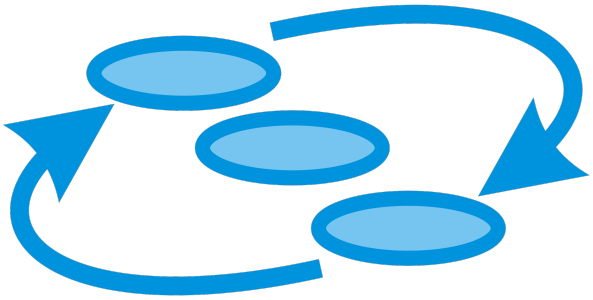
Grant Agreement: 644047

INtegrated TOol chain for model-based design of CPSs



INTO-CPS

Examples Compendium 1

Deliverable Number: D3.4

Version: 1.0

Date: 2015

Public Document

Contributors:

John Fitzgerald, UNEW
Carl Gamble, UNEW
Richard Payne, UNEW
Ken Pierce, UNEW
Jörg Brauer, VSI

Editors:

Richard Payne, UNEW

Reviewers:

Peter Gorm Larsen, AU
Etienne Brosse, ST
Adrian Pop, LIU

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	25-09-2015	Richard Payne	Initial example descriptions added
0.3	13-10-2015	Richard Payne	Version for internal review
0.31	23-10-2015	Richard Payne	Review comments addressed
0.32	26-10-2015	Richard Payne	Added COE simulation text for Three-Tank pilot
0.4	14-12-2015	Richard Payne	Amended to reflect revised SysML profile
1.0	15-12-2015	Richard Payne	Finished revisions in light of comments

Abstract

This deliverable is intended for users of the INTO-CPS technologies and contains a collection of example and pilot study model descriptions demonstrating baseline INTO-CPS technology. Each study has a description of the example and of the models available for the study. The examples demonstrate the baseline technologies and some early INTO-CPS tools. The deliverable also lays out a roadmap for the next 12 months of case study and example development to test and demonstrate upcoming INTO-CPS technologies.

Contents

1 Introduction

This deliverable provides an overview of different public example multi-models that stakeholders who are interested in experimenting with the INTO-CPS technology can use as a starting point. The examples have been developed using baseline technologies: Crescendo[1] (which includes two technologies: 20-sim[2] and Overture/VDM-RT[3]); OpenModelica[4]; SysML[5]; and RT-Tester[6]). This deliverable also outlines early use of INTO-CPS technologies; proposing initial multi-models using the INTO-CPS SysML profile and collections of Continuous Time (CT) and Discrete Event (DE) models elicited from the baseline models. The document concludes by laying out a roadmap for the next 12 months of case study and example development to test and demonstrate upcoming INTO-CPS technologies.

This deliverable is structured in different sections, each of which provides a brief (2 to 3 pages) introduction to each example model. The examples each illustrate different aspects of the baseline technology and INTO-CPS technology, as summarised here:

- Section 2 presents a Three-tank Water Tank model. This example considers the effect of moving from a single-CT Crescendo model to a multi-CT multi-model (both have a VDM DE model). This study demonstrates the impact of the division of CT elements across different FMUs. The study uses SysML to model DE behaviour.
- Section 3 illustrates a Fan Coil Unit (FCU), originally presented as a baseline OpenModelica model. The study aimed to represent this in a Crescendo model; testing the expressiveness of both CT notations and defining a DE controller. The study may be used as a two-model multi-model with the current version of the INTO-CPS technology.
- Section 4 presents a Line-following Robot. The study originated as a Crescendo co-model, and considered the representation of subsystems in the baseline OpenModelica notation. The study may be co-simulated as a three-model, multi-CT, multi-model. The study uses SysML to model CT behaviour.
- Section 5 presents a Turn Indicator example. This study demonstrates the RT-Tester baseline tool; comprising a collection of requirements, architectural model and test cases.

In order to guide you in what models to consider inspecting, we have created tables illustrating the different characteristics of the different publicly available multi-models. Table 1 shows the baseline technologies applied and Table 2 shows the INTO-CPS technologies applied (technologies not considered yet are greyed out). The turn indicator example is not used to demonstrate INTO-CPS technologies, and as such is not included in Table 2. It should be noted that FMU export has not been tested at this stage. In this first year, the emphasis was to define how to restructure existing models in order to produce multi-models and consider how to connect these separate models. These models

[1] `http://www.crescendotool.org`
[2] `http://www.20sim.com`
[3] `http://overturetool.org`
[4] `https://openmodelica.org`
[5] Using the Modelio tool: `https://www.modeliosoft.com`
[6] `https://www.verified.de/products/rt-tester/`

will be used to test the FMU export feature of the INTO-CPS tool chain, and in turn the COE, in the next 12 months.

	Baseline			
Multi-model	Crescendo (20-sim & VDM-RT)	OpenModelica	'Holistic' SysML model	RTTester
Three-tank Water Tank	x		x	
Fan Coil Unit (FCU)	x	x		
Line-following Robot	x	x	x	
Turn Indicator				x

Table 1: Overview of baseline technologies used for pilot studies

	INTO-CPS Technology													
Multi-model	Multi-DE model	Multi-CT model	20-Sim (for FMU)	OpenModelica (for FMU)	VDM-RT (for FMU)	INTO-CPS SysML	Co-simulation engine(COE)	SysML requirements	Traceability links included	Provenance graph included	DSE support included	Test Automation support	Model checking	SiL/HiL enabled
Three-tank Water Tank		x	x		x	x								
Fan Coil Unit (FCU)						x								
Line-following Robot		x	x	x	x	x								

Table 2: Overview of INTO-CPS technologies used for pilot studies

Section 6 presents a roadmap for the next 12 months of pilot case study development. We identify the various INTO-CPS technologies in production over the project, propose additional examples, and consider which examples may be used to test the emerging technologies.

2 Three-tank Water Tank

2.1 Example Description

The three-tank water tank model is an augmentation of a standard 20-sim example, and is developed to explore the impact on accuracy of multi-modelling across multiple CT models. The example comprises three water tanks which are filled and emptied. The first tank is filled from a source with a valve which may be turned on and off. The outflow of the first tank constitutes the inflow of the second, and so forth. A controller monitors the level of the third tank and controls a valve to a drain.

A key feature of this example is the close coupling required between water tank 1 and 2, and the loose coupling to water tank 3. Water tanks 1 and 2 are tall and thin and are connected by a pipe at the bottom of the tanks (a diagram of the example is shown in Figure 1), and therefore changes to the level of water tank 1 (due to water entering from the source) will quickly affect the level in water tank 2. This effect is not as prevalent between water tank 2 and 3.

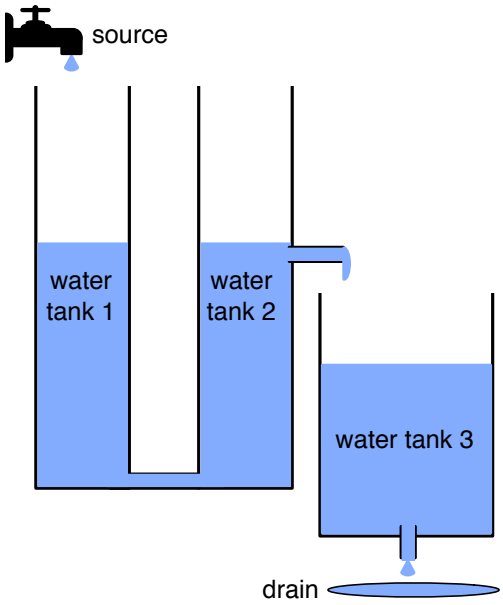


Figure 1: Overview of the three-tank water tank example

2.2 Baseline

In the three-tank water tank study, we demonstrate the use of the SysML (Section 2.2.1) and Crescendo (Section 2.2.2) baseline technologies.

2.2.1 SysML

In the SysML model, we concentrate largely on the structure of the example: producing block definition diagrams (BDDs) for the example composition and datatype definitions; and an internal block diagram (IBD) defining the connections between the model elements.

The BDD in Figure 2 states that the system constitutes 3 *Water Tank* components (each comprising a *Measured Vessel* and optionally a controllable *Valve*), a *Pipe*, a *Source*, a *Drain* and a *Controller*. Given the system composition, the connections between those components are specified in the IBD in Figure 3.

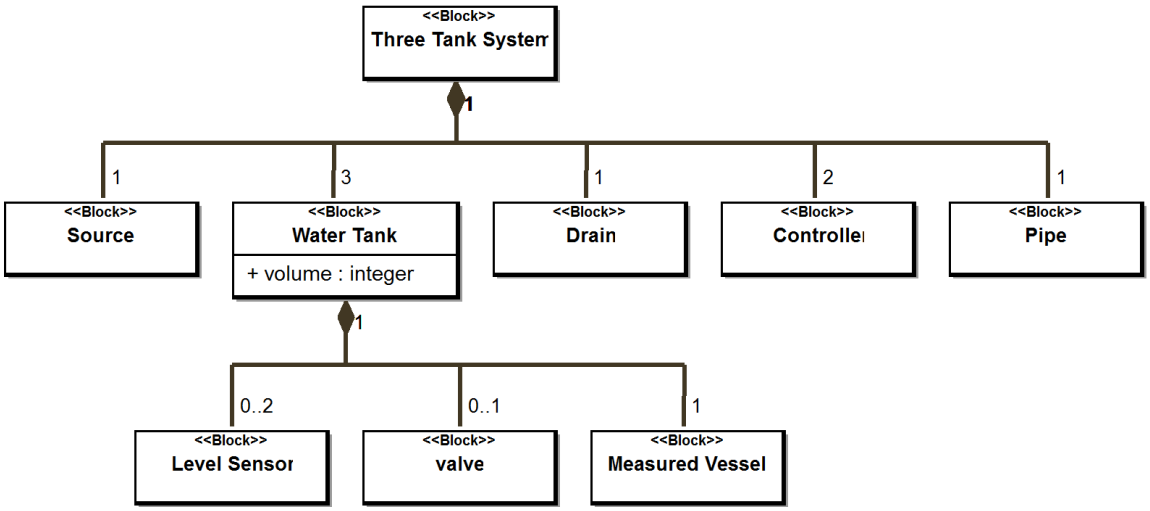


Figure 2: BDD defining the Three-tank Water Tank composition

The IBD defines input and output flow ports for each component with a directed connection between ports. The IBD states that there is a flow of water (typed by a model-specific value type *Water*) from the *Source* instance *src*, through the *Water Tank* instances *wt1, wt2* and *wt3* to the *Drain* instance *d*. A *Pipe*, *p* is situated between *wt1* and *wt2*. There are connections to the *Controller* instance *c* and *Water Tank*, *wt3*.

The controller is defined in Figure 4. Firstly, a UML class diagram in Figure 4(a) defines the substructure of the controller; comprising classes for obtaining sensor data (*LevelSensor*) and actuator control (*ValveActuator*). A state machine in Figure 4(b) describes the controller behaviour; defining the order of operation calls. The controller checks the water level of the tank, if it is higher than the max_limit, the valve is set to *open* and if the level falls below the min_limit, the valve is set to *closed*.

2.2.2 Crescendo

CT Model

The CT model of the water tanks system uses capacitors (labelled *Tank1*, *Tank2* and *Tank3*) to represent the water tanks, with *Tank3* also having a valve. The connecting pipe

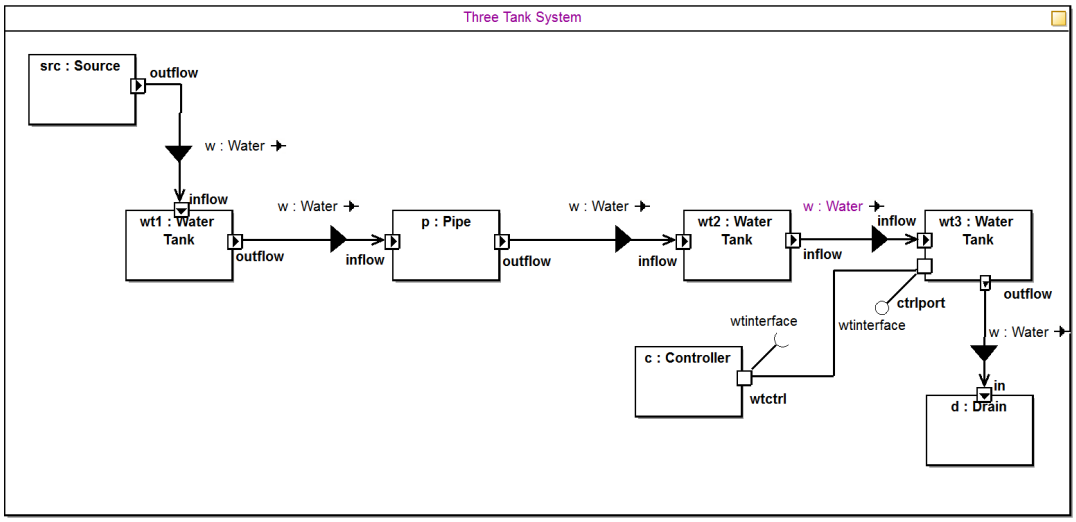
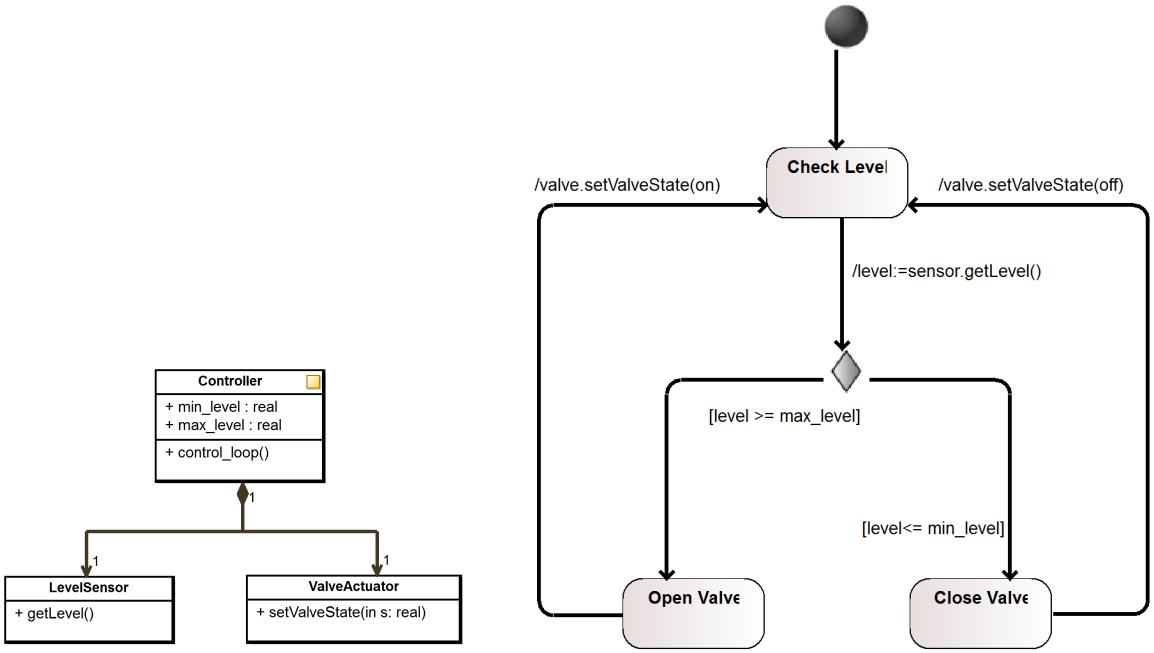


Figure 3: IBD defining the Three-tank Water Tank connections



(a) UML class diagram defining the Three-tank Water Tank controller

(b) State machine diagram defining the Three-tank Water Tank controller behaviour

Figure 4: Behavioural diagrams for Three-tank Water Tank controller

between *Tank1* and *Tank2*, and the outflow of *Tank2*, are modelled using a combination of an inductor and a resistor. Figure 5 shows the 20-sim block diagram.

