# Integrated Tool Chain for Model-Based Design of Cyber-Physical Systems

Peter Gorm-Larsen[1], Casper Thule[1], Kenneth Lausdahl[1], Victor Bandur[1], Carl Gamble[2], Etienne Brosse[3], Andrey Sadovykh[3], Alessandra Bagnato[3], and Luis Diogo Couto[4]

[1] Aarhus University, Department of Engineering, Denmark
[2] School of Computing Science, Newcastle University, UK
[3] Softeam Research & Development Division, Paris, France
[4] United Technologies Research Center, Cork, Ireland

**Abstract.** Having a well-founded connection between different modelling tools such that they form a chain from requirements over formal descriptions for the constituent elements towards final realisations of Cyber-Physical Systems (CPSs) is essential. In this tool paper we explain how this can be achieved with a collection of baseline tools that are adapted to fit into such an open tool chain. The semantic foundations for the different notations used for CPSs are based on different parts of mathematics and the heterogeneous nature of these gives challenges that are solved in the suggested tool chain.

**Keywords:** Well-founded tool chain, discrete event formalism VDM-RT, continuous-time formalism OpenModelica, co-simulation, FMI

## 1 Introduction

The development of Cyber-Physical Systems (CPSs) is challenging. The close interaction between a computer-controlled cyber part and a physical part makes it hard to make the kinds of abstractions normally made from a computer science perspective. The INTO-CPS project targets the production of a well-founded chain of tools for the model-based development of CPSs [8]. In this paper we present an overview of the tool chain, its semantic foundations, baseline tools that are adapted to fit this setting and its envisaged work flow.

In the INTO-CPS project this new technology is being tested with industrial case studies in four different application domains (automotive, railways, agriculture and building automation). In addition smaller academic sized pilot studies are carried out in order to easier introduce the different features of the technology. In this paper we make use of a small line-following robot pilot study that originally was introduced in one of the predecessor projects called DESTECS[5] [4].

Throughout the paper, images are used to illustrate the features being described. Where these images show model features they are taken from the models

[5] `http://destecs.org/`.

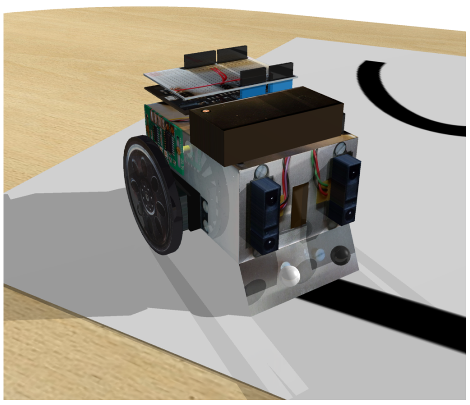


Fig. 1: The Line Follower Robot rendered from within 20-sim

of a Line Follower Robot that is being used as a pilot study in the INTO-CPS project. The robot, shown in fig. 1, comprises two infrared reflectivity sensors to detect a line, a body housing two servo motors connected to wheels for locomotion and a micro controller that reads the sensor values and produces signals that set the speed and direction of the servo motors.

The rest of the paper starts off with a short introduction to the Functional Mockup Interface (FMI) since this is essentially the glue that enables coupling between mathematical models produced in different notations and tools in Section 2. This is followed by a brief description of the semantic foundation of FMI which is using Unifying Theories of Programming (UTP) behind the scene in Section 3. Afterwards, an overview of the baseline tools that are adapted to fit an FMI context are presented in Section 4. Then Section 5 introduces a new application that serves as a front-end for end-users (typically domain experts) who need not be experts in any of the models used in the description of a CPS. The intended use of the tool chain is then presented in Section 6, followed by an outline of its design space exploration support in Section 7. Finally, Sections 9 and 10 complete the paper with information about related work, concluding remarks and future work respectively.

## 2 The Functional Mock-up Interface

When developing a CPS it can be useful to create models of the constituent components, that make up the system. These models can represent both cyber and physical parts and be described in different forms based on their nature such as Discrete Event (DE) and Continuous-Time (CT). These constituent models can then be used in a collaborative simulation (co-simulation), which couples the models created in different formalisms. Thereby the entire system can be simulated by simulating the constituent models and exchange data as the common simulated time is progressing. In principle many systems can be approximated with a DE or a CT approach alone. However, in order to accurately describe CPSs where the physical dynamics are non-linear such approximations would get far away from reality.

The Functional Mock-up Interface (FMI) defines a standardised interface to be used in computer simulations to develop complex CPSs. Such co-simulations are typically organised with a master-slave architecture, where a Master Algorithm (MA) is used to orchestrate a simulation. The simulation often consists of three phases: Initialisation, simulation, and tear down. In the initialisation phase the master retrieves the properties of the slaves and establishes communication links. Next, in the simulation phase the MA resolves the dependencies between slaves and invokes each slave to progress for a given time step. The slaves might reject the step and a rollback of one or more slaves can be necessary, and the simulation can be attempted again with a different step size. Lastly the outputs of the slaves are retrieved and the process repeats until a predetermined end time is reached. The final phase is freeing the slaves, releasing resources, and similar.

