FMI for Co-Simulation of Embedded Control Software

Nicolai Pedersen[1,2] Tom Bojsen[2] Jan Madsen[1] Morten Vejlgaard-Laursen[2]

[1] Technical University of Denmark, Embedded Systems Engineering, Kgs. Lyngby DK-2800,Denmark ,
{nicp, jama}@dtu.dk
[2]MAN Diesel & Turbo, Teglholmsgade 41 Copenhagen DK-2450, Denmark,
{nicolai.pedersen , tom.bojsen, morten.laursen}@man.eu

Abstract

Increased complexity of cyber-physical systems within the maritime industry demands closer cooperation between engineering disciplines. The functional mockup interface (FMI) is an initiative aiding cross-discipline interaction by providing, a widely accepted, standard for model exchange and co-simulation. The standard is supported by a number of modelling tools. However, to implement it on an existing platform requires adaptation. This paper investigates how to adapt the software of an embedded control system to comply with the FMI for co-simulation standard. In particular, we suggest a way of advancing the clock of a real time operating system (RTOS), by overwriting the idle thread and waiting for a signal to start execution until return to idle. This approach ensures a deterministic and temporal execution of the simulation across multiple nodes. As proof of concept, a co-simulation is conducted, showing that the control system of an SCR (selective catalyst reduction) emission reduction system can be packed in a functional mockup unit (FMU) and co-simulated with a physical model, built in Ptolemy II. Results show that FMI can be used for co-simulation of an embedded SCR control software and for control software development. *Keywords: Co-Simulation, RTOS, FMI, FMU, Embedded Systems*

1 Introduction

Designing the next generation of embedded cyber-physical systems (CPS) requires close collaboration between physical model developers and the engineers implementing the computation, communication and control. The amount of sub-systems, deviation in the tool chain and standards are often barriers between these disciplines. Teams are divided into different departments within organisations or in cross-company collaborations, further complicating the cooperation. One of the recent initiatives to lower this barrier is the functional mockup interface (FMI) (Blochwitz et al., 2009). It is a tool-independent standard for model exchange and co-simulation. FMI was initiated by the automotive industry and released in a version 1.0 in 2010 followed by a 2.0 version in 2014. This paper does not explain the standard, but aims to show the process of adapting an embedded system to comply with FMI. Implementing the FMI standard on an existing modelling platform is straightforward, especially since many of the open-source and commercial tools already support it. Forcing a specialised embedded system to comply is, however, a demanding task that requires adaptation.

At MAN Diesel & Turbo, legislation on pollution and a demand for support of alternative fuel types are increasing the amount of distributed sub-systems and the complexity of the traditional two-stroke diesel engine. The increased distributed complexity makes the cooperation between cyber and physical parts of the system even more crucial. Currently, simplified physical models are used for control algorithm development, and only estimations of the control system dynamics is considered when modelling the physical behaviour. The objective of this project is to enhance the modelling development and distribution at MAN Diesel & Turbo by introducing a more comprehensive system simulation. We wish to simulate both physical behaviour and control dynamics, combined with a model of the software. The software model will enable us to investigate system behaviour such as alarm handling, IO scaling and network communication/protocols. The main challenge is to adapt the embedded engine control system into a functional mockup unit (FMU). The process of this adaptation is what will be presented in this paper. As use case, a simple model of the SCR (Selective Catalyst Reduction) emission reduction system and its control software will be co-simulated.

FMI 2.0 for co-simulation has been chosen due to its strict type/execution structure combined with its freedom of implementation. The standard is highly recognised and applied within the automotive industry (Abel et al., 2012; Stoermer and Tibba, 2014), which has many similarities with the maritime. Recently, applications within energy and grid systems (Vanfretti et al., 2014; Elsheikh et al., 2013) and HVAC systems (Nouidui et al., 2014) are emerging as well. FMI applications within the maritime industry, like this, is limited (Pedersen et al., 2015).

DOI 10.3384/ecp1612470

This project uses the heterogeneous simulation software framework Ptolemy II (Liu et al., 2001; Brooks et al., 2010) to co-simulate a simple physical model with an imported FMU. Ptolemy II has been used for various FMI applications (Broman et al., 2013; Liu et al., 2001; Lee et al., 2015) Much attention has been put on implementing the standard, such as FMI++ (Widl et al., 2013) the FMI Library from (Modelon) and the FMU SDK from (QTronic). Examples of how to build an FMI master algorithm has been provided as well (Bastian et al., 2011; Broman et al., 2013). In (Bertsch et al., 2015) a prototypical realisation of an FMU executing on a Bosch electronic control unit was presented. However, the non-trivial process of adapting the software of an embedded system, with at real-time operating system (RTOS), into a co-simulation FMU, has not yet been described, but will be in this paper.