DE Model

The DE model is a simple controller, which governs *Tank3*. The model contains a *Controller* class, which has the main thread of control. An instance of the *LevelSensor* class (*sensor*) is created to represent the sensor that measures the current water level, and also

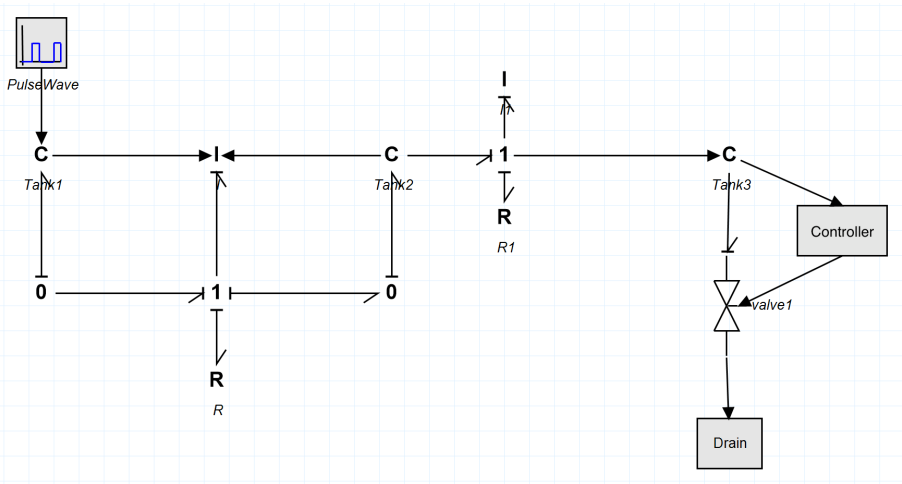


Figure 5: 20-sim CT model of the Three-tank Water Tank system

an instance of *ValveActuator* (*valve*) is created to represent the valve at the bottom of the tank.

The control loop retrieves the current level of water from the sensor and determines whether to set the valve to be open or closed depending on the level compared to some set maximum or minimum value.

Contract

The contract with the VDM controller consists of: two *shared design parameters* for the minimum and maximum levels of *Tank3* – `wt3_min` and `wt3_max`, both of type `real`; a *monitored* variable representing the current water level in *Tank3* – `wt3_level` of type `real`; and a *controlled* variable representing the valve being either open or closed – `wt3_valve` of type `bool`.

Co-simulation

In performing the co-simulation, we obtain the results of the water tank levels and rate of flow between graphs, as shown in Figure 6. We produce 5 graphs; the top two graphs show the water levels of the three water tanks – and as can be seen, the water level of *Tank1* and *Tank2* rise steadily as water flows into *Tank1* through the inflow. *Tank3* slowly fills once the water level of *Tank2* reaches the height of the outflow port (at approx 2.5 seconds). The water level of *Tank3* increases to slightly above *2.0 meters* when the controller opens the valve (shown in the bottom right graph) reducing the water level to just below *1.0 meter*.

The important feature of this model is revealed in the bottom-left and bottom-center graphs. of Figure 6. Two arcs are shown referring to the flow of water between the tanks. The nature of these flows differ widely due to way in which they are connected. The flow between *Tank1* and *Tank2* (bottom-left) has a high frequency with a large amplitude, in contrast to the flow between *Tank2* and *Tank3* (bottom-center) which has a steady flow.

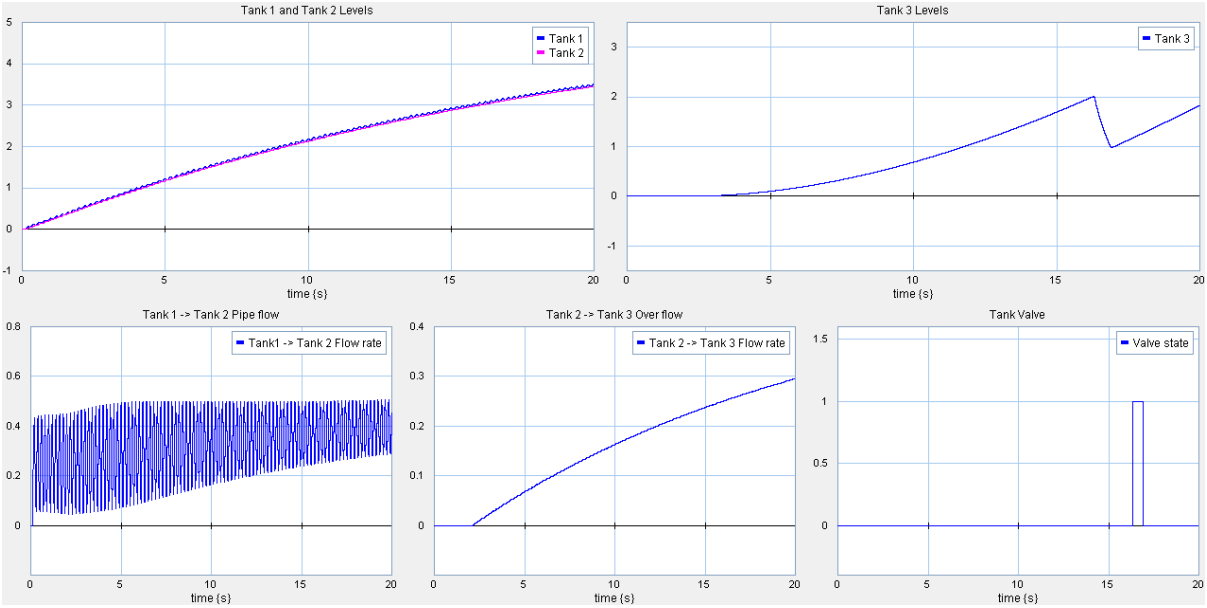


Figure 6: Co-simulation results of the Three-tank Water Tank system

2.3 INTO-CPS Technology

We demonstrate the use of the INTO-CPS SysML profile in Section 2.3.1. Based upon the design architecture defined using the SysML profile, a multi-model is constructed in Section 2.3.2 along with the defined connections. The study is also used to demonstrate the Co-Simulation Orchestration Engine (COE) in Section 2.3.3.

2.3.1 INTO-CPS SysML profile

A SysML model produced using the INTO-CPS profile comprises two diagrams and focusses on the structure of the water tank model for multi-modelling; an Architecture Structure Diagram and a Connections Diagram.

The Architecture Structure Diagram (ASD) in Figure 7 shows the system composition in terms of component subsystems from the perspective of multi-modelling. This model provides an example of a multi-model which does not have a subsystem composition in line with the nominal SysML model in Section 2.2.1. Each of the constituent component blocks identified in Figure 2 is present in the ASD, with the addition of subsystem components.

In this Water Tank system model, the water tanks are split between two subsystems: *WaterTanks1* subsystem contains the *Source*, two *Water Tank* and *Pipe* components; *WaterTanks2* subsystem comprises a single *Water Tank* and *Drain* components; a final cyber component *Controller* contains no other components.

The two water tank subsystems are defined as continuous time models, both with 20-sim as the target platform. The controller component is a VDM-RT discrete event model.

The Connections Diagram (CD) in Figure 8 defines connections similar to those in the IBD in Figure 3. The main changes concern the addition of subsystem components. The

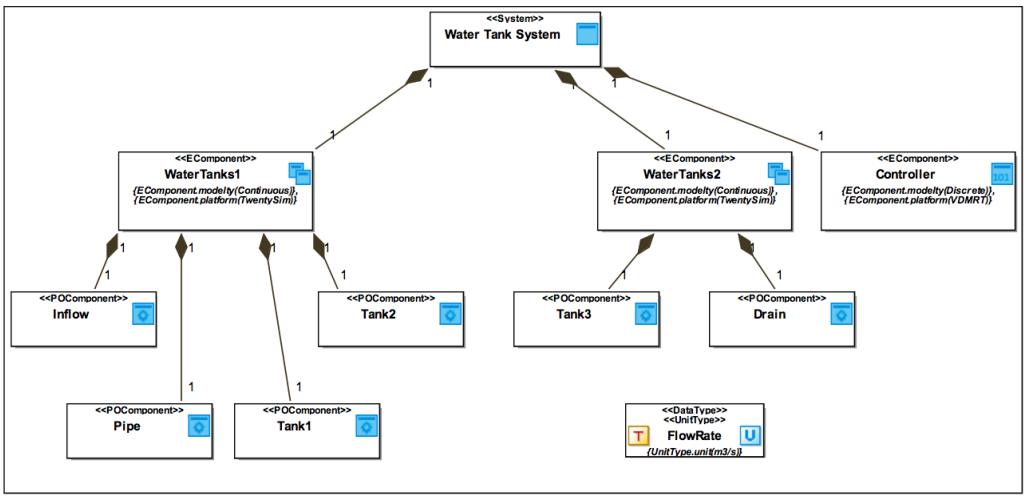


Figure 7: Architecture Structure Diagram defining the Three-tank Water Tank system composition

overall flow of water between physical components remains the same, with the output of *wt2* constituting the output of the *WaterTanks1* subsystem and the input of the *WaterTanks2* subsystem flowing to the input port of *wt3*. The final change concerns the connection between the controller *c* and *wt3*. Service-based interfaces are not included in this version of the INTO-CPS profile, and therefore flow ports are used.

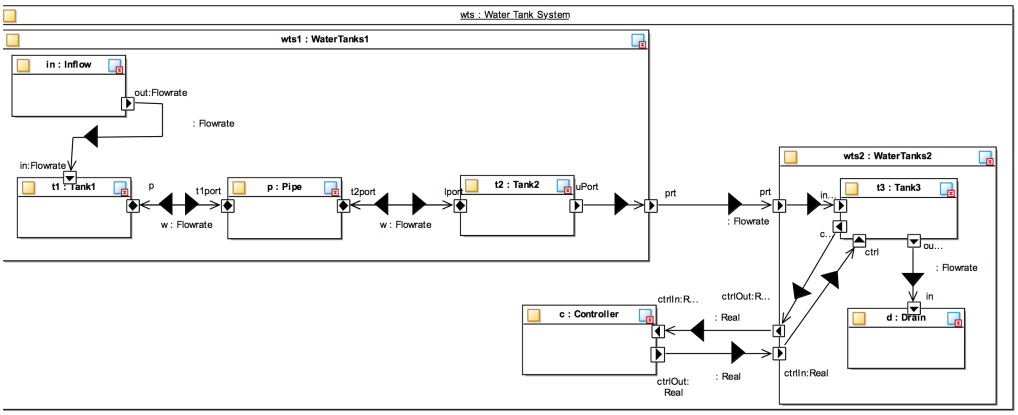


Figure 8: Connections Diagram defining the Three-tank Water Tank system connections

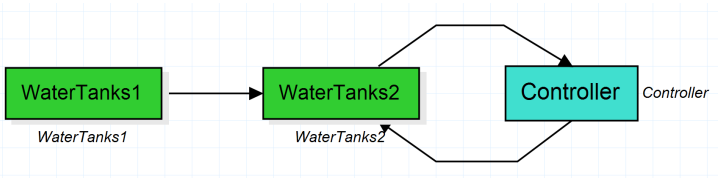
2.3.2 Multi-model

Models

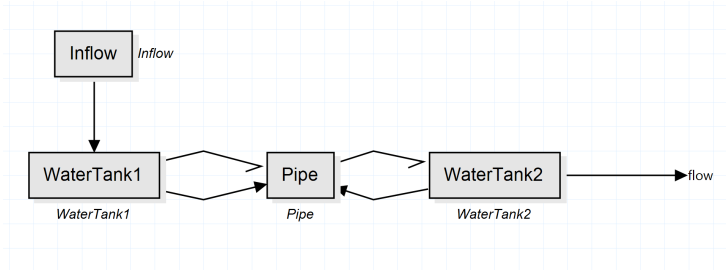
The multi model, corresponding to the SysML model in Section 2.3.1, comprising 2 20-sim subsystems and a VDM subsystem. We partition the original 20-sim model described in Section 2.2.2 to reflect this as shown in Figure 9(a).

The partitioning of the 20-sim model is straightforward, with a single signal between the two 20-sim subsystems representing the flow of water between tanks 2 and 3. The rationale behind this split is that, as observed in Figure 6, the flow rate between tank 1

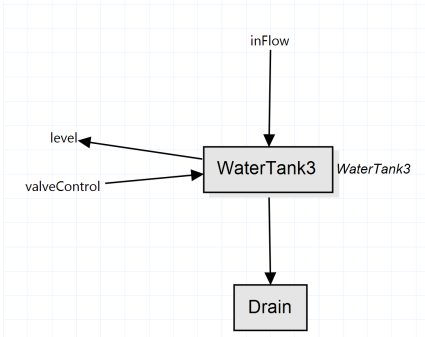
and 2 has a high frequency and amplitude, suggesting that splitting the two tanks would result in erroneous results when time steps are imposed in co-simulation. It is left for the next 12 months of the project to investigate the effect of different COE step sizes on the simulation results.



(a) Subsystems of Three-tank Water Tank multi-model



(b) WaterTanks1 subsystem



(c) WaterTanks2 subsystem

Figure 9: 20-sim models for the Three-tank Water Tank multi-model

The VDM-RT controller model is unchanged from the original Crescendo controller.

