subsystems defined as the interaction points between cyber and physical components. For example, as the physical *Substation Meter* communicates with the cyber *Substation Controller*, an interface is defined between the subsystems to allow this communication.

Multi-model

In contrast to the above co-model, INTO-CPS allows for multiple cyber- and physical-models. This gives more freedom of choice in the target platform and model types.

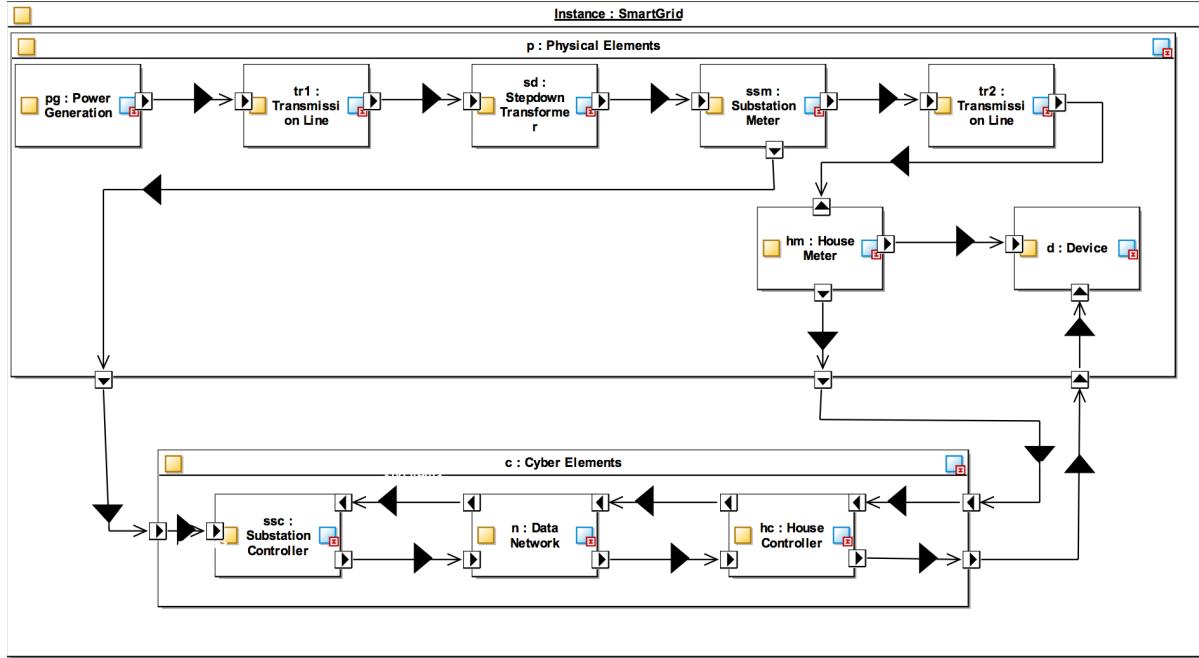


Figure 6: Connection Diagram for co-model of Smart Grid

Looking again at the holistic architecture defined in Figures 3 and 4 and moving towards a pure multi-model, we achieve and architecture structure shown in the ASD in Figure 7. This structure removes all subsystem structures such that each element is to be defined in a single FMU. Each element is defined as either a *Physical* or *Cyber* component, with the model type and platform identified.

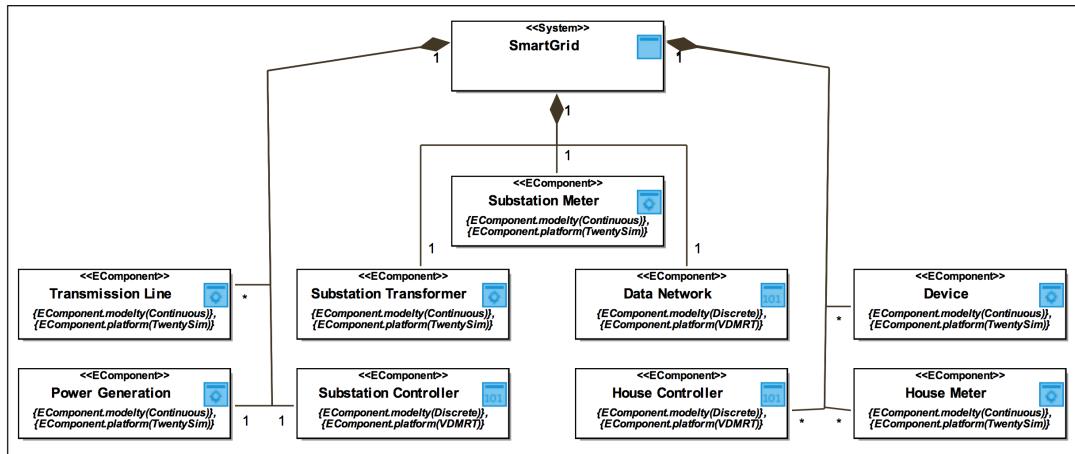


Figure 7: Architecture Structure Diagram for multi-model of Smart Grid

The CD, shown in Figure 8, lifts all the previously defined components to the top-level but is otherwise the same as in Figure 6.