First the cyber-physical system at hand will be introduced in Section 2. Section 3 shows how to move from a target embedded application to an FMU running in a regular Linux environment. A use-case implementation is presented in Section 4 and conclusions are drawn in Section 5

2 Cyber-Physical System

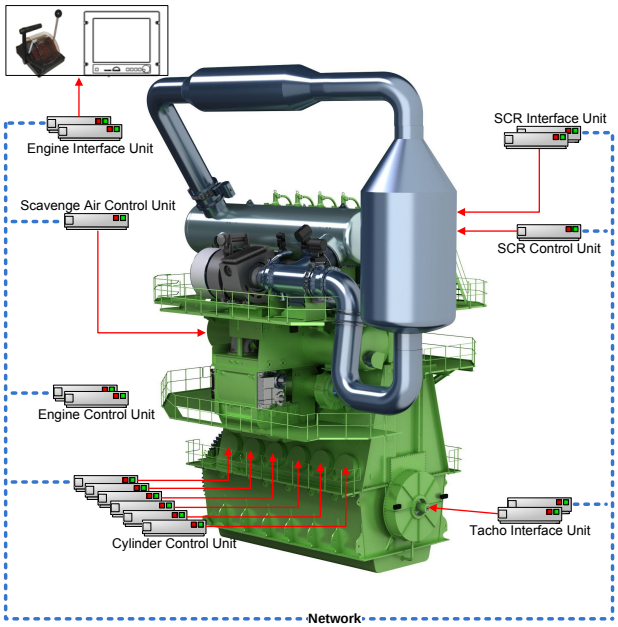

Figure 1. An MAN Diesel & Turbo two-stroke low-speed diesel engine with the SCR and the engine control system illustrated

MAN Diesel & Turbo designs large-bore diesel engines and turbomachinery for marine propulsion systems and stationary applications, such as power plants. With the introduction of the electronically controlled line of ME engines in 2002, MAN Diesel & Turbo moved into the development of Cyber-Physical System. In recent years, the demand for new emission reduction systems and alternative fuel types have made the core engine even more dependent on the surrounding control system. This dependency demands a more advanced simulation environment including co-simulation. The engine control system consists of numerous distributed controllers with each their specific control objective connected by a wired network. Figure 1 illustrates a 6-cylinder two-stroke ME-engine with an SCR system and engine control system. The main controllers are the engine interface units communicating with the operator, and the scavenge air control unit ensuring that pressures are balanced between the turbocharger and scavenging. The engine control units ensure that the cylinder control units perform the correct temporal injection ect. according to the information about the crankshaft position from the tacho interface units. Finally, if the engine is fitted with an auxiliary system e.g. an SCR system, it will be controlled and monitored by its own SCR units.

3 From Embedded Target to FMU

Figure 2. A multi-purpose controller of the MAN Diesel & Turbo engine control system

To achieve the objective of co-simulating the software control system together with a physical model, in a different environment(Ptolemy II), we need to make our target application code run in a functional mockup unit 0(FMU). It should be noted that the main objection of this solution is to aid physical modelling and control algorithm development. The solution will therefore demonstrate a deterministic simulation of both computational execution and network. Despite the previously described system behaviour investigation benefits, of including a software model in the FMU, the decision is also based on future ambitions and the current control system development at MAN Diesel & Turbo. Future

plans include a stochastic network model and HIL-nodes combined with FMI nodes.