Configuration

There are two connections in the multi-model; between *WaterTanks1* and *WaterTanks2*, and between *WaterTanks2* and the *Controller*.

The first connection connects the `flow` port of *WaterTanks1* to the `inFlow` of *WaterTanks2*.

The second connection mirrors the contract in the original Crescendo model. There are two *shared design parameters* – `wt3_min` and `wt3_max`, both of type `real`; a connection from the `valveControl` port of the *WaterTanks2* model to the `wt3_valve` of the *Controller*; and a connection from the `wt3_level` of the *Controller* to the `level` port of *WaterTanks2*.

2.3.3 Co-simulation

Using the INTO-CPS Co-simulation Engine (COE), we may simulate the three FMU multi-CT model. In the current version of the COE, only input/output values of FMUs may be logged. As such, we are not able to observe the level of tanks 1 or 2, or the flow rate between tanks 1 and 2. We are able to log the water level of tank 3 and the flow rate between tank 2 and 3. These values are shown in the graph in Figure 10, using a fixed step size of *0.00001*.

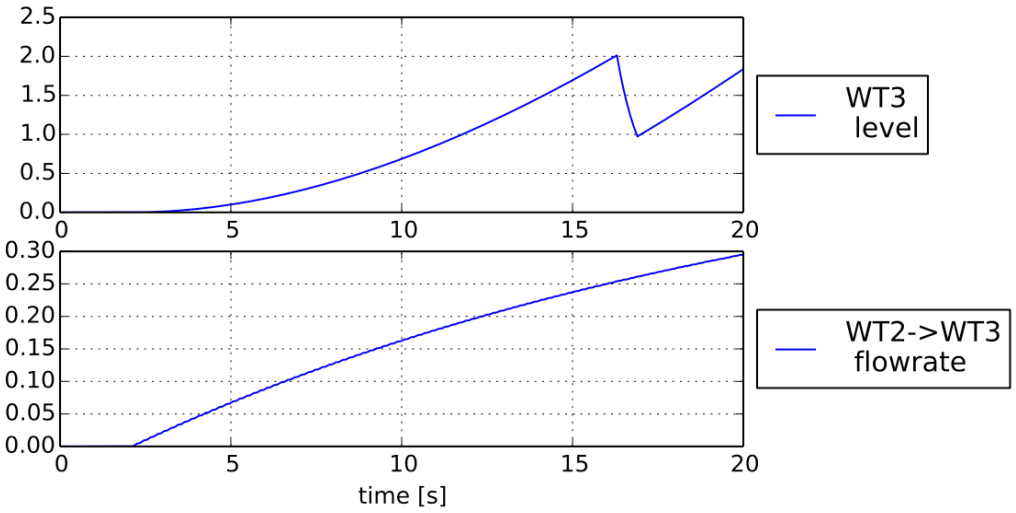


Figure 10: Simulation results using the INTO-CPS COE

The results in the graph correspond closely to those of the baseline Crescendo model illustrated in Figure 6. During simulation, the water level raised to the maximum value (2.0 meters) and at 16.3 seconds the tank 3 valve is opened by the VDM-RT controller and the level drops to just below the minimum (1.0 meters) and at 16.9 seconds the valve is closed and the water level begins to rise again.

3 Fan Coil Unit (FCU)

3.1 Example Description

This example is inspired by the Heating Ventilation and Air Conditioning (HVAC) case study developed in Task T1.3. The Fan Coil Unit (FCU) aims to control the air temperature in a room through the use of several physical components and software controllers. Water is heated or cooled in a *Heat Pump* and flows to the *Coil*. A *Fan* blows air through the *Coil*. The air is heated or cooled depending upon the *Coil* temperature, and flows into the room. A *Controller* is able to alter the fan speed and the rate of the water flow from the *Heat Pump* to the *Coil*. In addition, the room temperature is affected by the walls and windows, which constitute the environment of the FCU.

The aim of the system is to maintain a set temperature in the single room in which the FCU is located. The system is outlined in Figure 11.

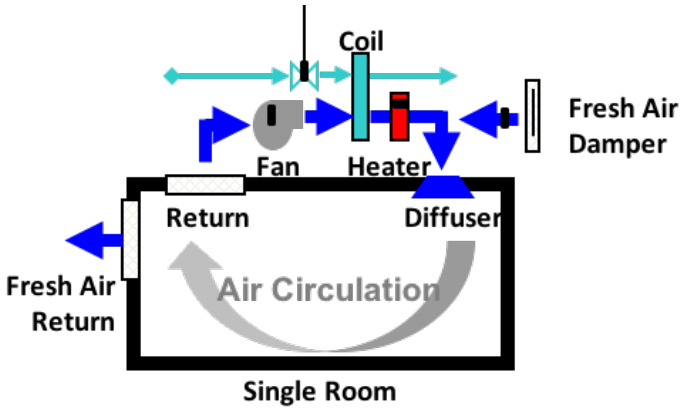


Figure 11: Overview of the fan coil unit (FCU) example

3.2 Baseline

In the FCU study, we demonstrate the use of the OpenModelica (Section 3.2.1) and Crescendo (Section 3.2.2) baseline technologies.

3.2.1 OpenModelica

This example was initially presented as an OpenModelica model with a CT controller defined in the OpenModelica notation.

Utilising PID control in Figure 12, the aim of the model is to provide smooth control of the room temperature. This model has several blocks: *OAT* — a table of timed outside air temperature values; *RATsp* — the room temperature set point; *Wall* — modelling the thermal model for the walls inside the room; *Room* — modelling the thermal model for a room; and *PID* that provides PID control for the *fanspeed* and *valveopen* values input into the *Room*.

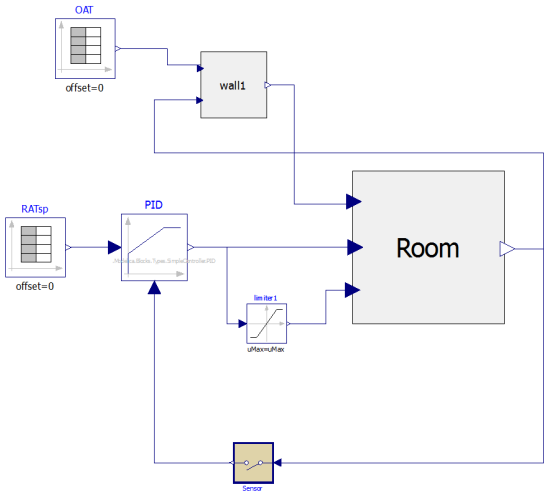


Figure 12: PID control FCU OpenModelica model

Simulating the model over a 7 day period we obtain the graph shown in Figure 13. The room set point (purple line) fluctuates between 16 and 21 degrees as the room users require the room to be heated. The room temperature, RAT, given by the red line rises to the set point of 21 and maintains that temperature until the set point drops. The heating is shown by the blue line, indication the valve is open and fan is working. The room temperature drops then both the valve is closed and fan has stopped. It should be noted that the temperature never reaches the minimum set point due to the density and thermal conductivity of the wall. The graph also displays the outside air temperature (the green line), which varies considerably over time.

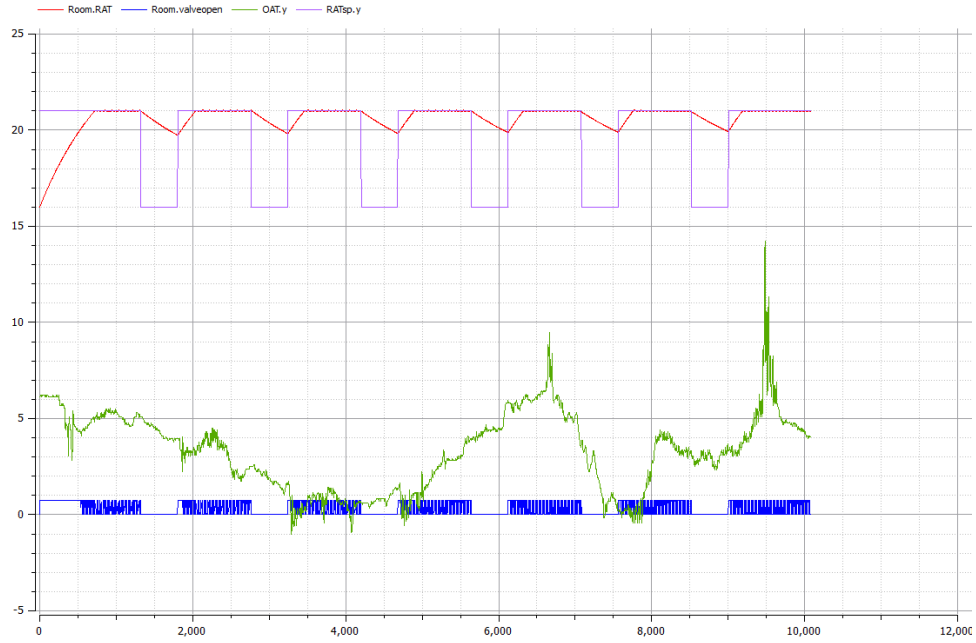


Figure 13: Simulation results for the FCU OpenModelica model

3.2.2 Crescendo

CT Model

A Crescendo co-model was developed that corresponds to the OpenModelica PID control model. The 20-sim model outlined in Figure 14 has a similar structure to the OpenModelica version in Figure 12. The 20-sim model has two main submodels: the *Wall* and *Room*, a table of outside air temperatures and a *Controller* block. The *Wall* and *Room* elements are defined as systems of ordinary differential equations.

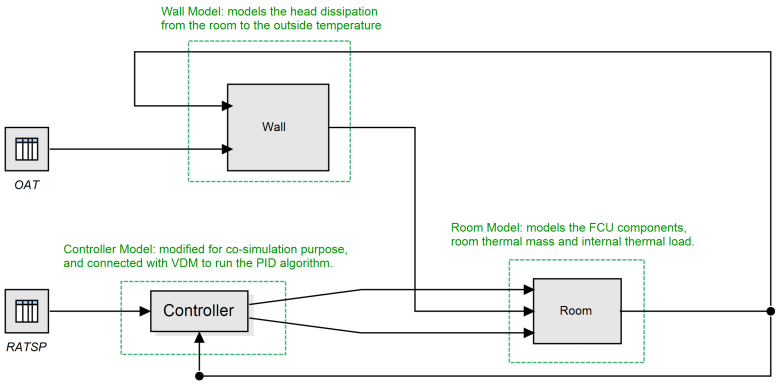


Figure 14: Crescendo FCU model

DE Model

In the baseline Crescendo model, the VDM-RT controller corresponds to the PID controller of the OpenModellica model in Section 3.2.1. Supporting the controller class, a *Sensor* class provides access to the current room temperature, and a *LimitedActuator* class provides output for the valveOpen and fanSpeed values. The actuator is limited such that values fall only between the real values 1.0 and 0.0000001.

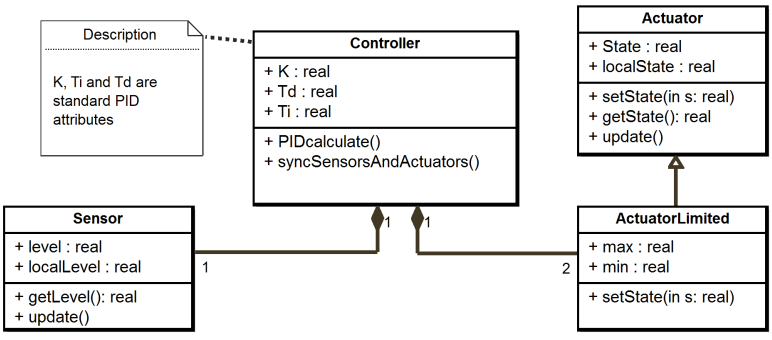


Figure 15: FCU DE model class structure

Contract

The contract between CT and DE models specifies that the VDM-RT controller monitors the `SetPoint` and `RoomTemp` values from the 20-sim model, and controls the `Control`

variable used by the *Room* block, which corresponds to the fan speed and valve open setting.

Co-simulation

Figure 16 shows the outputs of the co-simulation (generated by the 20-sim tool during a co-simulation run). Traces of the set point and RAT (top left) show the variation in RAT as the OAT (bottom left) changes. The upper right shows (rather extreme) actions of the control algorithm. The graph at bottom right shows the cumulative energy output from the FCU as the co-simulation proceeds.

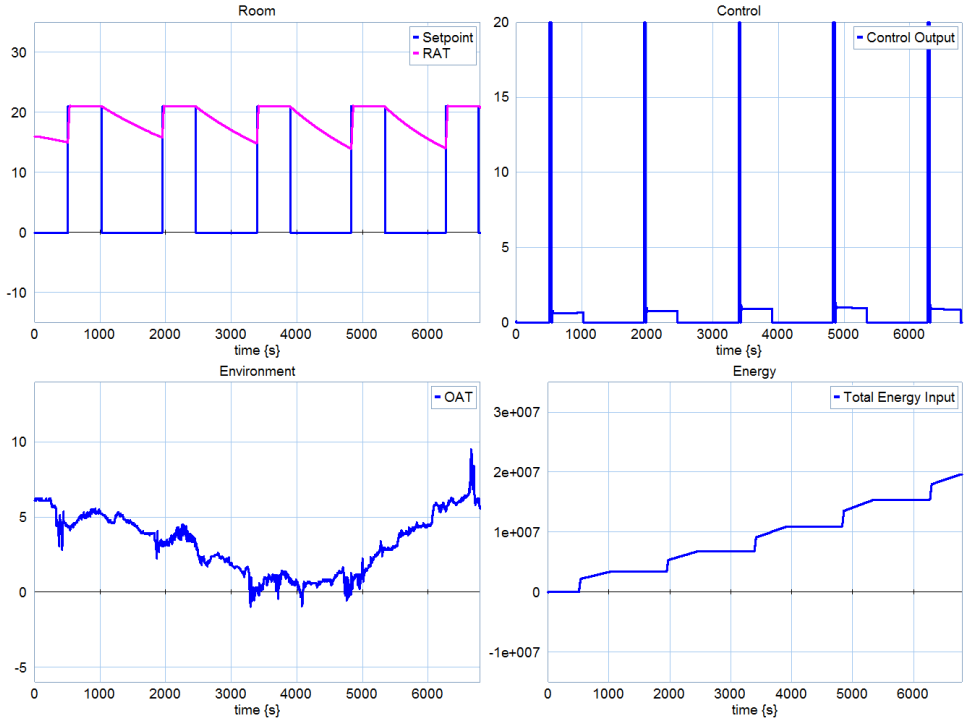


Figure 16: Co-simulation output from FCU co-model

Design Space Exploration can be performed by running co-simulations that sweep over the design parameters. We may sweep over a physical parameter such as the wall's thermal conductivity, calculating energy consumption. If combined with a model of energy and construction costs, we can examine overall cost of ownership for the room with the heating controller for a given period of operation. Figure 17 shows one such analysis, illustrating how the cost of a low conductivity will be outweighed by energy savings when taken over a longer ownership period.