As mentioned above the models are often created in different formalisms and therefore require different specialised simulation tools [2]. This leads to developing solutions for specific systems instead of a general applicable approach, which is expensive. FMI was created to solve these challenges, as it is a tool-independent standard for co-simulation [3]. The standard describes C interfaces, that a slave must partly or fully implement in order to participate in a co-simulation using FMI. Such a slave is then referred to as a Functional Mock-up Unit (FMU). This makes it possible for the FMUs to contain their own solvers while still adhering to FMI, and provides an opportunity for developing generalised solutions[6].

## 3 Semantic Foundation

The complete semantic foundation of the INTO-CPS tool chain consists of individual semantics for the fundamental underlying activities: modelling of CT and DE systems, and co-simulation in accordance with the FMI standard. Semantics for models of CT systems is provided by a UTP formalisation of a new hybrid relational calculus with differential algebraic equations [11]. Semantics for models of DE systems is provided by a novel UTP semantics of object orientation [10], the newest semantic foundation for the Vienna Development Method's real-time dialect (VDM-RT), which also forms the semantic basis for a C code generator. The semantics of FMI co-simulation is captured in a new formalisation [6] of the FMI standard in *Circus*, a re-casting of earlier work [6] expressed in the process algebra of Communicating Sequential Processes (CSP) [14].

The *Circus* semantics of FMI captures formally the description of co-simulation given in the standard. A generic MA is modelled which determines how FMUs

[6] See [5] for more information regarding MAs for co-simulation using FMI.

are orchestrated in terms of passage of time, requests to take a simulation step and exchange of simulation results. A model of a valid FMU is also defined. These elements combine into a full semantics of co-simulation according to the FMI standard used inside the Co-simulation Orchestration Engine (COE). The key advantage of this formalisation is that it can be checked for desirable properties, such as freedom from livelock and deadlock, using the CSP refinement checker called FDR3 [13]. Indeed this has already revealed that the example master algorithm given in the standard makes an implicit assumption that FMUs do not fail in a way that is fatal to the overall co-simulation. When models of specific putative master algorithms and FMUs are also constructed as *Circus* processes, FDR3 can be used to check that the resulting specific co-simulation model is a refinement of the FMI co-simulation semantics. Expressing a particular co-simulation using the semantics can also be used to formally verify the result of executing the same co-simulation using the COE. An ongoing study is investigating how to perform this verification process [23].

## 4 Baseline Tools

The INTO-CPS tool chain has been defined on top of five existing baseline tools. Each tool, described in the following paragraphs, is extended to fit the FMI context defined into the INTO-CPS project.

*Modelio*[7] is an open-source modelling environment supporting industry standards like UML and SysML. This is used for high-level system architecture modelling, Modelio extends the SysML language [21] and proposes extensions for CPS modelling. The extended system modelling language allows, in particular, requirement, FMI interface, and FMU connections definition which can be used for generation of FMI model descriptions and configurations of co-simulations.

*Overture*[8] supports modelling and analysis in the design of discrete, typically, computer-based systems using VDM-RT dialect including both time and distribution of functionality on different computational nodes [24]. 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 whereas 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*[9] tool can represent continuous time models using connected blocks [16]. Bond graphs may implement such blocks [12]. 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,

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

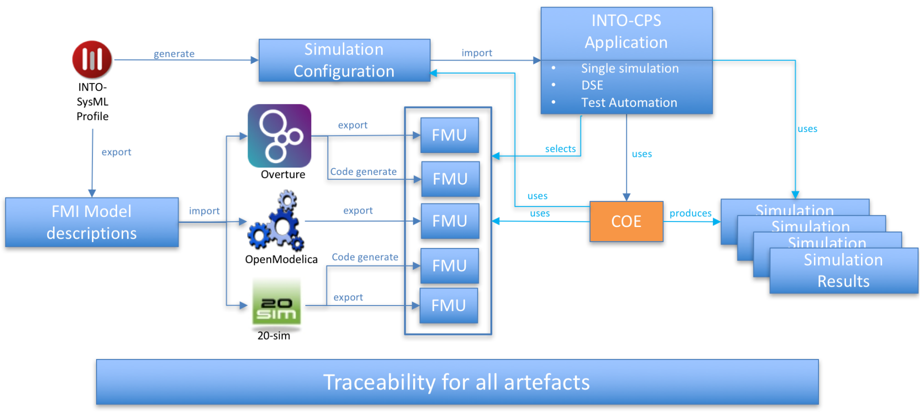


Fig. 2: The current INTO-CPS Tool Chain

which map to different concepts in different domains, for example voltage and current in the electrical domain.

*OpenModelica*[10] 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 [16]. 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. Overture, 20-sim, and OpenModelica are used for specifying FMI behaviour in their own formalism. These three tools are extended in order to consume the FMI interface definition defined previously, and, after modelled the FMI implementation, provide a FMU, conform to given FMI, for co-simulation.