3.1 Configuration Abstractions

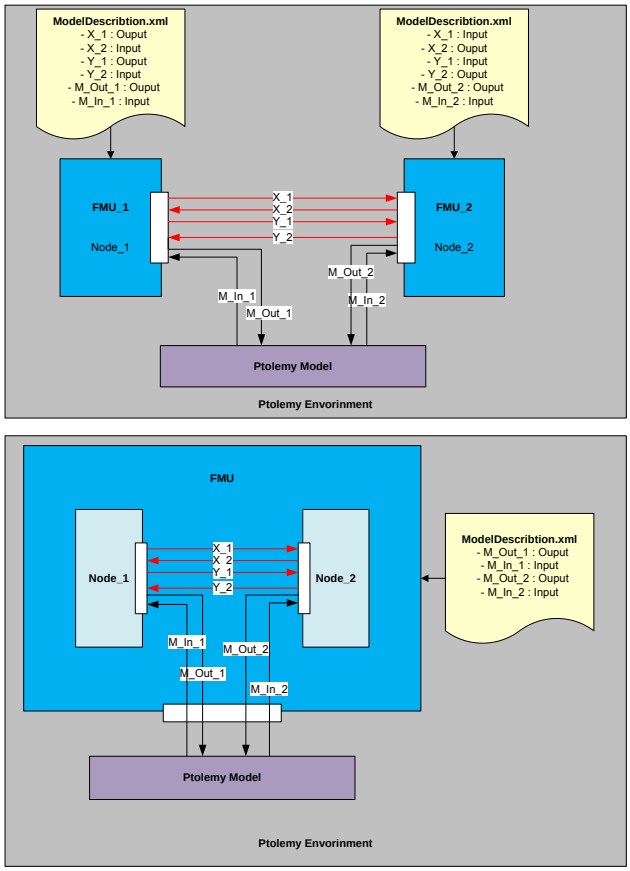

Figure 3. It is possible to change the level of configuration complexity exposed to the user. The top figure shows how each control system node can be packed in an FMU for maximal configuration flexibility. The bottom figure shows how multiple nodes can be packed and configured in a single FMU for a simpler user configuration setup.

One of the most important concerns when introducing FMI was the configuration complexity. The system is to be used by different disciplines, and it is important that the configuration level can be abstracted to fit the user objective - meaning that if a hydraulic engineer wishes to investigate the dynamic effects of the control system on his model, he should not have to connect all the wires of the control system to get started, but rather have one FMU with only relevant variables and parameters exposed. We found it beneficial to maintain the possibility of interconnecting multiple nodes of the control system before wrapping them into the functional mockup interface. As shown in Figure 3, this allows for different levels of configuration complexity. If we are interested in both the interaction between two nodes and a physical model, we can provide all variables, parameters and IOs through multiple FMUs and connect them in our environment, see top Figure 3. However, if we are only interested in the variables interacting with our external model, it is possible to connect the nodes internally, and only expose the relevant variables, bottom figure 3. The latter option provides a much simpler configuration and "ModelDescription.xml" for the user and lets the control system experts ensure that nodes are connected correctly.

3.2 Target to PC simulation

The target controllers used are multi-purpose, meaning e.g. that cylinders and SCR-control units are identical. The only deviation determining the specific controller objective is the software executed on the embedded system. A controller interfaces with sensors and other computational units, using the information to interact with the system through actuators. A controller contains a CPU module with an FPGA-based embedded system utilising a real-time operating system. The strategy for simulating our embedded system is to model the entire embedded system from the operating system and up, wrapping this into an FMU. Conclusively, our model is not simulating the behaviour of the embedded processor, but builds the target code for an x86 architecture in a so called PC-simulation application (PCSIM).

3.3 FMI implementation of PC simulation

To implement FMI 2.0 for co-simulation, we need further access to some main functionality embedded in the PCSIM. Looking at the FMI co-simulation state machine (Blochwitz et al., 2009), we need to access relevant data for $fmi2Set()$ and $fmi2Get()$ and a way of stepping the simulation according to the $fmi2DoStep()$ function. Furthermore, the network communication is to be reconnected and the FMI functions implemented.

3.3.1 Hook to OS clock

For the co-simulation to work correctly, we need to control the execution between the discrete communication points on each node. The approach is to access the clock of the operating system and let a simulation manager control the temporal execution. This is made possible by building the target code as a shared library and overwriting the idle thread method of the RTOS. The RTOS used in this project supports an x86 architecture and provides the board support package, which includes a bsp_idle_thread to be manipulated. The solution proposed will require customisation to work with different RTOS versions, however, the concept is generic. Besides the idle thread hook, we need to be able to start and stop the application by calling the main function through the library. The main function is executed in a separate thread until we force it to stop, having the main function return. The new idle thread function has an idle callback function that implements ticking of the RTOS clock. Each tick lasts for a simulated 1 ms, implemented