3.3 INTO-CPS Technology

The demonstration on INTO-CPS technologies with the FCU example concentrates on the INTO-CPS SysML profile, shown in Section 3.3.1

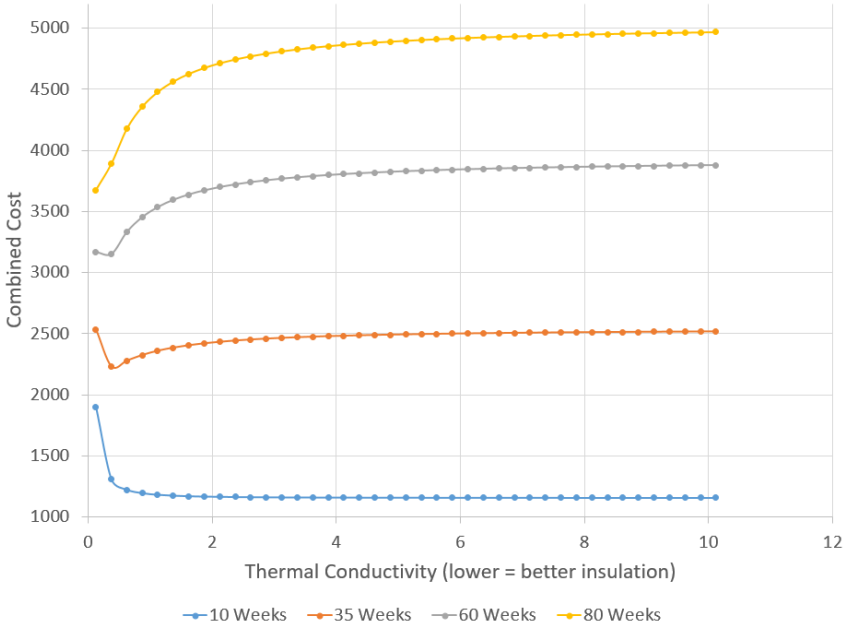


Figure 17: DSE Result: cost against thermal conductivity for a range of operating periods.

3.3.1 INTO-CPS SysML Profile

Co-model Version

Two versions of the FCU model are defined using the INTO-CPS SysML profile. The first corresponds to the architecture used in the baseline OpenModelica and Crescendo models. In this version, three constituent parts are defined – shown in Figure 18: the *RoomHeating* subsystem, a *Controller* cyber component and the physical *Environment*. The first is a continuous subsystem and comprises the *Room* and *Wall* components. The figure defines the model platform to be 20-sim, however, this could be OpenModelica too. All of the physical elements of the system are contained in a single CT model. The controller subsystem is a cyber element and modelled in VDM-RT.

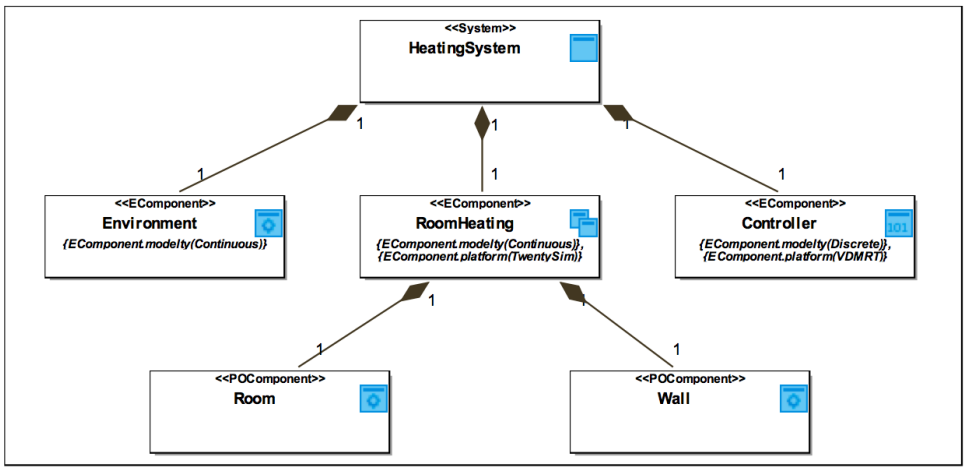


Figure 18: SysML Architecture Structure Diagram using INTO-CPS profile corresponding to baseline models

The connections between components, shown in Figure 19, are similar to those in the baseline CT models, although it should be noted that the subsystem hierarchy is shown, with the *Room* component supplying and receiving the flows of the *RoomHeating* subsystem. The connections between CT and DE models show the interface that is managed during the co-simulation. Specifically, the room air temperature (*RAT*) from the CT system is communicated to the controller, which sets the fan speed *fanSpeed* and the valve open state *valveOpen* used by the Room component model *r*, with the aim of achieving the room air temperature set point *RATSP* provided by the user in the *Environment*.

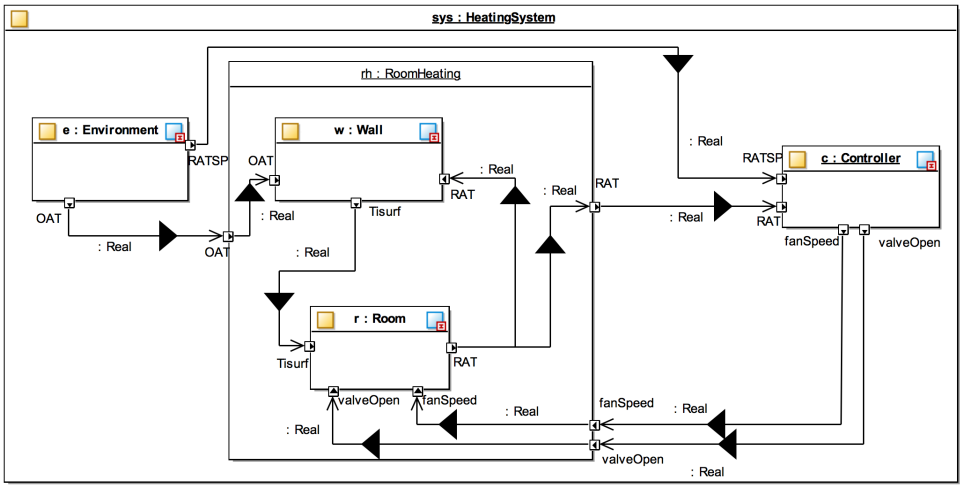


Figure 19: SysML Connection Diagram using INTO-CPS profile corresponding to baseline models

Multi-model Version

Moving to a more 'pure' multi-modelling approach, the next model proposes an alternative subsystem structure. In this model, the ASD in Figure 20 shows the HeatingSystem comprises four subsystems; the components comprising the *RoomHeating* subsystem in Figure 18 are lifted to be top-level components in their own right.

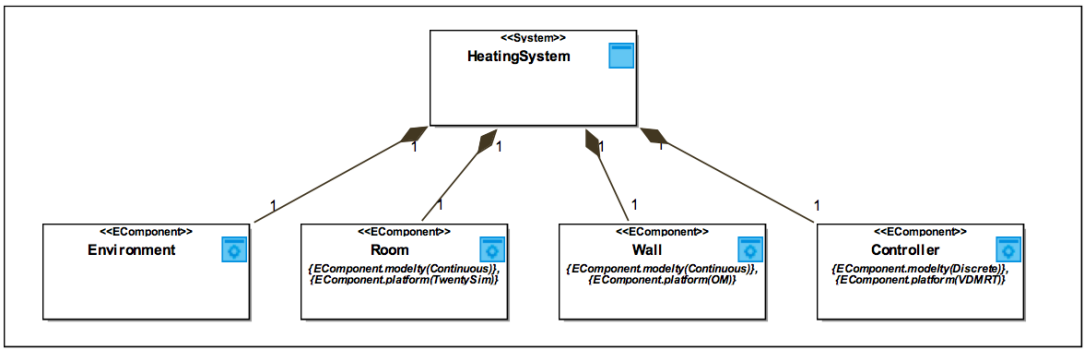


Figure 20: SysML Architecture Structure Diagram using INTO-CPS profile corresponding to 'pure' multi-model approach

This is reflected in the CD in Figure 21, with direct connections between the elements. Each of the CT components (*Room*, *Wall* and *Environment*) may now be modelled in different notations.

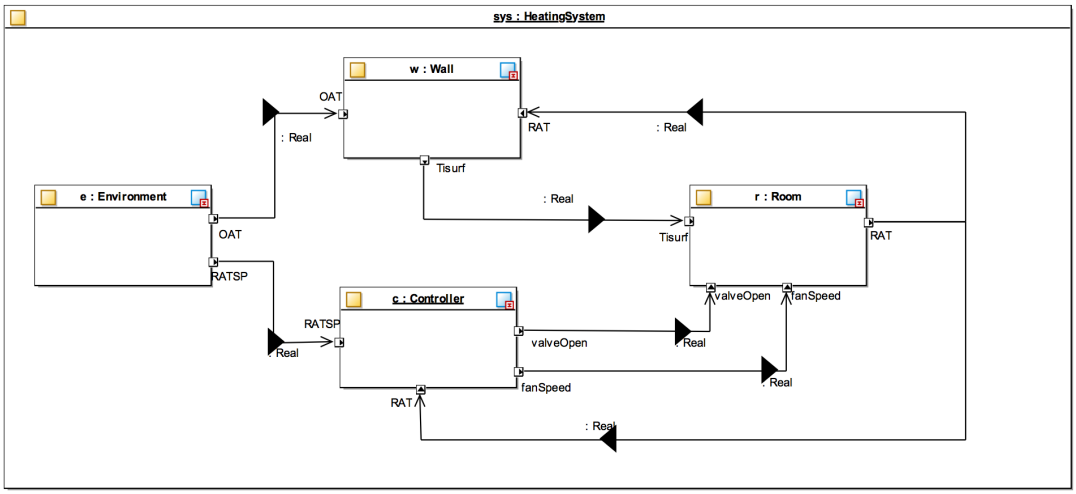


Figure 21: SysML Connection Diagram using INTO-CPS profile corresponding to 'pure' multi-model approach

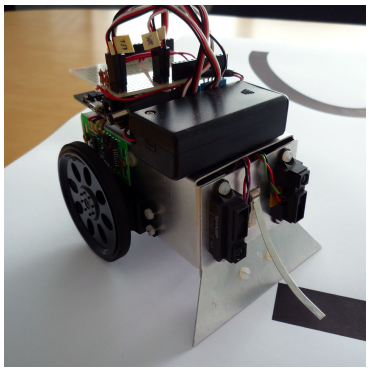
4 Line-following Robot

4.1 Example Description

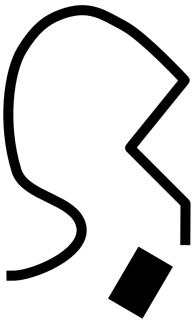
This example, originally developed in the DESTECS project and presented in [IPG+12]. The model simulates a robot that can follow a line painted on the ground. The line contrasts from the background and the robot uses a number of sensors to detect light and dark areas on the ground. The robot has two wheels, each powered by individual motors to enable the robot to make controlled changes in direction. The number and position of the sensors may be configured in the model. A controller takes input from the sensors and encoders from the wheels to make outputs to the motors.

Figure 22 provides an overview of different aspects of the example: the real robot; an example path the robot will follow; and a 3D representation in 20-sim.

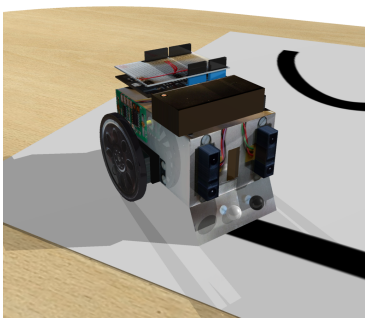
The robot moves through a number of phases as it follows a line. At the start of each line is a specific pattern that will be known in advance. Once a genuine line is detected on the ground, the robot follows it until it detects that the end of the line has been reached, when it should go to an idle state.



(a) A line-following robot



(b) A line-follow path



(c) 3D representation of the line-following robot

Figure 22: The line-following robot

4.2 Baseline

In the line-following robot study, we demonstrate the use of the Crescendo (Section 4.2.1), OpenModelica (Section 4.2.2) and SysML (Section 4.2.3) baseline technologies.

4.2.1 Crescendo

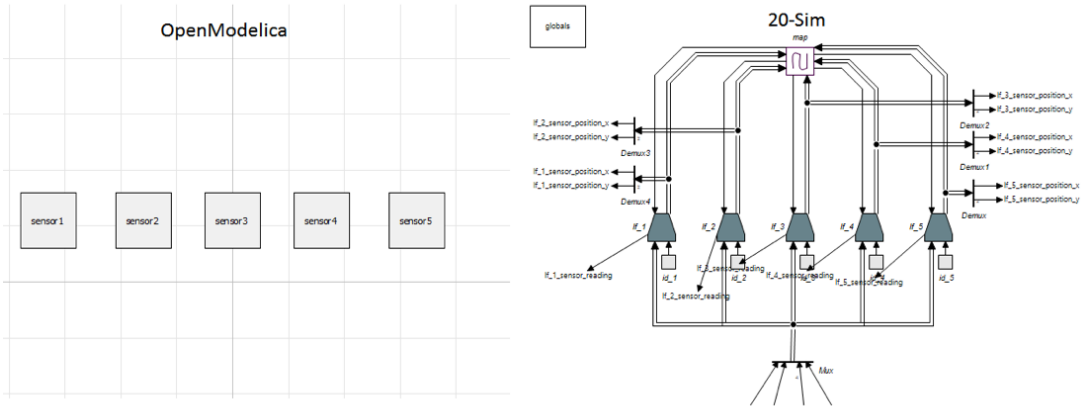
The line-following robot model was initially developed in the DESTECS project, and detailed in [IPG+12]. The Crescendo robot model comprises a 20-sim body with sensors, motors, actuators and wheels, and a VDM-RT model controlling the speed and direction of the motors based upon the sensor input. The model is a co-model of an embedded system, in that this division of modelling has a DE controller and CT plant model.

The model description is not given here, more information about the example (the Crescendo contract, a 20-sim CT model and a VDM-RT model) is provided in the DESTECS example compendium [IPG+12]. In addition, the DESTECS report describes uses of DSE with this example.