Commentary

Contrasting the architectures shown in the initial model (Figures 3 and 4) to that in the multi-model (Figures 7 and 8), whilst the same base components are present in both, we

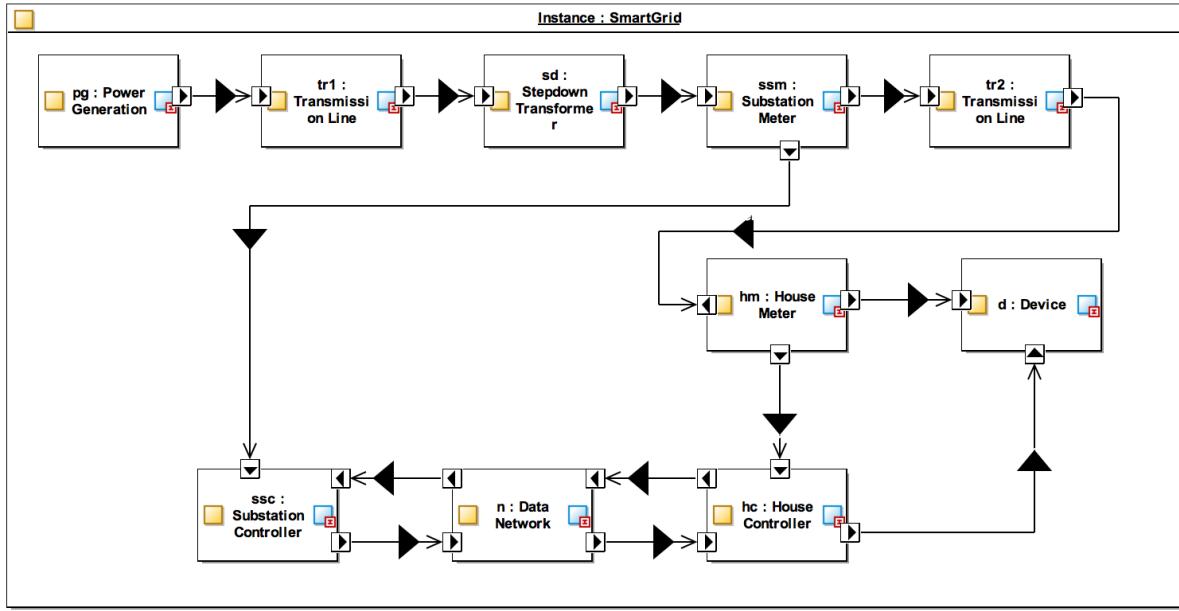


Figure 8: Connections Diagram for multi-model of Smart Grid

lose some of the intuitive domain-specific structures when moving to a multi-model. For example, it is now not clear where the *substation* or *house* elements are in the multi-model. This is primarily due to the fact that the INTO-CPS profile does not support subsystems with mixed model types and platforms.

An important and subtle issue here is in the reason behind producing the different architectural model types. As defined in the INTO-CPS concept base, using SysML diagrams in a *holistic* approach, a CPS engineer 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 architecture* approach advocated in INTO-CPS requires the modeller to define the system structure directly in terms of the target model.

Both approaches are clearly valid, and as shown in the co-modelling and multi-modelling approaches, there are degrees to the extent of splitting the modelling elements into separate subsystems. This may be guided by the modelling goals (e.g. understanding the system in terms of the application domain, or capturing the system components in a way that targets multi-modelling).