DOI 10.3384/ecp1612470

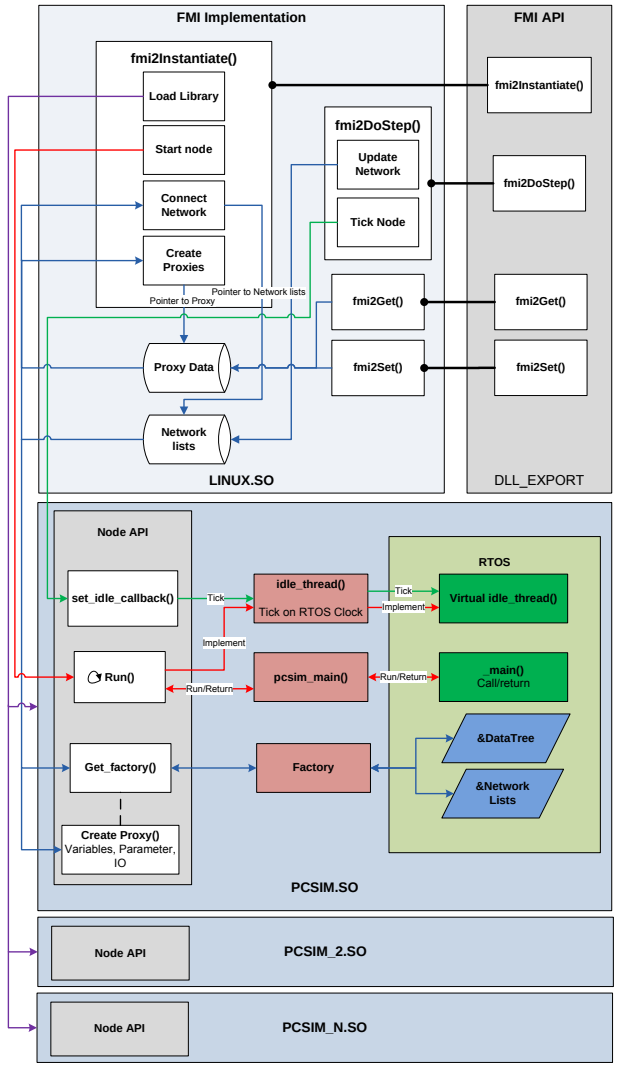

Figure 4. The implementation of FMI on the MAN engine control system

by assuming unlimited CPU power - thus an execution time of zero for every node, followed by a 1 ms delay. A node will run until it returns to idle, meaning for every tick, all task will finish and never be interrupted. This guaranties a common perception of time across nodes. The assumption of unlimited processing power will obviously make the simulation results deviate slightly from a real stochastic execution. However, it ensures a determinism which is important during control algorithm development and regression testing. All interrupts are currently software simulated and scheduled as regular tasks. Further work will aim to implement a more temporal scheduling of especially high frequency interrupts.

Having a hook to the clock and a joint time perception makes it possible for a manager to call the $fmi2DoStep()$ function and orchestra a correct temporal execution of the co-simulation.

3.3.2 Connecting variables, parameters and IO channels

On the target application all variable, parameters and IO channels are organised in a component-oriented data tree structure with unique IDs. Using a factory method design, we make it possible to create proxies for both variables, parameters and IO channels, providing a $Proxy.Get()$ and $Proxy.Set()$ function that will effect the source on the specific node. For IO channels, we communicate on micro-ampere level, so prober conversion is needed.

The $fmi2Set()$ and $fmi2Get()$ functions will write and return the value of the proxies. The instantiation of proxies are done in the $fmi2Instantiate()$ function and is based on the "$ModelDescription.xml$". One of the advantages of FMI is the strict data type definition. However, the target application utilises more data types than the ones allowed by FMI, such as fix-point and unsigned short. As a result, a type conversion layer had to be added.

3.3.3 Solving network communication

To simulate the network communication between nodes, we replace the RTOS network driver with a deterministic input/output queue implementation. Each node is given an address corresponding to the unique *node_id* already provided by the controller. Through the factory design from 3.3.2 input and output lists are made available across nodes. A network manager then redirects packages from output to input queues according to network address. The network manager support both unicast, multicast and broadcast. Communication is done at every discrete communication point, and the network driver is activated every ms tick of the OS clock, if any data is available in the input or output queue. Currently, the network is only available with interconnected nodes and not as an output through the FMU. However, this is something we are working on.