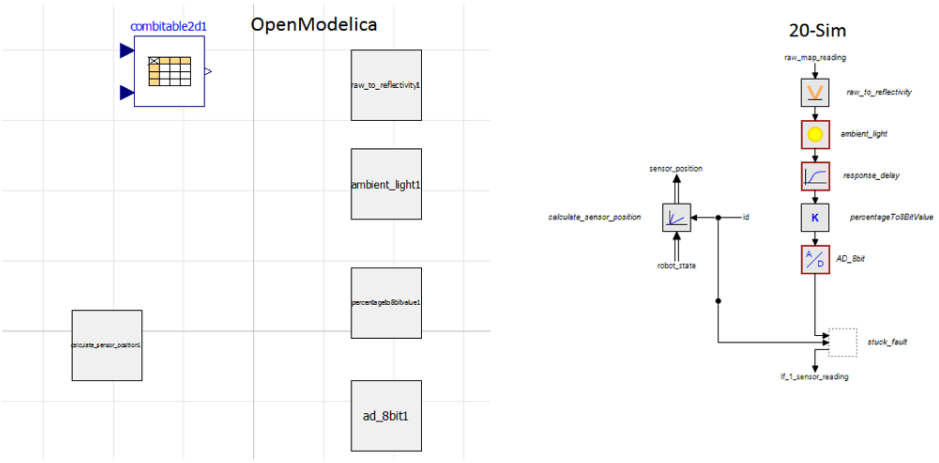
4.2.2 OpenModelica

The OpenModelica model "LineFollowerSensor" is a direct translation from the 20-Sim collection of sensors. The purpose of this model is to simulate the sensors of the line following robot in place of the 20-Sim version.

The structure between both versions of the sensors is almost identical. As seen in Figure 23, the connections from inputs and outputs are not visible in the OpenModelica version; however they are still there programmatically. A notable difference in this version is the location of the "map" block, which now resides in each sensor individually, and not as its own block (as seen in the 20-Sim model).



(a) A comparison between the OpenModelica and 20-Sim LineFollowerSensors model structure



(b) A comparison between the OM and 20-Sim sensor structure

Figure 23: Modelling line-following sensors in OpenModelica

The Sensor block in OM contains blocks from the 20-Sim version translated into OM. The

"response_delay" and "stuck_fault" blocks that are present in the 20-Sim implementation have been omitted in OM due to technical issues; however these blocks do not affect the final signal output. OM uses a different method for noise generation than 20-Sim in the "AD_8bit" block, this currently results in a different set of ranges for the noise signal output (Figure 24).

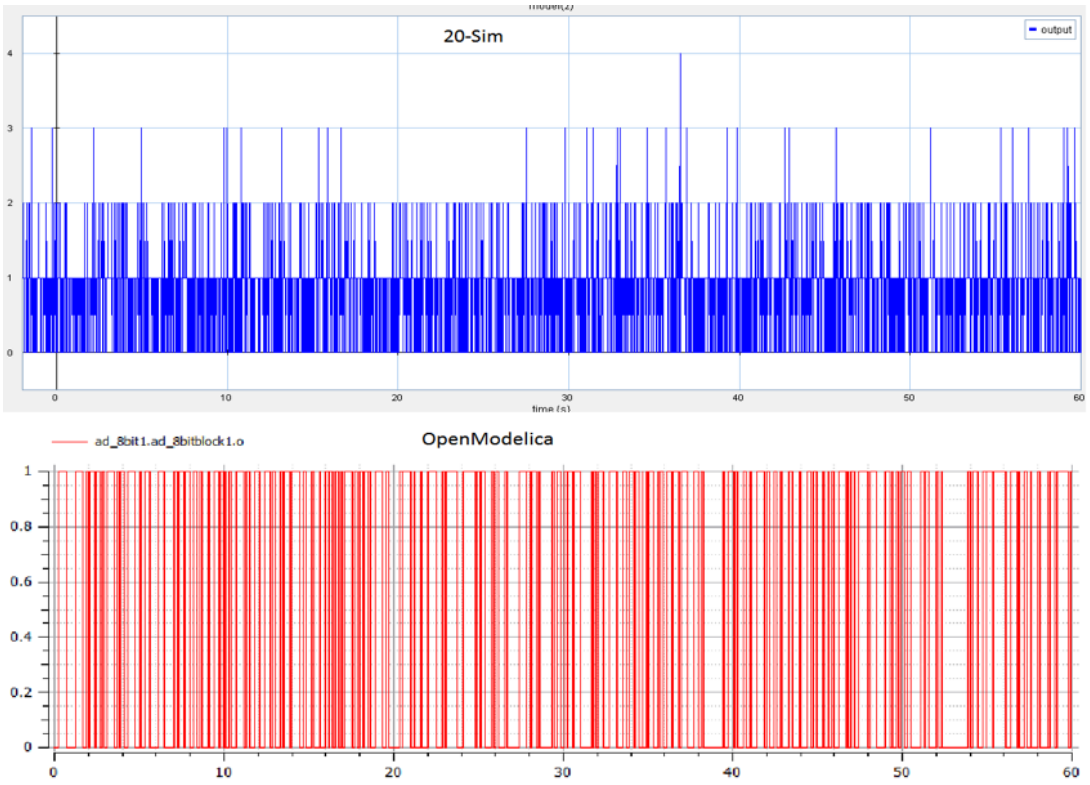


Figure 24: Different ranges between noise signal outputs

A set of tests were performed to check the output of both models. To do this the same input was introduced into both models to make sure the line was detected within the map.

The input introduced corresponds to a constant value for the y-axis (0), and an integrated value for the x-axis (0.01-0.6). The purpose of this is to sweep across the map to detect the line. The outputs for both versions are shown for OM and 20-Sim (Figure 25), in red and blue respectively.

4.2.3 SysML

An architectural description of the robot is produced in SysML, taking a broadly similar structure as the 20-sim block diagram. The SysML model comprises diagrams describing the system structure and behaviour.

Two structural diagrams are defined: in Figure 26 a BDD defining the system composition and Figure 27 an IBD identifying the connections between the components. The BDD corresponds to the 20-sim block diagram, stating that the robot comprises several components, with varying cardinality. The BDD also includes the specification of the

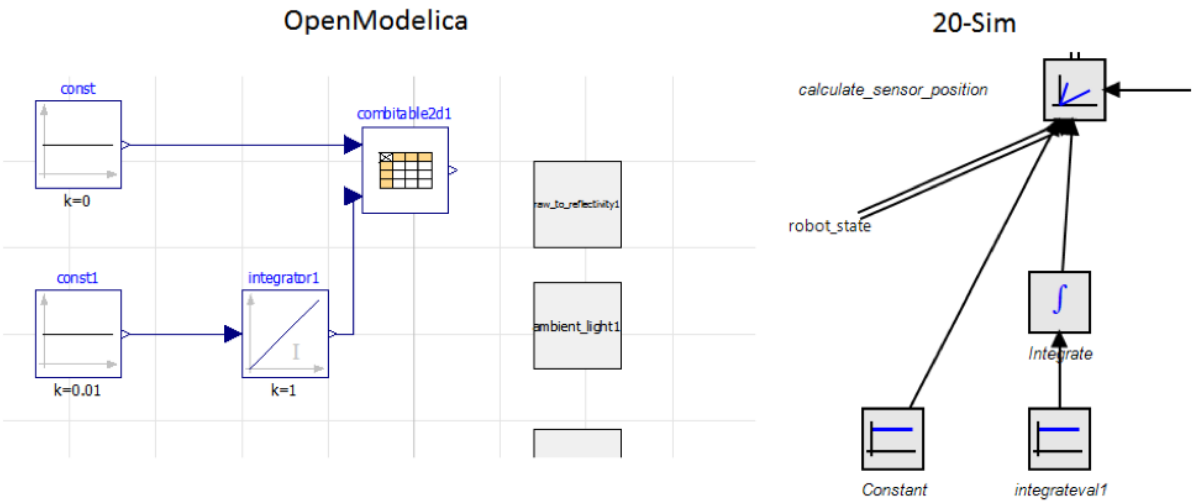


Figure 25: Test input introduced into both the OM and 20-Sim Sensor model block

robot's environment, comprising the line to be followed. The IBD of the system in Figure 27 describes the flow of signals (using real numbers) and different forces (rotational force from the motors and transitional force between wheels and the body).

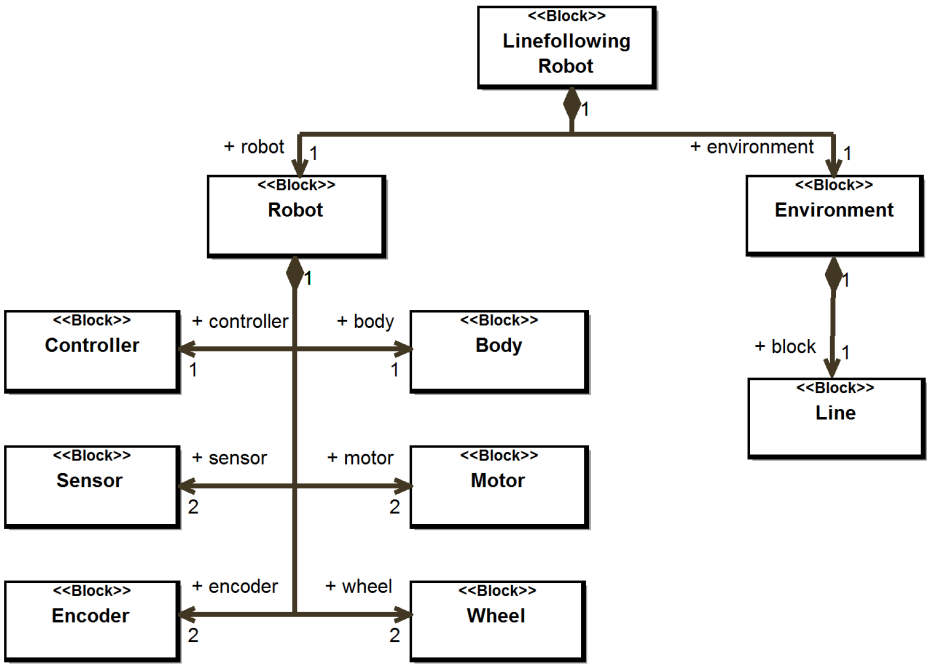


Figure 26: BDD of the line-following robot SysML model

The behaviour of the controller is defined in a state machine diagram (STD) in Figure 28. This diagram specifies the state transitions from calibrating the robot, reading sensors and determining directions to move the robot. The state machine is decomposed in Figure 29 to specify the order in which the sensors are read and how the sensor readings relate to the values of the model.

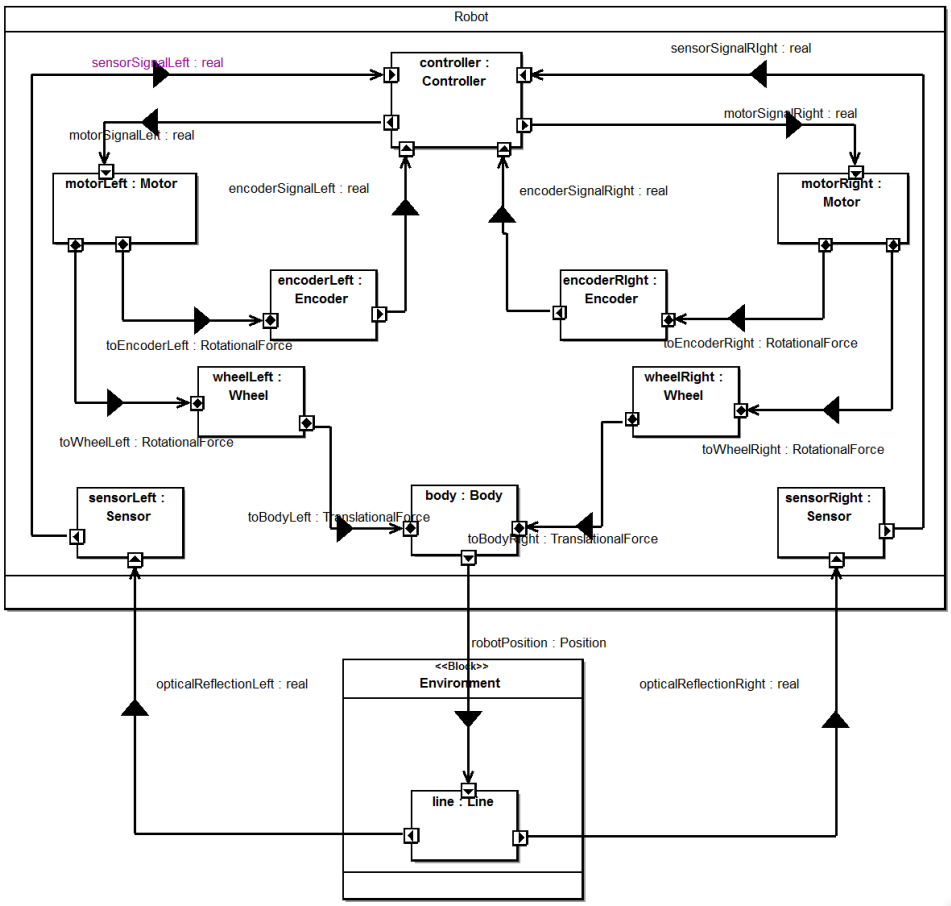


Figure 27: IBD of the line-following robot SysML model

Finally, the model includes diagrams to define the continuous behaviour of the robot architecture. In Figure 30, a BDD defines a collection of constraint equations regarding force. These equations are then linked in a parametric diagram (PD) in Figure 31 to define the overall continuous behaviour of the system.

4.3 INTO-CPS Technology

We demonstrate the use of the INTO-CPS SysML profile in Section 4.3.1. Based upon the design architecture defined using the SysML profile, a multi-model is constructed in Section 4.3.2 along with the defined connections.

4.3.1 INTO SysML profile

The multi-model architecture, defined in the INTO-CPS SysML profile, splits the original SysML model into three subsystems, as shown in the Architecture Structure Diagram in Figure 32. This version comprises *BodyParts* and *SensorParts* subsystems and a *Controller* cyber component. The *BodyParts* subsystem, with a target platform as 20-sim as a CT model, comprises the Body, Encoder, Wheel and Motor components. The *SensorParts* subsystem contains only Sensor components. The *Controller* component remains the same as the original SysML model.