*RT-Tester*[11] is a test automation tool for automatic test generation, test execution, and real-time test evaluation [18]. The RT-Tester Model Based Test Case and Test Data Generator supports model-based testing: automated generation of test cases, test data, and test procedures from SysML models. In our context, tests are generated as FMUs which are executed against the system under test.

The different baseline tools are combined together forming a chain of tools as illustrated in fig. 2. The core of the integration here is ensured by the INTO-CPS Application introduced below.

## 5 The INTO-CPS Application

In the INTO-CPS Project, the INTO-CPS application[12] has two primary responsibilities: defining an INTO-CPS project structure, and providing a UI for tool chain features that are not exposed via baseline tools such as co-simulation.

[10] `https://www.openmodelica.org/`
[11] `http://www.verified.de/products/rt-tester/`
[12] Available from `https://github.com/into-cps/intocps-ui`.

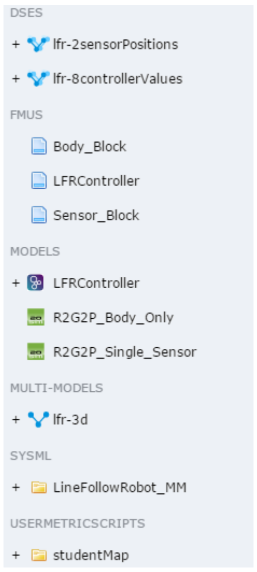


Fig. 3: The project browser.

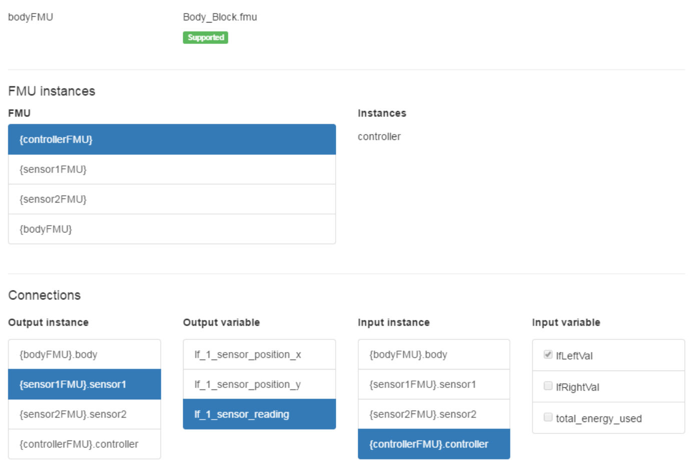
The INTO-CPS application has two regions – the project browser on the left side, and the main view in the center. The project browser shows the main artefacts in an INTO-CPS project. The browser is shown, with an example project open, in fig. 3. The main view changes dynamically, based on the activity currently being carried out by the user.

An INTO-CPS project is based on two kinds of entities: models that are produced by the baseline tools, and artefacts that are derived from models such as the results from co-simulations. The INTO-CPS application primarily interacts with model-derived artefacts. A particularly relevant model-derived artefact is the *Multi-Model*, that is produced from a combination of a connection mapping and loaded FMUs and submitted to the Co-simulation Orchestration Engine (COE) for co-simulation. The INTO-CPS application is capable of creating and editing Multi-Models, as shown in fig. 4.

From a Multi-Model, the application is capable of generating and then editing a Co-Simulation configuration which originally can be generated from SysML, as shown in fig. 5. This enables application users to set various relevant co-simulation parameters such as start and end time, the desired co-simulation algorithm, and which variables to livestream.

It is possible for the user to download the COE (and other INTO-CPS tools) and execute it from within the INTO-CPS application, as shown in fig. 6. In this way the newest released version of all the tools in the overall tool chain can always be obtained with minimum effort.

If variables have been selected for live-streaming in the co-simulation configuration, the application will plot these variables dynamically as they are streamed


bodyFMU Body_Block.fmu
Supported

FMU instances

| FMU | Instances |
|---|---|
| {controllerFMU} | controller |
| {sensor1FMU} | |
| {sensor2FMU} | |
| {bodyFMU} | |

Connections

| Output instance | Output variable | Input instance | Input variable |
|---|---|---|---|
| {bodyFMU}.body | lf_1_sensor_position_x | {bodyFMU}.body | ✓ lfLeftVal |
| {sensor1FMU}.sensor1 | lf_1_sensor_position_y | {sensor1FMU}.sensor1 | ☐ lfRightVal |
| {sensor2FMU}.sensor2 | lf_1_sensor_reading | {sensor2FMU}.sensor2 | ☐ total_energy_used |
| {controllerFMU}.controller | | {controllerFMU}.controller | |

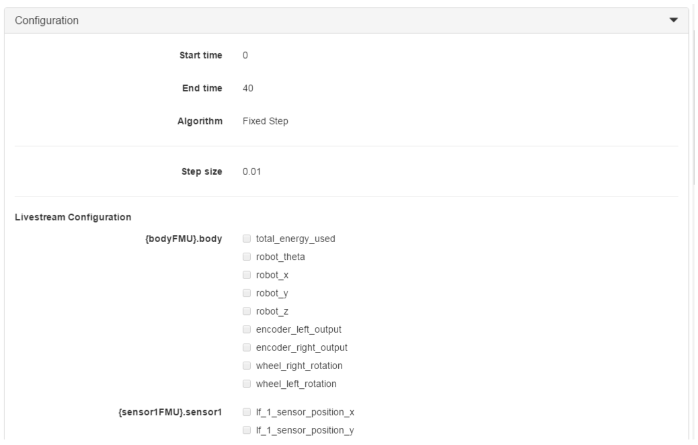
Fig. 4: Multi-Model editor.

by the COE – see fig. 7. Afterwards, the plot can be explored visually and exported as an image. The full results of the co-simulation are also exported as a Comma-Separated Values (CSV) file, for further analysis after a co-simulation.