3.3.4 FMI implementation

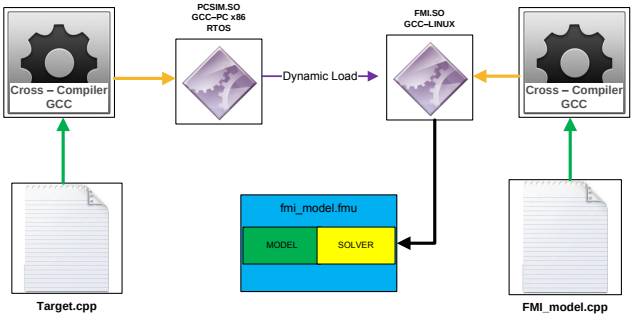

Figure 5. The compiling routine from target to functional mock-up unit.

According to the FMI standard the application should be compiled into a shared library with the FMI functions exported. As described, we are able to build each of our control nodes into PC shared libraries (PCSIM.so) including a, x86 RTOS. We now need to wrap these into a Linux shared library (FMI.so) implementing the FMI application interface. One or more PCSIM.so are loaded into the FMI.so which is the main binary in the co-simulation FMU, see figure 5. A MAN Diesel & Turbo FMU will have the 2.0 FMI for Co-simulation API. The $fmi2Instantiate()$ will load the PCSIM.so's required for the specific scenario and create the relevant parameters, inputs and outputs according to the *ModelDescription.xml* and start each node executing. The $fmi2DoStep()$ is able to call the idle callback function on each node, signalling the idle thread to tick the RTOS. If an FMU contains more than one node the network will be updated at every communication point. The remaining FMI functions have been implemented but not illustrated in Figure 4.

4 Use Case: SCR Temperature Dynamics and Control

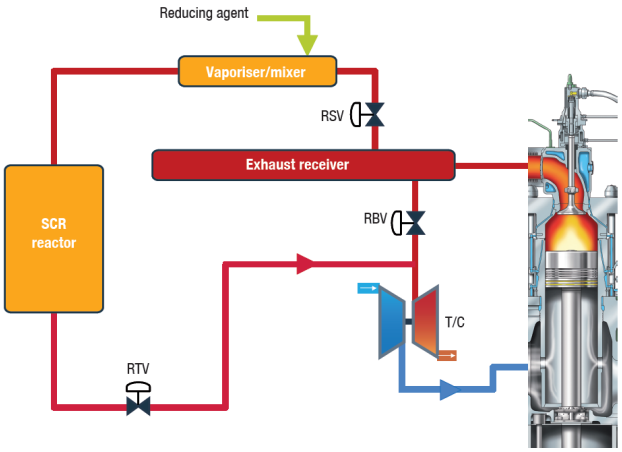

Figure 6. Diagram of the SCR system

As a simple use case, we look at the dynamics and control of heating up the SCR reactor. When a vessel is to comply with the Tier III emission limits (IMO, 2008) for NO_x reduction, a command is sent from the operator to activate the SCR control. The SCR control unit will then redirect the exhaust gas through the reactor by opening the reactor sealing valve (RSV) and the reactor throttle valve(RTV). The controller has to balance the RTV opening, to ensure that the flow to the turbine inlet of the turbocharger is sufficient. As soon as the reactor is properly heated, the reactor bypass valve (RBV) can be closed; consequently, only cleaned air from the reactor leaves the system as exhaust. A diagram of the SCR control is illustrated in Figure 6. The SCR controller uses the difference between the reactor input and output temperature as a reference residual signal for controlling the position of the RTV valve. By modelling the time delay of heating the reactor and passing the resulting output temperature back to the SCR controller, we will show that it is possible to investigate the dynamic interaction between a physical model and the actual control software.

Many additional observations regarding the engine physics are required for all aspects of the SCR controller to perform correctly. An advantage of being able to connect more nodes within a single FMU is that the so-called engine simulation unit (ESU) used for hardware in the loop test can be included. The ESU contains numerous physical models executing within the embedded controller environment. Model execution on the ESU must comply with real-time requirements and should therefore not be too complex. With FMI, it is possible to make a hybrid simulation of the engine physics where ESU models can be combined with Ptolemy models. In this use case, the reactor heating model provides physical insight into the SCR controller together with the ESU.