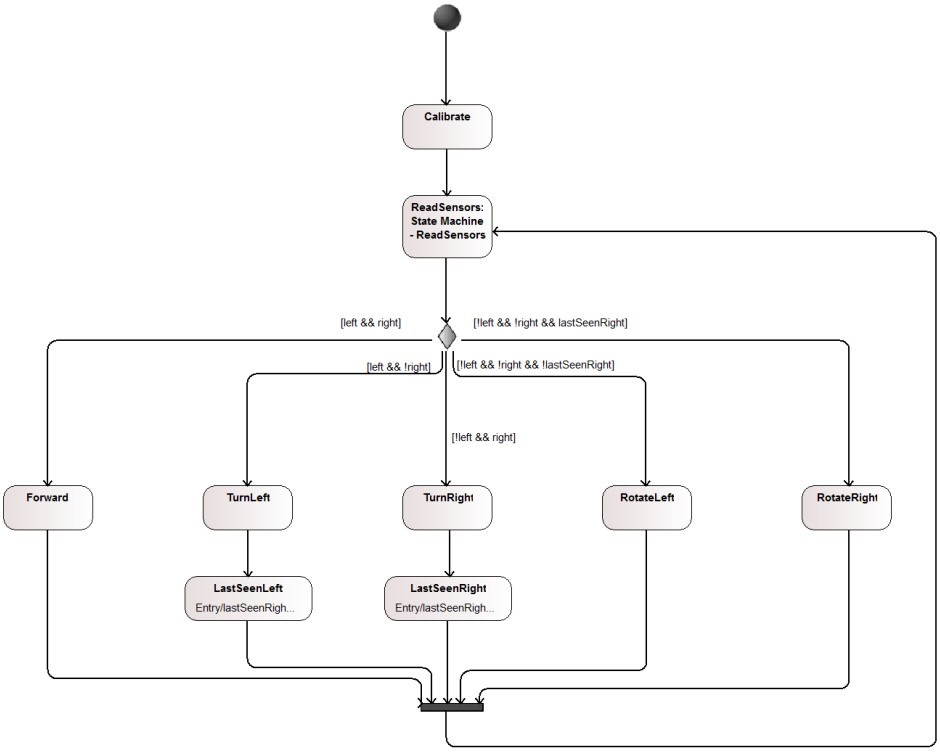


Figure 28: State Machine Diagram 1 of the line-following robot SysML model

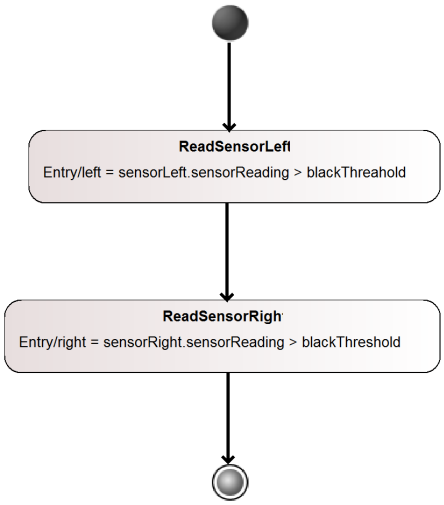


Figure 29: State Machine Diagram 2 of the line-following robot SysML model

The connections between components remain largely the same as the nominal SysML model, as shown in Figure 33, with the same value types flowing between components. Connections are made between the different subsystems, and the underlying components realising the source and destination of values.

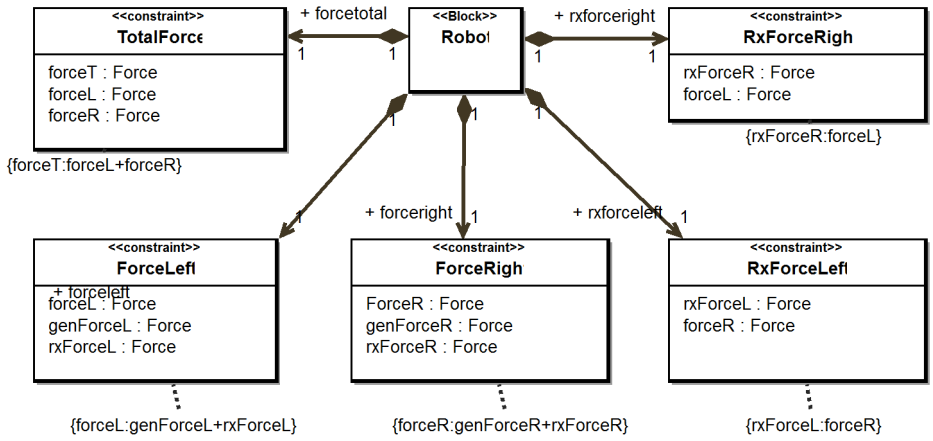


Figure 30: BDD defining constraint equations of the line-following robot SysML model

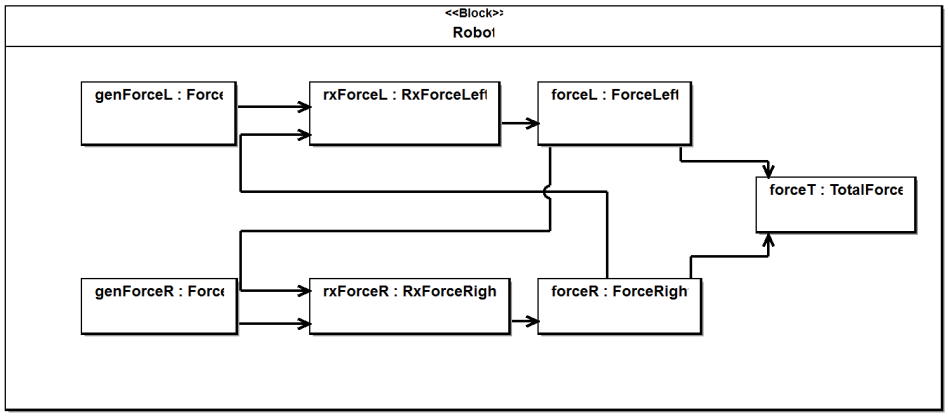


Figure 31: Parametric Diagram defining constraint relationships of the line-following robot SysML model

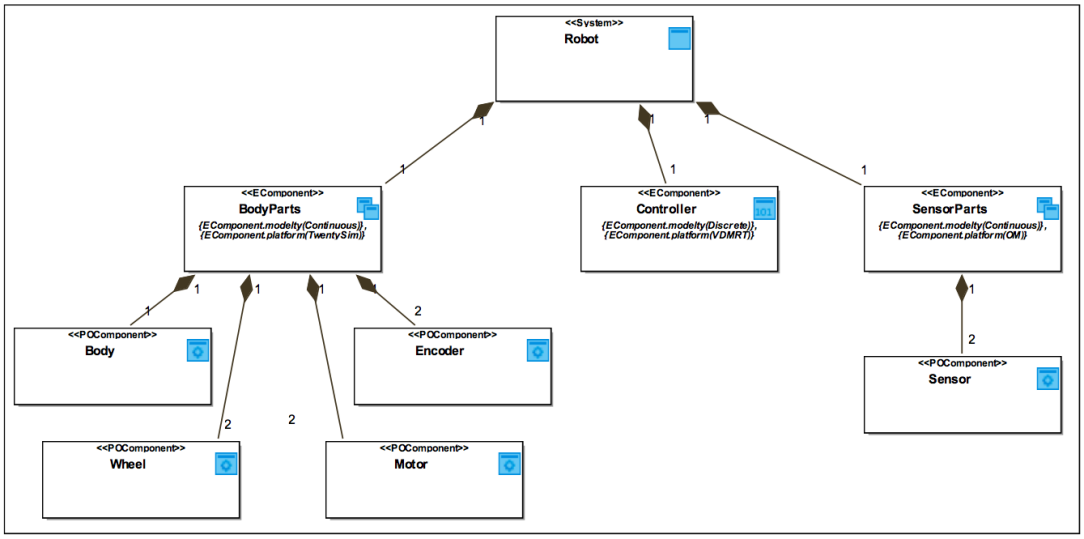


Figure 32: The line-following robot Architecture Structure Diagram

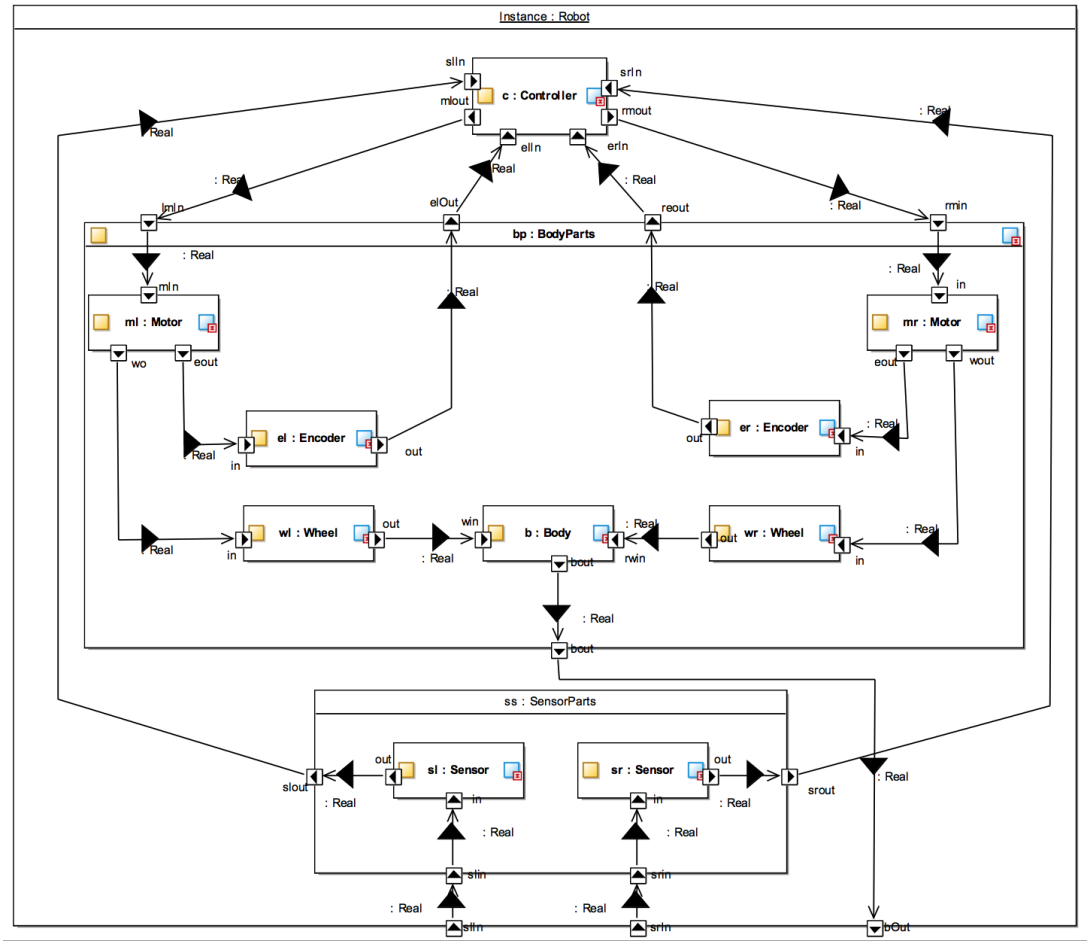


Figure 33: The line-following robot Connections Diagram

4.3.2 Multi-model

Models

The multi-model produced and analysed using INTO-CPS technology stems from the baseline Crescendo co-model. The multi-model comprises 3 models, splitting the Crescendo model as depicted in Figure 34. This example, therefore is a multi-CT model, with a single DE model.

The Crescendo elements (a VDM-RT controller and a 20-sim plant) are largely unchanged, modified so that the 20-sim model is partitioned into 2 high-level submodels: a *body* and the body's *environment*. This environment block corresponds to the other parts of the system to be modelled in other notations. By modelling in this way, each submodel can be exported as a separate FMU. This is shown in Figure 35.

For the purposes of multi-modelling, we concentrate on the 20-sim Body subsystem which does not contain the sensors, as shown in Figure 36. In their place, the body submodel contains ports for the robot position: *robot_x*, *robot_y*, *robot_z* and *robot_theta*. The other ports are the same as in the baseline Crescendo model (*wheel_left_location*, *wheel_right_location*, *total_energy_used*, *servo_left_input* and *servo_right_input*) for interfacing with the controller and visualisation models.

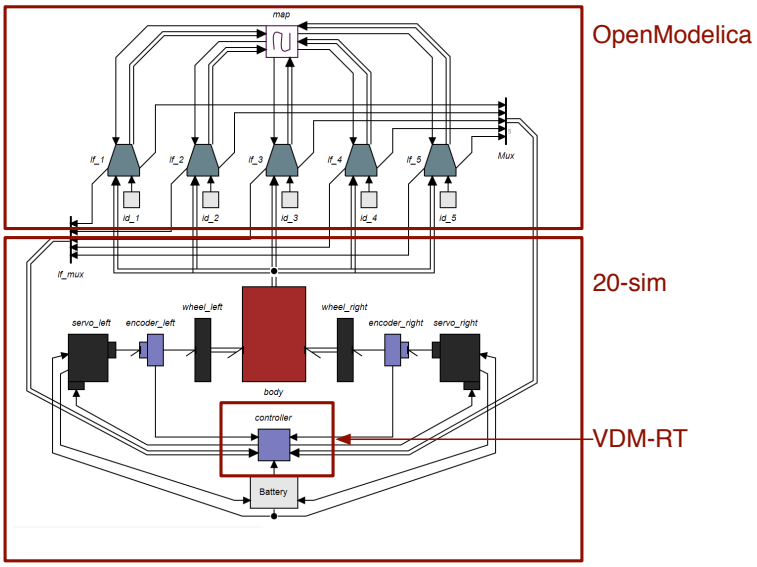


Figure 34: Splitting the line-following robot Crescendo model

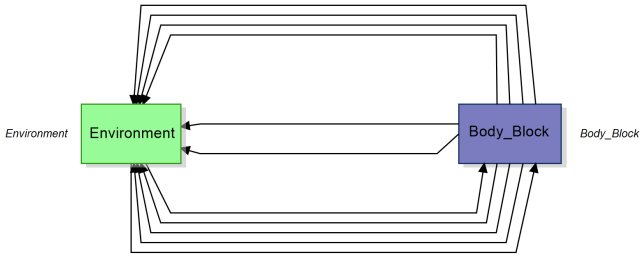


Figure 35: Modified 20-sim model of the line-following robot

The VDM-RT controller model is unchanged from the original Crescendo controller.

The OpenModelica model produced for the line-following robot concentrates on describing a collection of robot components; the sensors. This model is described in the previous section.