Figure 9 presents an overview of the relationships between the different models in INTO-CPS. 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 3 to be individual FMUs in a multi-model is an evolution which will likely result in a feasible model. It may, therefore be advisable to define the architecture using the INTO-CPS profile ab initio. However, the Three-tank Water Tank pilot study (described in detail in Deliverable D3.4 [FGP⁺15]) shows an example of a model which requires close coupling of components. This therefore may result in a model that would not be intuitively reached using domain knowledge only. It is therefore important to

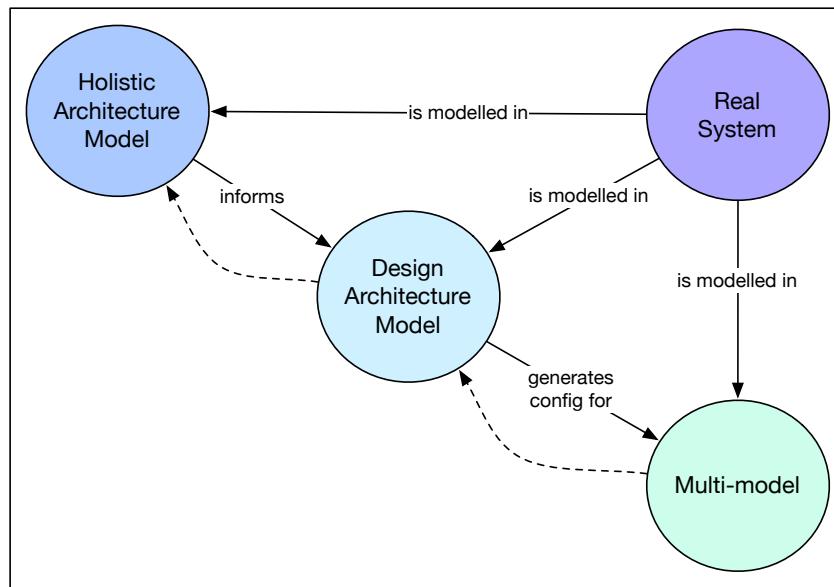


Figure 9: Relating holistic and design architectures

define the architecture holistically at the start of the CPS engineering design stage and move to a *design architecture* with experience and understanding of the nature of the modelling notations and analysis goals.

References

- [APCB15] Nuno Amadio, Richard Payne, Ana Cavalcanti, and Etienne Brosse. Foundations of the SysML profile for CPS modelling. Technical report, INTO-CPS Deliverable, D2.1a, December 2015.
- [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.
- [Blo14] Torsten Blochwitz. Functional mock-up interface for model exchange and co-simulation. <https://www.fmi-standard.org/downloads>, July 2014. Torsten Blochwitz Editor.
- [BW98] Ralph-Johan Back and Joakim Wright. *Refinement Calculus: A Systematic Introduction*. Springer, 1998.
- [CMC⁺13] Joey Coleman, Anders Kael 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.
- [DAB⁺15] Lipika Deka, Zoe Andrews, Jeremy Bryans, Michael Henshaw, and John Fitzgerald. D1.1 definitional framework. Technical report, The TAMS4CPS Project, April 2015.
- [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.
- [FGP⁺15] John Fitzgerald, Carl Gamble, Richard Payne, Ken Pierce, and Jörg Brauer. Examples Compendium 1. Technical report, INTO-CPS Deliverable, D3.4, December 2015.
- [FGPP15] John Fitzgerald, Carl Gamble, Richard Payne, and Ken Pierce. Methods Progress Report 1. Technical report, INTO-CPS Deliverable, D3.1b, December 2015.
- [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.
- [FLV14] John Fitzgerald, Peter Gorm Larsen, and Marcel Verhoef, editors. *Collaborative Design for Embedded Systems – Co-modelling and Co-simulation*. Springer, 2014.

- [GF94] Orlena C.Z. Gotel and Anthony C.W. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of the First International Conference on Requirements Engineering*, pages 94–101, April 1994.
- [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.
- [Hoa85] Tony Hoare. *Communication Sequential Processes*. Prentice-Hall International, Englewood Cliffs, New Jersey 07632, 1985.
- [INC15] INCOSE. Systems Engineering Handbook. A Guide for System Life Cycle Processes and Activities, Version 4.0. Technical Report INCOSE-TP-2003-002-04, International Council on Systems Engineering (INCOSE), January 2015.
- [Jif94] He Jifeng. A classical mind. chapter From CSP to Hybrid Systems, pages 171–189. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1994.
- [MG13] Luc Moreau and Paul Groth. PROV-Overview. Technical report, World Wide Web Consortium, 2013.
- [Mor90] Carroll Morgan. *Programming from Specifications*. Prentice-Hall, London, UK, 1990.
- [NLF⁺13] Claus Ballegaard Nielsen, Peter Gorm Larsen, John Fitzgerald, Jim Woodcock, and Jan Peleska. Model-based engineering of systems of systems. *Submitted to ACM Computing Surveys*, June 2013.
- [NN92] Hanne Riis Nielson and Flemming Nielson. *Semantics With Applications – A Formal Introduction*. John Wiley & Sons Ltd, 1992.
- [PF10] Richard J. Payne and John S. Fitzgerald. Evaluation of Architectural Frameworks Supporting Contract-based Specification. Technical Report CS-TR-1233, School of Computing Science, Newcastle University, December 2010.
- [PHP⁺14] Simon Perry, Jon Holt, Richard Payne, Jeremy Bryans, Claire Ingram, Alvaro Miyazawa, Luís Diogo Couto, Stefan Hallerstede, Anders Kael Malmos, Juliano Iyoda, Marcio Cornelio, and Jan Peleska. Final Report on SoS Architectural Models. Technical report, COMPASS Deliverable, D22.6, September 2014. Available at <http://www.compass-research.eu/>.
- [Plo81] Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.
- [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 SysMLTM). 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.
- [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.
- [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.