4.1 SCR Heating Model

The reactor heating model chosen as proof of concept is described below. The output temperature can be modelled as the relationship between the RTV position, the flow through the reactor and the input temperature, resulting in two low-pass filters with a significant time constant. The inputs to the model is provided by the SCR controller and ESU.

The mass flow into the reactor $\dot{M}$ is estimated from the engine load L.

$$\dot{M}_n = \dot{M}_{n-1} + \frac{L - \dot{M}_{n-1}}{1 + \tau_{Scavenge} \cdot T} \tag{1}$$

where T is the sampling frequency.

The time constant of the reactor output temperature, is estimated as the RTV valve opening with the mass flow plus a time constant, converted into seconds.

$$\tau_{out} = (\dot{M}_n \cdot RTV + \tau_{reactor}) \cdot 3600 \tag{2}$$

Finally, the output temperature is calculated as

$$Tout_n = Tout_{n-1} + \frac{Tin + Tout_{n-1}}{1 + \tau_{out} \cdot T} \tag{3}$$

This is naturally a simplified approach, however, it goes to show, that it is possible to distribute the control system and co-simulate with other thermodynamic models regardless of the abstraction level.

4.2 Ptolemy II as simulation framework

As simulation framework, the open-source Ptolemy II was chosen due to its heterogeneous actor-oriented

DOI 10.3384/ecp1612470

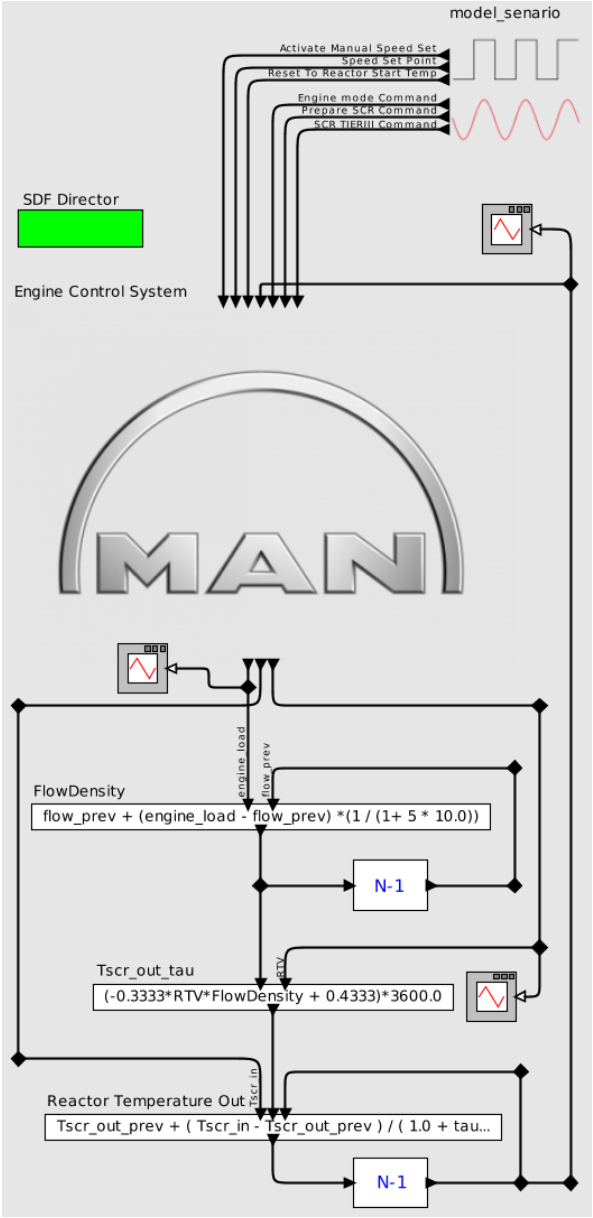

Figure 7. FMU import in Ptolemy II and simple physical model implementation

design and comprehensive support for different software components and the FMI interface as described in (Broman et al., 2013). The FMU is imported as a co-simulation actor automatically configured by the "*ModelDescribtion.xml*". Using "*Vergil*", the graphical user interface shipped with Ptolemy, the equations from 4.1 are created and connected to the FMU outputs. A simulation scenario is likewise defined in Vergil and connected to the input ports of the FMU, see Figure 7. The scenario sets a reactor start temperature and an engine speed set point. After 700 seconds, a simulated bridge command is send to the SCR controller, activating the SCR control strategy.