In terms of end users, the primary intent behind the application is to enable stakeholders that are not experts in any of the INTO-CPS modelling notations to still be able to execute and evaluate co-simulations. This is possible in the current version of the application via the co-simulation configuration view and plotting and export of results. Additional views are in development for tool chain features such as traceability analysis and model checking. These views are kept isolated from each other in order to allow different kinds of experts to focus on their own tasks without being distracted by UI elements that are not relevant to them. The only view that is always visible is the project browser, since it provides navigation between views by selecting the relevant files.

## 6 Work flow

The INTO-CPS tool chain includes many tools and spans from requirements through to simulation results and generated source code and as such it may not be immediately apparent how to begin using it. Figure 8 shows and outline of the suggested workflow along with two entry points into the tool chain. The first entry point is to use SysML to model and decompose the system into tractable blocks for later analysis and development. Here the modelling is guided by an INTO-CPS SysML profile that defines suitable diagrams and model elements. This entry point requires knowledge about SysML. The second entry point is used when an organisation has pre-existing FMU models, here a subset of the SysML profile diagrams may be used to compose the FMUs into a model of the whole system. These approaches are not mutually exclusive and it is possible to compose system models from a mix of new and pre-existing models.



Configuration

| | |
|---|---|
| **Start time** | 0 |
| **End time** | 40 |
| **Algorithm** | Fixed Step |
| **Step size** | 0.01 |

**Livestream Configuration**

**{bodyFMU}.body**
- ☐ total_energy_used
- ☐ robot_theta
- ☐ robot_x
- ☐ robot_y
- ☐ robot_z
- ☐ encoder_left_output
- ☐ encoder_right_output
- ☐ wheel_right_rotation
- ☐ wheel_left_rotation

**{sensor1FMU}.sensor1**
- ☐ lf_1_sensor_position_x
- ☐ lf_1_sensor_position_y

Fig. 5: Co-Simulation configuration editor.

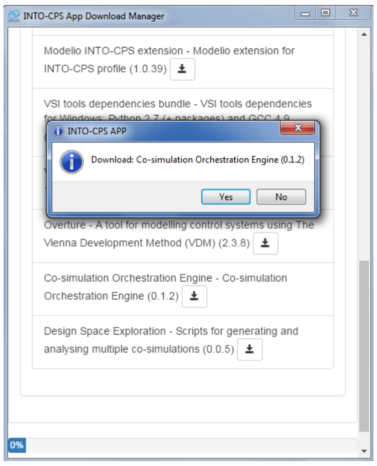
Using either the first or the second entry point both lead to the definition of a set of FMU and their connections. A specific diagram has been defined inside the INTO-CPS-SysML profile for this purpose. This is called a Connections Diagram (CD) and it represents the instantiation (possibly multiple) of FMUs and the connection existing between FMU inputs and outputs. Here fig. 9 has been extracted from the line following robot case study of the INTO-CPS project. Four instances (named controller, body, sensor1, and sensor2) of three FMUs (named Controller, Body, and Sensor) are connected.

From this diagram, a simulation configuration can be generated and then enhanced using the INTO-CPS Application.

The resulting multi-models may be analysed using a range of techniques. Simulation is the primary technique, where single designs or sets of designs, automatically generated by Design Space Exploration (DSE) scripts (see Section 7), are measured according to objectives and the values of these objectives are used to rank designs in partial order of preference. Formal analysis techniques are also supported in the form of Linear Temporal Logic (LTL) formulas acting as witnesses that temporal constraints on simulations are respected and the model checking of state machine representations of suitably abstracted CT and DE models [19].

As development proceeds further confidence may be gained by performing software in the loop (SiL) and hardware in the loop (HiL) simulation. Here the open nature of FMI and the COE allows selected model components to be replaced by their realised counterparts that then take part in simulations. Cyber components may be based upon source code automatically generated from DE models while CT models are replaced by physical components, such that simulation results may be validated.