A List of Acronyms

20-sim	Software package for modelling and simulation of dynamic systems
ACA	Automated Co-model Analysis
AST	Abstract Syntax Tree
AU	Aarhus University
CLE	ClearSy
CLP	Controllab Products B.V.
COE	Co-simulation Orchestration Engine
CPS	Cyber-Physical Systems
CT	Continuous-Time
DE	Discrete Event
DESTECS	Design Support and Tooling for Embedded Control Software
DSE	Design Space Exploration
FMI	Functional Mockup Interface
FMI-Co	Functional Mockup Interface – for Co-simulation
FMI-ME	Functional Mockup Interface – Model Exchange
FMU	Functional Mockup Unit
HiL	Hardware-in-the-Loop
HMI	Human Machine Interface
HW	Hardware
ICT	Information Communication Technology
IDE	Integrated Design Environment
M&S	Modelling and Simulation
MBD	Model Based Design
MiL	Model-in-the-Loop
OMG	Object Management Group
OS	Operating System
PROV-N	The Provenance Notation
RPC	Remote Procedure Call
SiL	Software-in-the Loop
ST	Softeam
SVN	Subversion
SysML	Systems Modelling Language
TA	Test Automation
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 Technology Research Center
UY	University of York
VDM	Vienna Development Method
VSI	Verified Systems International
WP	Work Package
XML	Extensible Markup Language

B Glossary

20-sim The 20-sim tool can represent continuous time models in a number of ways. The core concept is that of connected *blocks*.

Abstraction Models may be abstract “in the sense that aspects of the product not relevant to the analysis in hand are not included” [FL98]. CPS models may reasonably contain multiple levels of abstraction, for representing views of individual constituent systems and for the view of the CPS level. Adapted from [HIL⁺14].

Architecture The term architecture has many different definitions, and range in scope depending upon the scale of the product being ‘architected’. In the INTO-CPS project, 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. 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.”

Architecture Diagram In the INTO-CPS project, a diagram refers to the symbolic representation of information contained in a model.

Architectural Framework “A defined set of viewpoints and an ontology” and “is used to structure an architecture from the point of view of a specific industry, stakeholder role set, or organisation. [HIL⁺14]. [HIL⁺14].

Architecture Structure Diagram (ASD) The INTO-CPS SysML profile ASDs specialise SysML block definition diagrams to support the specification of a system architecture described in terms of a system’s components.

Architecture View “work product expressing the architecture of a system from the perspective of specific system concerns” [PHP⁺14].

Bond graph 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.

Co-model “The term *co-model* is used to denote a model comprising a DE model, a CT model and a contract” [BFG⁺12].

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

Collaborative simulation (co-simulation) 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.

Co-simulation Configuration The configuration that the COE needs to initialise a co-simulation. It contains paths to all FMUs, their inter connection, parameters and step size configuration. When this is combined with a start and end time, a co-simulation can be performed.

Co-simulation Orchestration Engine (COE) The Co-simulation Orchestration Engine combines existing co-simulation solutions and scales them to the CPS level, allowing CPS co-models to be evaluated through co-simulation. 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.

Component The constituent elements of a system.

Connections Diagram (CD) The INTO-CPS SysML profile CDs specialise SysML internal block diagrams to convey the internal configuration of the system's components and the way they are connected.

Continuous Time (CT) model A model with state that can be changed and observed *continuously* [vA10], and are described using either explicit continuous functions of time either implicitly as a solution of differential equations.

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

Discrete Event (DE) model A model with state that can be changed and observed only at fixed, *discrete*, time intervals [vA10].

Denotational Semantics 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].

Design Alternatives Where two or more models represent different possible solutions to the same problem. Each choice involves making a selection from alternatives on the basis of criteria 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.

Design Architecture The design architectural model of the system is effectively a multi-model. The INTO-CPS SysML profile [APCB15] is designed to enable the specification of CPS design architectures, which emphasises a decomposition of a system into *subsystems*, where each subsystem is modelled separately in isolation using a special notation and tool designed for the domain of the subsystem.