To execute the simulation, a synchronous dataflow (SDF) director was chosen. The SDF director is appropriate because we have a predictable and regular execution (firing) of the FMU. At regular communication points, inputs/outputs are updated in a predefined order.

4.3 Results

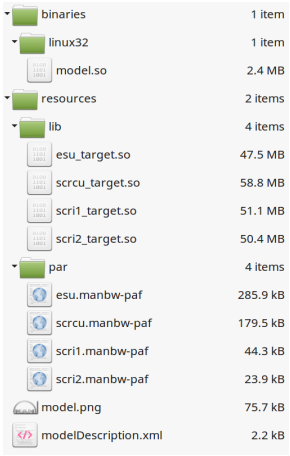

Figure 8. The use-case example of a functional mock-up unit containing the MAN SCR control nodes

To run the simulation, an FMU was build as seen in Figure 8. Here four PCSIM.so corresponding to the code of four embedded controllers, are packet into "*resources/lib*". The engine simulation unit (*esu_target.so*) models the entire engine, except the SCR heating model, using the target solver ect. An SCR Control Unit (*scrcu_target.so*) containing all the control algorithms for the reactor control and two SCR interface controllers (*scri1_target.so*,*scri2_target.so*) redirecting all the sensor values connected as simulated cables from the ESU to the SCRCU by network. Configuration of the PCSIM applications are provided via the MAN parameter files located at "*resources/par*"

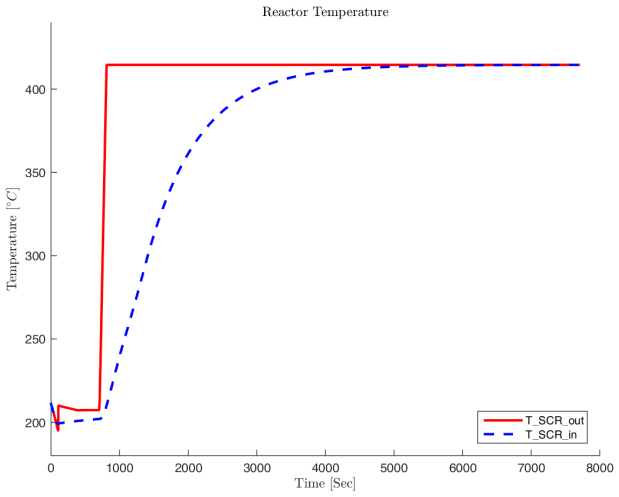

Figure 9. The in- and output temperature of the simulated SCR heating

The simulation of the FMU and reactor heating model is presented in Figure 9. Here we see that the SCR reactor out temperature start to increase after 700 seconds when the SCR start command is sent. The heating has the expected low-pass behaviour and takes approximately 1.5 hours to heat up.

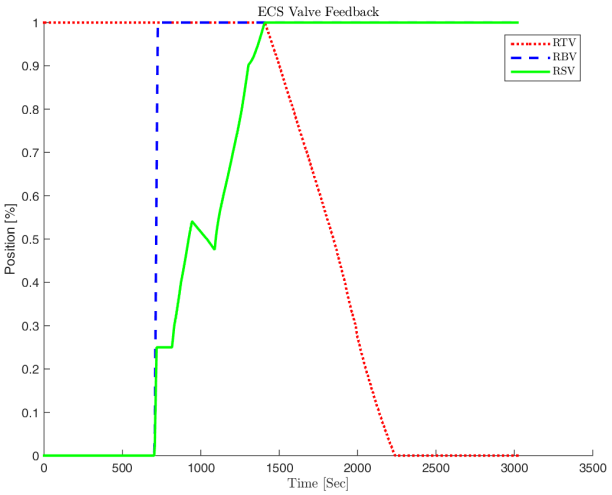

Figure 10. Valve feedback from the SCR simulation

In Figure 10, we clearly see that the SCR control works as intended, even though we have replaced the SCR heating model from the original ESU and replaced it by a Ptolemy implementation. As soon as the SCR activation occurs, the RTV and RSV valves start to open. The RTV valve is clearly controlled to balance the flow to the turbocharger. This actuation is filtered from the temperature by the low-pass behaviour of the reactor, as expected. As soon as the RTV valve is fully open the RBV valve can be closed, and output temperature keeps increasing until it eventually reaches the inlet temperature.