To help manage the complexity of CPS development including many modelling artefacts, the tool chain includes support for tracking model provenance


INTO-CPS App Download Manager

Modelio INTO-CPS extension - Modelio extension for INTO-CPS profile (1.0.39)

VSI tools dependencies bundle - VSI tools dependencies for Windows: Python 2.7 (+ packages) and GCC 4.9

INTO-CPS APP

Download: Co-simulation Orchestration Engine (0.1.2)

Yes No

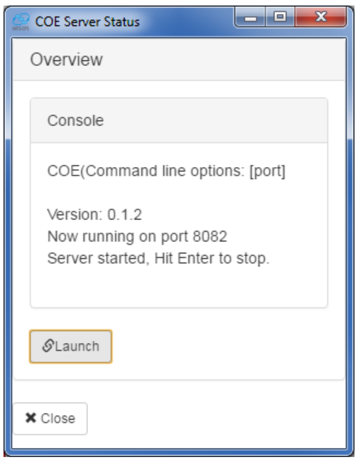
Overture - A tool for modelling control systems using The Vienna Development Method (VDM) (2.3.8)

Co-simulation Orchestration Engine - Co-simulation Orchestration Engine (0.1.2)

Design Space Exploration - Scripts for generating and analysing multiple co-simulations (0.0.5)

0%

(a) Download Manager.


COE Server Status

Overview

Console

COE(Command line options: [port]

Version: 0.1.2
Now running on port 8082
Server started, Hit Enter to stop.

Launch

✖ Close

(b) COE Execution.

Fig. 6: Downloading and launching the COE.

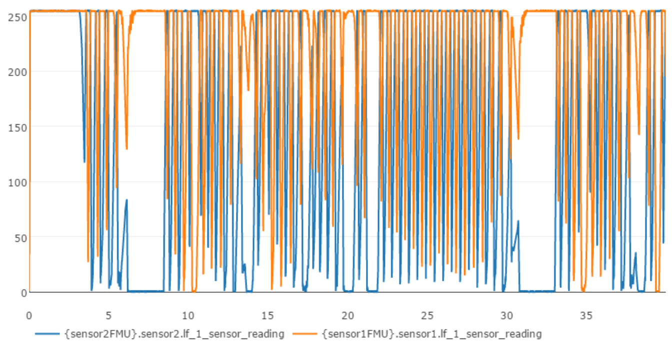


Fig. 7: Plotting of livestream variables.

and requirements traceability using a complimentary set of PROV [17] and OSLC [1] relations. Using the tools to capture these relations results in a graph that records the provenance of the modelling artefacts along with links to the related requirements. The resulting graph,of modelling and simulation activities and artefacts may then be queried to support, for example, an impact analysis exercise. A fragment of such a graph can be seen in FIgure 10.

## 7 Design Space Exploration

As an engineer proceeds with the design of a CPS they will likely be faced with many options and design parameters that must be decided upon for the final CPS to be produced. Here DSE support within INTO-CPS can be of assistance. This

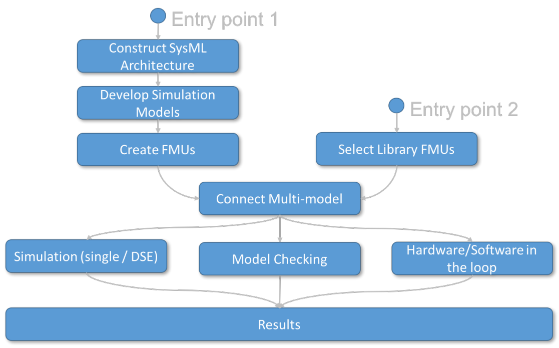

Entry point 1
Construct SysML Architecture
Develop Simulation Models
Create FMUs
Entry point 2
Select Library FMUs
Connect Multi-model
Simulation (single / DSE)
Model Checking
Hardware/Software in the loop
Results

Fig. 8: Outline work flows for using the INTO-CPS tool chain

can be divided into three parts, support for analysing each simulation; support for ranking of competing designs; and algorithms that automatically sweep over ranges of parameter values.

The DSE scripts support both open and closed loop exploration of a design space. The open loop, exhaustive algorithm, is simplest of all and it will result in every combination of the design parameters being simulated. This results in a complete coverage of the design space but it suffers from the space explosion problem and so it only generally practical for small design spaces. The closed loop algorithms, such as a genetic algorithm, use past simulation results to generate new designs to be simulated with the goal of finding a set of optimal designs without having to explore the entire design space.

In order to compare individual simulations we must have measures that characterise their behaviour in some way, these we term the objective values. The DSE scripts include built-in support to calculate a range of simple objectives from the raw simulation data, for example finding the maximum value of some variable of the simulation. It also allows the user to define their own objective scripts to calculate measures that are specific to a model or its configuration. Taking the line follow robot as an example, it uses two user defined objective scripts, one to calculate the lap time round a track and another to calculate the mean cross track error, which is a measure of how accurately the robot followed the line.