At present, it is unclear where the robot's environment – that is the map to follow – is modelled. It could be modelled as a part of an existing model or as a separate FMU. Experiments are required to determine this.

Configuration

There are several connections between the models in the multi-model.

The first collection of connections is between the Body 20-sim model and the Controller VDM-RT model. In this collection, there are several connections corresponding to signals for the actuators that power the motors for the wheels, and feed back information about the rotations of the wheels:

- from the *Controller* `servo_left` variable of type real to the `servo_left_input` port of the *Body*;

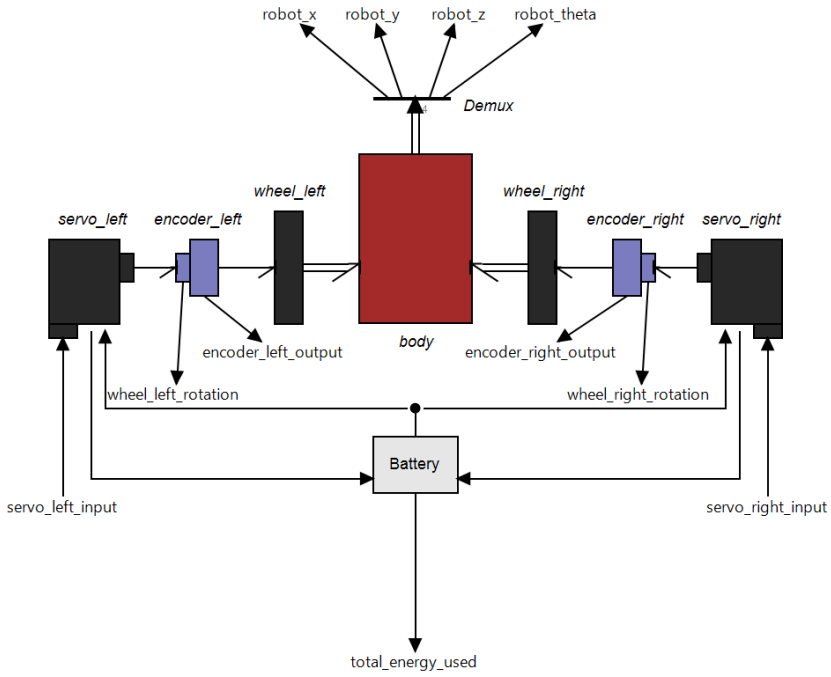


Figure 36: 20-sim model of the line-following robot body

- from the *Controller* `servo_right` variable of type real to the `servo_right_input` port of the *Body*;
- from the *Body* `encoder_left_output` port to the `encoder_left` variable of the *Controller*; and
- from the *Body* `encoder_right_output` port to the `encoder_right` variable of the *Controller*.

This connection also includes seven shared design parameters: wheel radius, in metres (`wheel_radius`); encoder resolution, in counts per revolution (`encoder_resolution`); and two shared variables representing controller aggression in terms of the ratio betwen target and maximum wheel speed (`fast_wheel_speed`) and the turn ratio (`slow_wheel_ratio`), both in the range [0,1].

The second collection of connections is present between the Sensor OpenModelica model and the Controller VDM-RT model. For each sensor there is one connection to the controller to represent inputs from line-following sensors that can detect areas of light and dark on the ground. Therefore for a two-sensor model there are two connections:

- from the *Sensor* `sensor_value` port[7] to the *Controller* `lf_left` variable; and
- from the *Sensor* `sensor_value` port to the *Controller* `lf_right` variable.

Two shared design parameters are present also: the separation of the line-following sensors from the centre line, in metres (line_follow_x); and the distance forward of the line-following sensors from the centre of the robot, in metres (line_follow_y).

[7]The `sensor_value` port is defined as an array in the OpenModelica model, which is flattened on FMU export.

A third collection of connections exist between the body and the sensors related to the robot position:

- from the *Body* `robot_x` port to the *Sensor* `robot_state` port[8];
- from the *Body* `robot_y` port to the *Sensor* `robot_state` port;
- from the *Body* `robot_z` port to the *Sensor* `robot_state` port; and
- from the *Body* `robot_theta` port to the *Sensor* `robot_state` port.

Connections will also exist between the *Body* and *Environment* and the *Sensors* and *Environment*. As described above, the nature of the *Environment* is not yet defined, therefore the connections are yet to be determined.

[8]The `robot_state` port is defined as an array in the OpenModelica model, which is flattened on FMU export.

5 Turn Indicator

5.1 Example Description

The turn indicator model discussed here is an adaption of a model that was designed with an industrial partner from the automotive domain[9]. The model specifies the behaviour of a turn indication controller, which essentially supports left and right flashing as well as emergency flashing. The functionality is modelled using three inputs (the voltage, the control lever and the emergency flash button) and two outputs (the states of the left and right turn indication lights, respectively). The model can then be used to automatically generate test cases for a system that shall implement the specified behaviour using the RT-Tester Model-Based Test Case Generator (RTT-MBT).

A key feature of this example is that it combines several features which are important for effective modelling of system specifications using SysML state charts: It uses variables of different types (voltage is real-valued, the other ones are integral), it uses hierarchical state machines and concurrent component. The overall composite structure of this test model in Modelio is depicted in Figure 37.

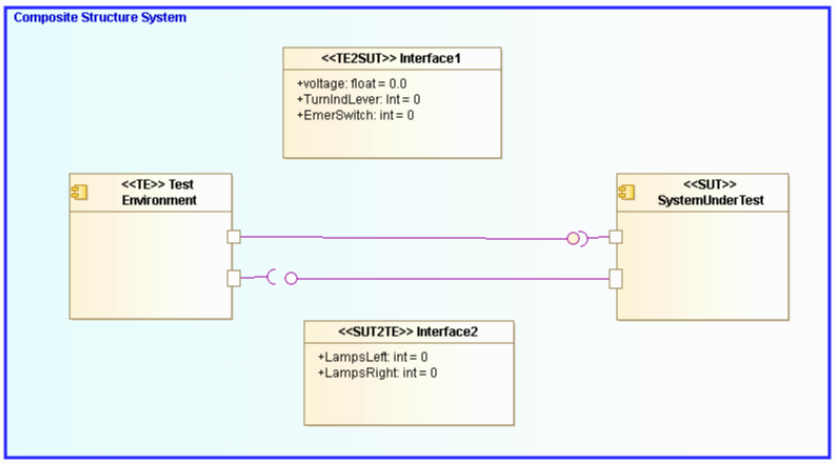


Figure 37: Composite structure diagram of the turn indicator model.

5.2 Baseline

The turn indicator study is primarily used to demonstrate the SysML and RT-Tester baseline technologies, shown in Section 5.2.1.

5.2.1 SysML and RT-Tester

In this section, we concentrate on the structure of the turn indicator model in Modelio. As highlighted in Figure 37, the top-level structure of the model consists of four components:

[9]The detailed model is described in [PVL11].

- The desired behaviour of the turn indicator is specified using a state chart called `SystemUnderTest`, which is annotated with stereotype *SUT*.
- The environment is modelled using a state chart called `TestEnvironment`, which is annotated with stereotype *TE*.
- The input variables to the system-under-test are specified in `Interface1`, which is annotated with stereotype *TE2SUT*. This interface specifies all signals whose values are chosen by the environment and read by the system-under-test.
- The output variables of the system-under-test are specified in `Interface2`, which is annotated with stereotype *SUT2TE*. These variables are written by the system-under-test and evaluated by the test environment.

Observe that the correct stereotype annotations of the components are important for test case generation using RTT-MBT.

In the example, the `TestEnvironment` does not constrain the input variables in any way (RTT-MBT automatically ensures that the values assigned during test case generation are within the specified range). The relevant logic is thus implemented in `SystemUnderTest`, which is divided into two hierarchical state charts called `FLASH_CTRL` (Figure 38) and `OUTPUT_CTRL` (Figure 39).

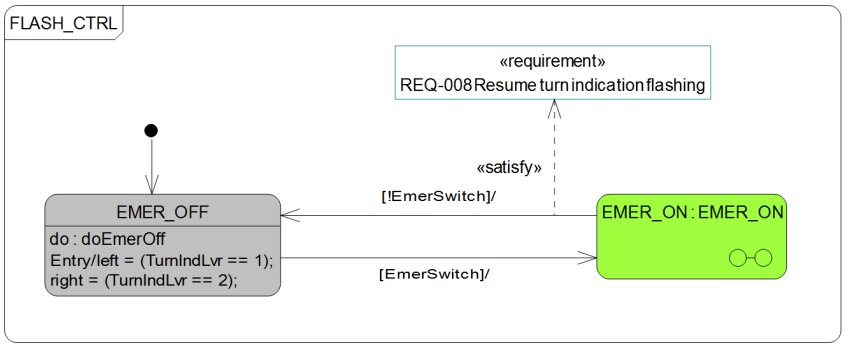


Figure 38: The `FLASH_CTRL` state machine.

`OUTPUT_CTRL` implements two modes for setting the outputs. It can be in either idle or flashing mode, where the flashing mode itself is implemented as composite state that can switch from off to on and vice versa. It does so in a regular interval if the system has enough power and a lever or the emergency button has been used.

The `FLASH_CTRL` state machine controls the impact of emergency flashing, which may both override and be override by the normal operation of the turn indication lever since there are requirements which state that "`Left/right flashing overrides emergency flashing`" (REQ-006 in Figure 40) and "`Emergency flashing overrides left/right flashing`" (REQ-005 in Figure 40). Likewise, if regular flashing is enabled before emergency flashing is activated, then regular flashing shall be resumed once emergency flashing is deactivated again (REQ-008 in Figure 38).

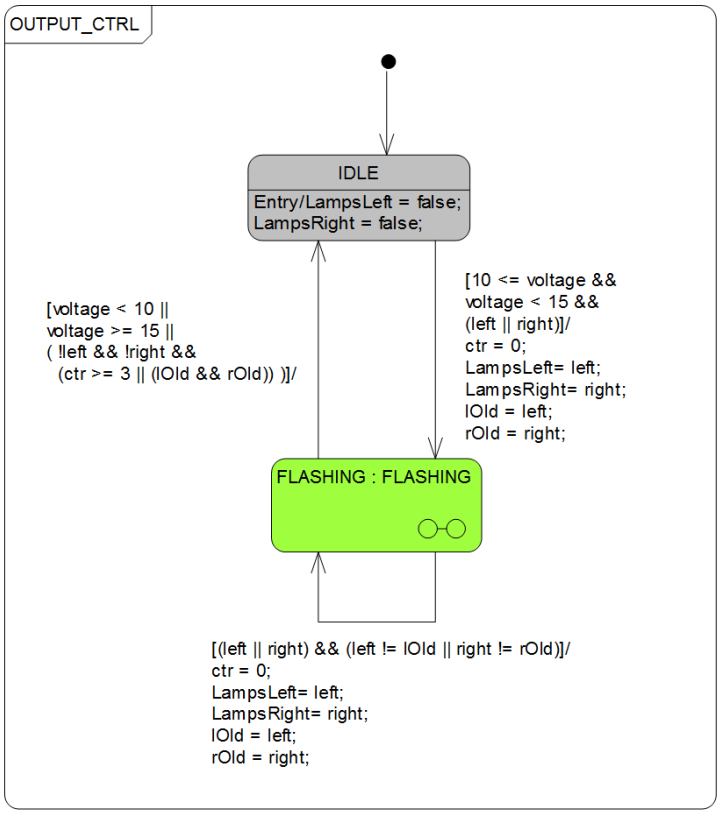


Figure 39: The `OUTPUT_CTRL` state machine.

5.3 INTO-CPS Technology

Recently, a connection between the baseline technologies Modelio and RTT-MBT has been established via XMI export from Modelio. RTT-MBT can import such XMI files without any adaptations. The imported model is then used by RTT-MBT to generate test cases depending on the specified coverage criterion, which can be, for example, transition coverage or requirements coverage[10].

[10]Observe that certain transitions in the model have been annotated with requirements. For example, the transition from state `TURN_IND_OVERRIDE` → `EMER_ACTIVE` in Figure 40 has been linked to requirement REQ-007 via a *satisfy relation*. Likewise, state `TURN_IND_OVERRIDE` has been linked to requirement REQ-006. Requirements coverage then means that RTT-MBT generates test cases which are required to exhaustively exercise the linked requirements.

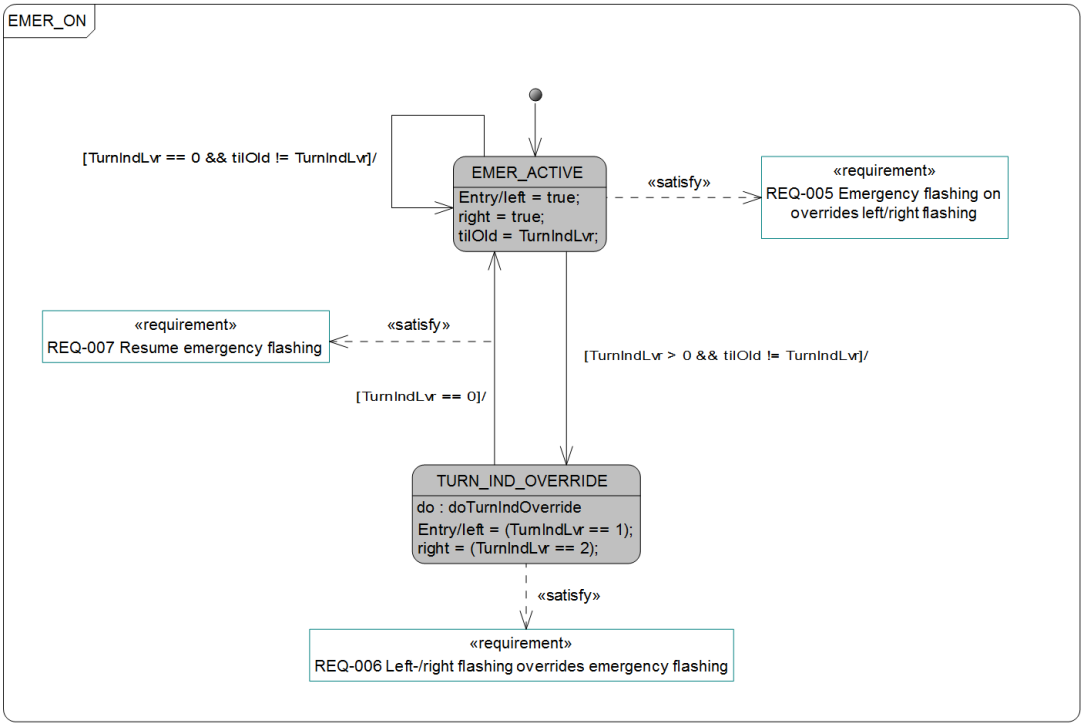


Figure 40: The `EMER_ON` composite state in the `FLASH_CTRL` state machine.