Design Parameter 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].

Design Space “The *design space* is the set of possible solutions for a given design problem” [BFG⁺12].

Design-Space Exploration (DSE) “an activity undertaken by one or more engineers in which they build and evaluate co-models in order to reach a design from a set of requirements” [BFG⁺12].

Effort and Flow 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.

Environment A system's *environment* is everything outside of the system. The behaviour exhibited by the environment is beyond the direct control of the developer [BFG⁺12].

Evolution This refers to the ability of a system to benefit from a varying number of alternative system components and relations, as well as its ability to gain from the adjustments of the individual components' capabilities over time (Adjusted from SoS [NLF⁺13]).

Functional Mockup Interface (FMI) The Functional Mock-up Interface (FMI) 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].

Functional Mockup Unit (FMU) Component that implements FMI is a Functional Mockup Unit (FMU) [Blo14].

Hardware-in-the-Loop Testing Testing with DE models running on target hardware components.

Holistic Architecture The aim of a holistic architecture is to identify the main units of functionality of the system reflecting the *terminology and structure of the domain of application*. It describes a conceptual model that highlights the main units of the system architecture and the way these units are connected with each other, taking a holistic view of the overall system.

Hybrid-CSP This 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.

Interface “Defines the boundary across which two entities meet and communicate with each other” [HIL⁺14]. Interfaces may describe both digital and physical interactions: digital interfaces contain descriptions of operations and attributes that are *provided* and *required* by components. Physical interfaces describe the flow of physical matter (for example fluid and electrical power) between components.

INTO-CPS Application The INTO-CPS Application is a front-end to the INTO-CPS tool chain. The application allows the specification of the co-simulation configuration to be orchestrated by the COE.

INTO-CPS tool chain 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.

INTO-CSP 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.

Master Algorithm A Master Algorithm (MA) controls the data exchange between FMUs and the synchronisation of all simulation solvers [Blo14].

Model a potentially partial and abstract description of a system, limited to those components and properties of the system that are pertinent to the current goal [HIL⁺14].

“A model is a simplified description of a system, just complex enough to describe or study the phenomena that are relevant for our problem context” [vA10]. A model “may contain representations of the system, environment and stimuli” [FLV14]

Model Description The model description file is an XML file that supplies a description of all properties of a model (for example input/output variables) [Blo14].

Model-in-the-Loop Testing Testing with co-simulated CT/DE models.

Modelling “The activity of creating models” [FLV14]. See also **co-modelling** and **multi-modelling**.

Modelica 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.

Multi-model “A model comprising *multiple* DE and CT models”.

Non-functional Property 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].

Objective Criteria or constraints that are important to the developer, such as cost or performance

Port 20-sim blocks may have input and output *ports* that allow data to be passed between them. In SysML, blocks own ports — the points of interaction between blocks.

Proof The process of showing how the validity of one statement is derived from others by applying justified rules of inference [BFL⁺94].

Provenance “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].

Refinement 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.

Semantics Describes the meaning of a (grammatically correct) language [NN92].

Software-in-the-Loop Testing Testing with software running on CT model simulator

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.

SysML The systems modelling language (SysML) [Sys12] extends a subset of the Unified Modelling language (UML) to support modelling of heterogeneous systems.

System “A combination of interacting elements organized to achieve one or more stated purposes” [INC15].

System boundary The *system boundary* is the common frontier between the system and its environment. System boundary definition is application-specific [BFG⁺12].

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

System Under Test “The system currently being tested for correct behaviour. An alias for system of interest, from the point of view of the tester. The same concept can be extended from systems engineering to SoS engineering, changing the focus from a single system of interest to an SoS under test.”

“The system of systems currently being tested for correct behaviour” [HIL⁺14].

Test Automation 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 their requirements.

Test Case A finite structure of input and expected output [UPL06].

Test model Specifies the expected behaviour of a system under test. Note that a test model can be different from a design model. It might only describe a part of a system under test that is to be tested and it can describe the system on a different level of abstraction [CMC⁺13].

Test procedures Detailed instructions for the set-up and execution of a set of test cases, and instructions for the evaluation of results of executing the test cases [WG-92, CMC⁺13].

Test suite A collection of test procedures.

Traceability The association of one model element (e.g. requirements, design artefacts, activities, software code or hardware) and specifically requirements traceability “refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction” [GF94].

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

Variable A *variable* is feature of a model that may change during a given simulation [BFG⁺12].

VDM-RT 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.

Workflow 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.