Using these objective scripts to reduce the raw simulation results to a few measures of performance allows the engineer to define a method ranking a set of designs. The engineer may define which objective values are important for the ranking of designs and whether higher or lower values are preferred for each. Using this information the scripts are able to rank the results of all the simulations that have been run using Pareto efficiency [13] to produce a non-dominated set of results representing a range of trade-offs between the selected

[13] A description of Pareto Efficiency may be found at `https://en.wikipedia.org/wiki/Pareto_efficiency`

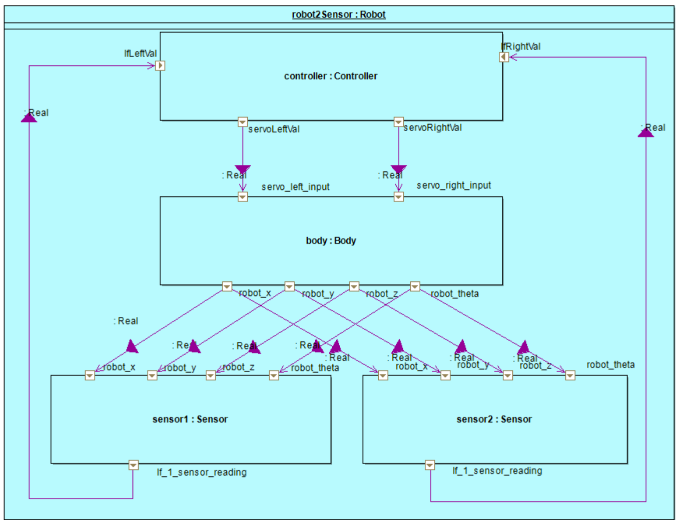


Fig. 9: The connections diagram for the line-following robot

objectives. Here fig. 11 shows the result of varying controller parameters for the line follower robot, using the lap time and mean cross track error as the means to compare designs. Here the non-dominated set, which is the green bottom-left most line, plots the results of the non-dominated set and therefore the best designs according to these measures. The DSE functionality can also be invoked from the INTO-CPS application where the graph is accompanied by a table allowing the engineer to determine the design parameters that produced each of these results.

## 8 Model Refinement and Implementation

The various optimization and verification mechanisms of the INTO-CPS tool chain enable the development of CPS multi-models to high levels of maturity. Once it is confirmed that the constituent models behave as expected in their environment, it is desirable to refine some of these to executable implementations. This mostly applies to models of control software, but there are situations in which executable implementations of models of continuous systems are desired (for instance, real-time co-simulation against cost-prohibitive environments such as large engines.) Modelica and 20-sim can generate such implementations.

With INTO-CPS, control software is discrete in nature, and is modelled in VDM-RT using Overture. There exist two approaches to refinement of models to executable implementations: formal stepwise refinement, and code generation. Since there is currently no formally defined refinement strategy for VDM-RT, Overture adopts the code generation approach. Overture's C code generator embodies a refinement strategy for VDM-RT that builds on the semantic foundation due to Foster *et al.* described in Section 3. The code generator essentially

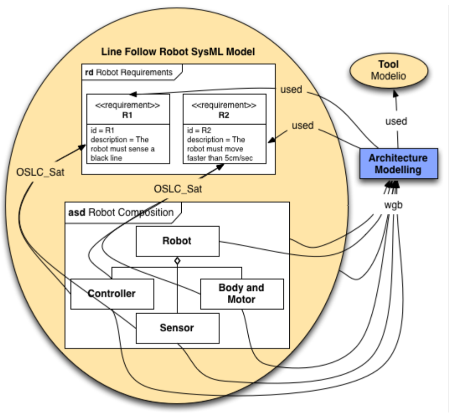


Fig. 10: Traceability links around architecture modelling of the line follower robot

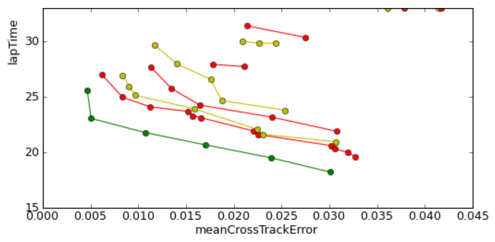


Fig. 11: Pareto plot of DSE results.

achieves an automated, one-step refinement directly to C. This refinement process is therefore not formal, but the strategy implemented is thoroughly tested to ensure that the resulting implementations conform to the aforementioned semantics. Because the refinement step is fully automated, only an executable subset of VDM-RT can be used for model construction. Naturally some underspecification (or *looseness*), for example an arbitrary choice of value from a set, can be accommodated. In contrast, "manufacturing behaviour" in accordance with contract-based specifications is considered outside the remit of code generation in the INTO-CPS context, and such constructs are not allowed in the executable subset of the language.

As a proof of concept, Overture's C code generator was used to generate an implementation of a model of a simple on/off controller that maintains the level of water in a tank between some specified limits. The core of the model is excerpted in Fig. 12. The implementation was compiled and executed on an Atmel ATmega 1284P development board[14]. A potentiometer was used to manually emulate the water level in the tank and an LED was used as feedback of the status of the tank drain valve. This is likewise shown in Fig. 12. This example

[14] A demonstration video can be found at `https://youtu.be/Qgw5NAgv3pw`.

```
loop()==
  cycles(2)
  let level : real = levelSensor.getLevel()
  in
  (if( level >=
      HardwareInterface`maxlevel.getValue())
   then valveActuator.setValve(open);
   if( level <=
       HardwareInterface`minlevel.getValue())
   then valveActuator.setValve(close);
  ); );
```

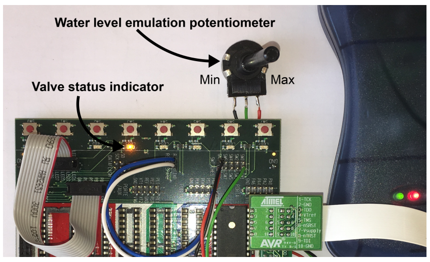


Fig. 12: Excerpt of water tank valve controller and hardware simulation.

deployment is typical of the final stages of the INTO-CPS workflow, and configuration of the hardware interface and periodic call to the control task was the only necessary manual intervention.