6 Roadmap for Pilot Studies

In the next 12 months of the project, the pilot studies must begin to address a wider range of INTO-CPS technologies; those available during the first year and as they become available during the second year. In addition, future pilot studies must exhibit the property of CPSs not covered in current studies; *network communication*.

6.1 Future INTO-CPS Technology Demonstration Needs

In the next 12 months, various technologies developed in the project should be demonstrable by the pilot studies. As a part of producing a roadmap, the technology developers in the project were asked for some properties required of future studies. In this section, we briefly outline characteristics the future studies should target. We do not aim to identify which studies may test these specific areas, only that they should be targets.

INTO-CPS application An example should exhibit: some changes in the simulation in terms of FMU and their connections e.g. replace a given FMU to a another one (provided by another tool); changing some parameters of the simulation (time step, total duration); the need to perform a simulation due to the result of a previous co-simulation; and the ability to manage several simulation or simulation results.

COE To test the COE, examples are sought to: use a fixed step size; uses variable step size; and exercise the constraint checking (this is the part that checks that a calculated variable step is valid and is able to reject it and calculate an alternative step size). A multi-model that becomes unstable if the wrong step size is used would be of use.

Code Generation Models are required to demonstrate both non-distributed (single CPU) and distributed (multiple CPUs) simulation.

Design-Space Exploration Examples should demonstrate the different DSE techniques. Such an example should contain 2-3 simple FMUs with short simulations, allowing us to run a complete set of parameters to determine if they reach objective; and determine how the different DSE search techniques reach best objective using rankings.

Test Automation A pilot study is required with one or more *manual implementations* of a system under test, so there is something to run the tests against. There should be at least one example with a (deliberate) functional fault, i.e., one where we have an implementation that deviates from the design model in one aspect. A timing fault should also be present (too slow / too fast). These deliberate faults should be documented in the case study description as such, to avoid confusion. In addition, as a matter for discussion, a system requirement that does not directly map to states/transitions should be constructed (e.g., using the requirement model in OpenModelica).

Model Checking Pilot studies should include purely discrete event (DE) models, continuous time (CT) models and, also a multi-model to try out the approximation technique. Ideally, one model would be scalable in the sense that the number of components for a concrete system is a selectable parameter N; this would allow to

explore the limits of technical feasibility. One or more (deliberate) modelling faults would be helpful, e.g., a failure state that cannot be left, because a reset operation is missing (deadlock).

It would be helpful to indicate a few "interesting" model properties that could be model-checked, like time response queries. For example: "After the operator selects temperature value X, it does not take longer than time Y until a temperature X+/- epsilon has been established." These could be part of the system requirements.

Traceability Pilots should give us the opportunity to exercise different workflows spanning varying combinations of the INTO-CPS tool chain. This will test and demonstrate the various provenance and traceability links between models and model elements.

INTO-CPS profile A study should exhibit non-trivial dependencies between the inputs and outputs of multi-models. In particular, there should be a cycle in their connection, so that investigation of absence of cycles in the graph of dependencies is of interest.

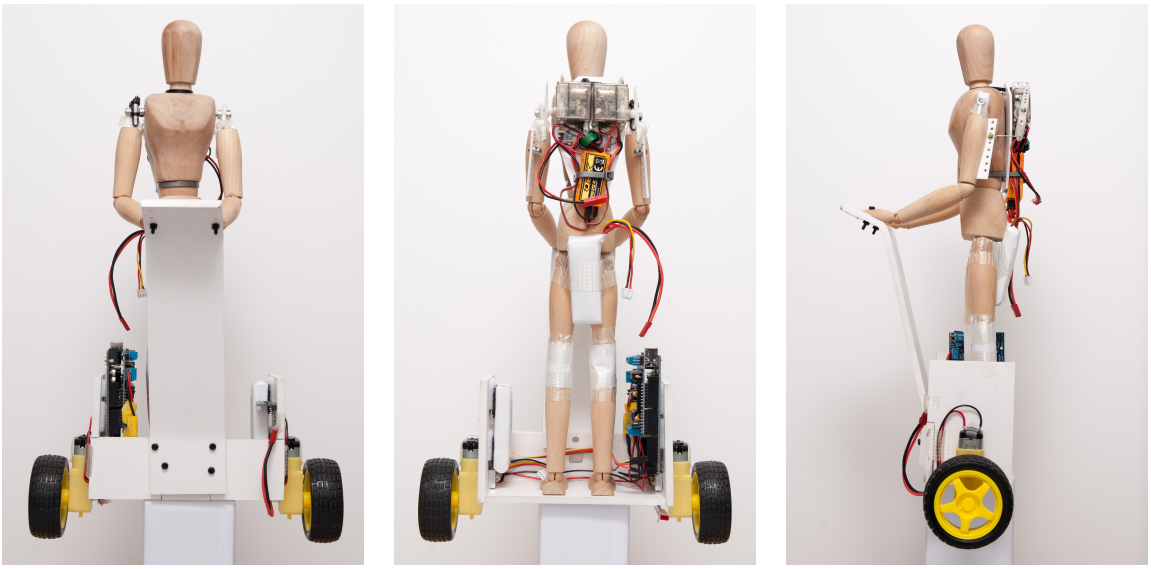
It is clear that there are different scales in requirements – some require small facets of the examples to demonstrate or test particular technologies, and others require larger scale examples. We propose in the forthcoming pilot studies work that we use the 'AAA categories' observations as proposed in the COMPASS project [NFS+12]. That is examples should progress from *adequacy*, through *application* to *adventure* – progressively testing the INTO-CPS technologies and identifying areas in the CPSE state of the art to push. There is a requirement covering this (see Deliverable D7.3 [LPH+15]), R0081 *"The examples should push the tools where possible"*. However this requirement requires careful thought to have testable acceptance criteria. This requirement will be revised at the start of Year 2 after discussions with the tool providers.

6.2 Candidate Pilots

Choosing which studies will be carried out is a task for year 2, however there are several candidates for future pilot studies in the next 12 months and in this section we outline some of those candidates. We give a brief description along with some intuition as to how they may demonstrate INTO-CPS technologies. We will also seek input from the INTO-CPS Industry Follower Group in collaboration with Task T6.2 for issues the pilots should address.

Tedway The Tedway study is a prototype self-balancing scooter [Pie15]. This example currently exists as a real, physical, prototype as shown in Figure 41. In the next year, we would propose building multi-models to represent the scooter (including gyroscope sensors, motors and wheels) and behaviours of the driver (a controllable mannequin). This example provides the possibility of demonstrating traceability and provenance features during multi-model construction, and SiL/HiL testing.

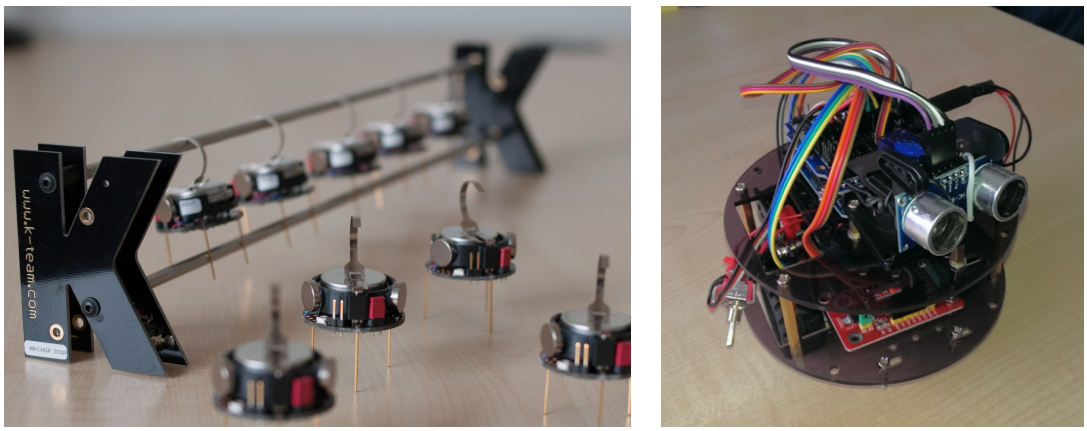
Swarm of robots This example build upon previous work in developing models of robot swarms. In [FPL14] a swarm of Kilobots is used to detect a black line of tape on a table top and for all the robots to gather together on the line. Kilobots, as shown



(a) (b) (c)

Figure 41: The Tedway self-balancing scooter

in Figure 42(a), are small cheap robots, are designed for swarm robotic experiments and may communicate of short distances using infrared (IR) light emitting diodes (LEDS). Initial work may be built upon in the use of 'TurtleBots' (shown in Figure 42(b)) which have an arduino-compatible controller, providing the option for wireless networked communication. This example provides the possibility of demonstrating traceability and provenance features during multi-model construction, SiL/HiL testing and DSE.



(a) Kilobot robots (b) Turtlebot robot

Figure 42: Swarm robot examples

Whilst both these studies use homogenous robots, we could consider heterogenous physical and cyber models, and target software hardware. Introducing heterogeneous models expands the use of multi-DE and multi-CT modelling.

Smart grid A smart grid is a modernised electricity grid that gathers and acts on a wide variety of data in order to control energy generation, distribution and consumption. Initial models have been produced using baseline technologies (SysML

and Crescendo) (see Figure 43, reproduced from [PF15]), which provides the opportunity to demonstrate a multi-model approach and would be useful in demonstrating COE simulation, applying DSE techniques and traceability.

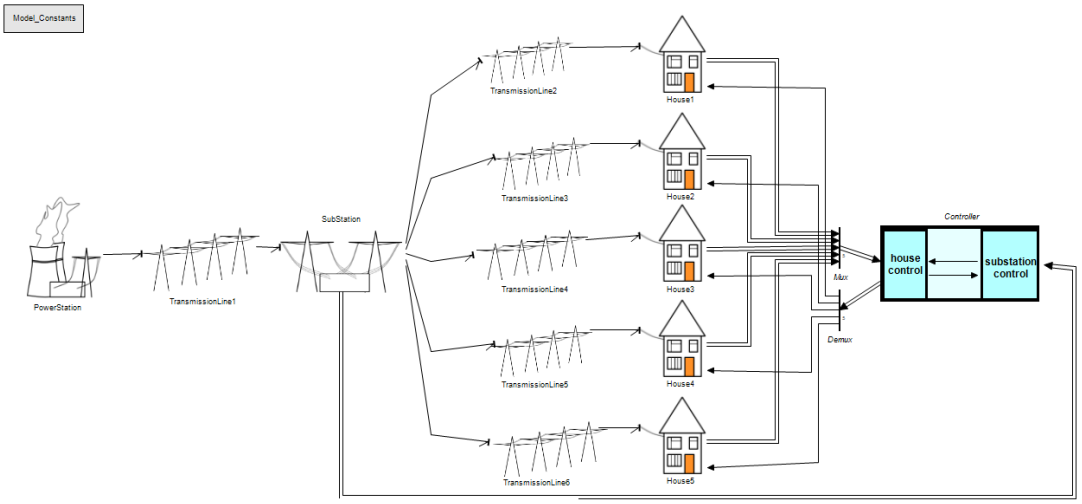


Figure 43: Smart Grid 20-sim block diagram

References

[FPL14] John Fitzgerald, Ken Pierce, and Peter Gorm Larsen. Co-modelling and co-simulation in the engineering of systems of cyber-physical systems. In *System of Systems Engineering (SoSE), 2011 6th International Conference on*, 2014.

[IPG+12] Claire Ingram, Ken Pierce, Carl Gamble, Sune Wolff, Martin Peter Christensen, and Peter Gorm Larsen. Examples compendium. Technical report, The DESTECS Project (INFSO-ICT-248134), October 2012.

[LPH+15] Peter Gorm Larsen, Ken Pierce, Francois Hantry, Joey W. Coleman, Sune Wolff, Kenneth Lausdahl, Marcel Groothuis, Adrian Pop, Miran Hasanagić, Jörg Brauer, Etienne Brosse, Carl Gamble, Simon Foster, and Jim Woodcock. Requirements Report year 1. Technical report, INTO-CPS Deliverable, D7.3, December 2015.

[NFS+12] C. B. Nielsen, J. S. Fitzgerald, R. Lloyd Stevens, S. Perry, S. Riddle, A. Romanovsky, M. Forcolin, and L. Lorenzen. Convergence report 1. Technical report, COMPASS Deliverable, D11.1, February 2012.

[PF15] Richard Payne and John Fitzgerald. Visualising Cyber-Physical Systems in the Decision Theatre. Technical report, Science Central Project, August 2015.

[Pie15] Ken Pierce. Cyber-Physical Lab: Initiating Public Engagement. Technical report, Science Central Project, August 2015.

[PVL11] Jan Peleska, Elena Vorobev, and Florian Lapschies. Automated Test Case Generation with SMT-Solving and Abstract Interpretation. In Mihaela Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, editors, *Nasa Formal Methods, Third International Symposium, NFM 2011*, pages 298–312, Pasadena, CA, USA, April 2011. NASA, Springer LNCS 6617.

A List of Acronyms

20-sim	Software package for modelling and simulation of dynamic systems
AI	Agro Intelligence
ASD	Architecture Structure Diagram
AU	Aarhus University
BDD	Block Definition Diagram
CD	Connections Diagram
CLE	ClearSy
CLP	Controllab Products B.V.
COE	Co-simulation Orchestration Engine
COMPASS	Comprehensive Modelling for Advanced Systems of Systems
CPS	Cyber-Physical Systems
CT	Continuous-Time
DE	Discrete Event
DESTECS	Design Support and Tooling for Embedded Control Software
DSE	Design Space Exploration
FCU	Fan Coil Unit
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
HVAC	Heating, Ventilation, and Air Conditioning
HW	Hardware
IBD	Internal Block Diagram
IFG	Industry Follow Group
MBD	Model Based Design
MiL	Model-in-the-Loop
PROV-N	The Provenance Notation
RTT-MBT	RT-Tester Model-Based Test Case Generator
SiL	Software-in-the Loop
SoS	System of Systems
ST	Softeam
STD	State Machine Diagram
SUT	System Under Test
SysML	Systems Modelling Language
TA	Test Automation
TWT	TWT GmbH Science & Innovation
UCD	Use Case Diagram
UNEW	University of Newcastle upon Tyne
UTRC	United Technology Research Center
UY	University of York
VDM-RT	Vienna Development Method for Real Time
VSI	Verified Systems International