This exercise revealed that the value semantics of VDM-RT is one of the most problematic aspects of the language, as a naïve strategy results in implementations with very large memory footprints. For very resource-constrained embedded platforms, very aggressive measures for reducing memory usage are necessary. We have observed that, of all the language features, value semantics must receive priority when designing these measures.

## 9 Related work

The INTO-CPS tool is related to the DESTECS and SPEEDS projects, which both supports co-simulation with their own protocols and tools but they do not make use of the FMI standard as in the INTO-CPS tools. The Ptolemy II [20] is a single-tool simulation and modelling platform which can perform simulation of heterogeneous models. The tool has the ability to import standalone FMUs, leading to a high degree of model heterogeneity through a combination of native domains and external FMUs. However, it is unclear at this time whether tool-wrapper FMUs can be co-simulated. The iCyPhy project [9] focuses on the semantics of component interoperation, but a simulation tool based on Ptolemy II, FIDE [7], achieves co-simulation of FMUs.

The DANSE project models System of System (SoS) using block diagrams and is able to export this as FMUs which can be simulated in their DESYRE environment. The project developed its own specification language, the DANSE language. In addition to simulation, the project also supports statistical model checking and optimised simulation based on metrics of interest. Both are carried out by reading information directly from DANSE specifications, since the FMI standard does not include the required structural information. Of note is the fact that the technology allows multiple levels of abstraction of any given model component in a simulation. The connection between the two levels is made stochastically. It is believed that allowing such multi-level abstraction makes simulations requiring high numbers of components more tractable while still yielding accurate results. In terms of simulation support, the project supports both local simulation (termed “hosted simulation”), as well as distributed co-simulation, in the sense of INTO-CPS. However, the project makes use of FMI only for hosted simulation, where essentially only standalone FMUs are co-simulated on a single host, whereas distributed co-simulation is achieved through the use of the US Department Modeling and Simulation’s High-Level Architecture (HLA). Further information is available from the project website, as all project deliverables are publicly available [15].

The CosiMate project develops a co-simulation approach for heterogeneous systems which is very similar to INTO-CPS. However, it allows the connection of external simulation tools not only through FMI, but also through their native control interfaces. Addition of a new simulation tool to a co-simulation scenario is facilitated by an Eclipse-based interface construction environment. The co-simulation platform supports variable time steps in the same way at the COE from INTO-CPS.

The ADVANCE project[15] [22] allows co-simulation of Event-B machines with external continuous-time FMUs through FMI version 1. The resulting technologies support model-based testing and model-checking of CPS using ProB. The co-simulation capabilities of the ADVANCE MultiSim simulation framework are similar to those projected for the INTO-CPS tool chain, and are implemented as a plugin for the Rodin platform for Event-B. However, owing to the capabilities of Rodin, proof in that domain is better integrated with the relevant tool than current proof support for VDM-RT, but INTO-CPS has the main advantage that it seeks to make a co-simulation platform. The aim in INTO-CPS is to co-simulate both discrete-event and continuous-time FMUs together without knowing the details about the implementation of the FMUs, as long as they are compatible with the FMI version 2 standard. Further information is available from the project website, as all project deliverables are publicly available. This work will like the above mentioned projects support FMI for all base line tools and therefore enable fixed/variable-stepsize co-simulation. In addition, it will provide traceability support, and test automation at the FMU level as well as model checking, and design space exploration for optimised simulation based on metrics of interest.

## 10 Concluding Remarks and Future Work

In this paper we have provided an overview of the INTO-CPS tool chain and briefly touched upon its foundations. We believe that in order to ensure interoperability between different models of different constituent elements of a CPS, a semantic foundation such as suggested above is paramount. This is an area where we hope that others in the formal methods community will take inspiration from this work.

The INTO-CPS tool chain described in this paper is not yet complete, but the connectivity between the different parts has already been demonstrated: the tool

[15] `http://www.advance-ict.eu/`

chain is being used for industrial case studies in railways, agriculture, automotive and building automation. Most notably, support for traceability, essential for providing well-founded arguments for the analysis conducted for the different models to be presented to external stakeholders, is not yet implemented.

*Acknowledgments.* We would like to thank the anonymous referees for valuable input on this work. Our current work is partially supported by the INTO-CPS project (Horizon 2020, 664047). We would also like to thank all of the INTO-CPS participants for all their contributions making the INTO-CPS tool chain a reality.

## References

1. Open Services for Lifecycle Collaboration (OSLC). `http://open-services.net/`
2. Bastian, J., Clauss, C., Wolf, S., Schneider, P.: Master for Co-Simulation Using FMI. In: 8th International Modelica Conference (2011)
3. Blochwitz, T.: Functional Mock-up Interface for Model Exchange and Co-Simulation. `https://www.fmi-standard.org/downloads` (July 2014)
4. Broenink, J.F., Larsen, P.G., Verhoef, M., Kleijn, C., Jovanovic, D., Pierce, K., Wouters, F.: Design Support and Tooling for Dependable Embedded Control Software. In: Proceedings of Serene 2010 International Workshop on Software Engineering for Resilient Systems. pp. 77–82. ACM (April 2010)
5. Broman, D., Brooks, C., Greenberg, L., Lee, E.A., Masin, M., Tripakis, S., Wetter, M.: Determinate Composition of FMUs for Co-Simulation. In: 13th International Conference on Embedded Software (EMSOFT), Montreal (September 2013), `http://chess.eecs.berkeley.edu/pubs/1002.html`
6. Cavalcanti, A., Woodcock, J., Amálio, N.: Behavioural models for fmi co-simulations. In: Sampaio, A.C.A., Wang, F. (eds.) International Colloquium on Theoretical Aspects of Computing. Lecture Notes in Computer Science, Springer (2016)
7. Cremona, F., Lohstroh, M., Tripakis, S., Brooks, C., Lee, E.A.: FIDE – An FMI Integrated Development Environment. In: Symposium on Applied Computing (2015)
8. Fitzgerald, J., Gamble, C., Larsen, P.G., Pierce, K., Woodcock, J.: Cyber-Physical Systems design: Formal Foundations, Methods and Integrated Tool Chains. In: FormaliSE: FME Workshop on Formal Methods in Software Engineering. ICSE 2015, Florence, Italy (May 2015)
9. Fizher, A., Jacobson, C.A., Lee, E.A., Murray, R.M.: Industrial Cyber-Physical Systems – iCyPhy. In: et al., M.A. (ed.) Complex Systems Design and Management. pp. 21–37. Springer (January 2014)
10. Foster, S.: Final version of the Semantics for VDM-RT. Tech. rep., INTO-CPS Deliverable, D2.2b (December 2016)
11. Foster, S., Thiele, B., Woodcock, J.: Differential Equations in the Unifying Theories of Programming. Tech. rep., INTO-CPS Deliverable, D2.1c (December 2015)
12. Gawthrop, P.J., Bevan, G.P.: Bond-graph modelling: A tutorial introduction for control engineers. IEEE Control Systems Magazine 27(2), 24–45 (2007)
13. Gibson-Robinson, T., Armstrong, P., Boulgakov, A., Roscoe, A.: FDR3 — A Modern Refinement Checker for CSP. In: Tools and Algorithms for the Construction and Analysis of Systems. LNCS, vol. 8413, pp. 187–201 (2014)

14. Hoare, C.: Communicating Sequential Processes. Communications of the ACM 21(8) (August 1978)
15. IEEE Standard for Modeling and Simulation: High Level Architecture (HLA)– Framework and Rules. IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000) pp. 1 –38 (August 2010)
16. Kleijn, C.: Modelling and Simulation of Fluid Power Systems with 20-sim. Intl. Journal of Fluid Power 7(3) (November 2006)
17. Moreau, L., Missier, P.: PROV-DM: The PROV Data Model. Tech. rep., World Wide Web Consortium (2012), `http://www.w3.org/TR/prov-dm/`
18. Peleska, J.: Industrial-Strength Model-Based Testing - State of the Art and Current Challenges. Electronic Proceedings in Theoretical Computer Science abs/1303.1006, 3–28 (2013)
19. Pnueli, A.: The Temporal Logic of Programs. In: 18th Symposium on the Foundations of Computer Science. pp. 46–57. ACM (November 1977)
20. Ptolemaeus, C. (ed.): System Design, Modeling, and Simulation using Ptolemy II. Ptolemy.org (2014), `http://ptolemy.org/books/Systems`
21. Sandford Friedenthal, Alan Moore, R.S.: A Practical Guide to SysML. Morgan Kaufman OMG Press, Friendenthal, Sanford, First edn. (2008), ISBN 978-0-12-374379-4
22. Savicks, V., Butler, M., Colley, J.: Co-simulating event-B and Continuous Models via FMI. In: Proceedings of the 2014 Summer Simulation Multiconference. pp. 37:1–37:8. SummerSim '14, Society for Computer Simulation International, San Diego, CA, USA (2014), `http://dl.acm.org/citation.cfm?id=2685617.2685654`
23. Thule, C.: Verifying the Co-Simulation Orchestration Engine for INTO-CPS. In: Doctoral Symposium FM 2016. Limassol, Cyprus (November 2016)
24. Verhoef, M., Larsen, P.G., Hooman, J.: Modeling and Validating Distributed Embedded Real-Time Systems with VDM++. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006: Formal Methods. pp. 147–162. Lecture Notes in Computer Science 4085, Springer-Verlag (2006)