Each node in the simulation executes an application task running on top of the RTOS, updating variables at a specific sampling frequency. From Figure 11, we clearly see how the SCR control unit runs at 5 Hz and the engine control unit at 10 Hz. The SCR temperature is calculated in Ptolemy, resulting in the same frequency as the simulation time step of 1 ms.

5 Conclusion

This paper showed the non-trivial process of implementing FMI for co-simulation of an embedded system. We proposed to compile a target platform RTOS into an x86 architecture, which most RTOS systems support. By replacing the idle thread of the RTOS, a hook for the system clock can be provided and used to advance through the application. To match the "$Get()/Set()$" structure of the standard, the same was implemented through simulation proxies identified by unique ID numbers of target variables. The FMI API is wrapped around the x86 RTOS by loading it as a shared library, with the FMI step function "$fmi2DoStep()$" activate the RTOS clock through a callback function. The configuration of an entire control system results in a vast amount of connections, not necessary relevant for all modelling purposes. One of the advantages of the proposed method is that the configuration abstraction can be varied. If relevant, each node of the control system can be packed in individual FMUs, or all nodes can be enclosed in a single FMU, with all configuration and data/network exchange done internally. We have provided a use case where part of the engine control system is packed in an FMU and imported into Ptolemy II. By connecting the FMU to a physical model, we proved that the system could be co-simulated with an external tool, resulting in correct control system behaviour.

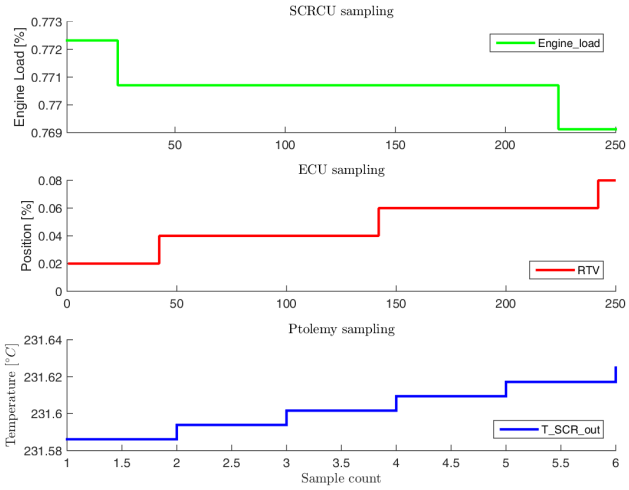

Figure 11. Illustration of the different sub-system sampling frequencies

References

Andreas Abel, Torsten Blochwitz, Alexander Eichberger, Peter Hamann, and Udo Rein. Functional mock-up interface in mechatronic gearshift simulation for commercial vehicles. *9th Int. Model. Conf.*, pages 775–780, 2012. doi:10.3384/ecp12076775.

Jens Bastian, Christoph Clauß, Susann Wolf, and Peter Schneider. Master for Co-Simulation Using FMI. *8th Int. Model. Conf. 2011*, pages 115–120, 2011. doi:10.3384/ecp11063115.

Christian Bertsch, Jonathan Neudorfer, Elmar Ahle, Siva Sankar Arumugham, Karthikeyan Ramachandran, and Andreas Thuy. FMI for Physical Models on Automotive Embedded Targets. *Proc. 11th Int. Model. Conf.*, pages 43–50, 2015. doi:10.3384/ecp1511843.

T Blochwitz, M Otter, M Arnold, C Bausch, C Clauß, H Elmqvist, A Junghanns, J Mauss, M Monteiro, T Neidhold, D Neumerkel, H Olsson, J V Peetz, and S Wolf. The Functional Mockup Interface for Tool independent Exchange of Simulation Models. In *8th Int. Model. Conf. 2011*, pages 173–184, 2009. doi:10.3384/ecp12076173.

David Broman, Christopher Brooks, Lev Greenberg, Edward A. Lee, Michael Masin, Stavros Tripakis, and Michael Wetter. Determinate composition of FMUs for co-simulation. *2013 Proc. Int. Conf. Embed. Software, EMSOFT 2013*, 2013. doi:10.1109/EMSOFT.2013.6658580.

Christopher Brooks, Edward A Lee, and Stavros Tripakis. Exploring Models of Computation with Ptolemy II. *10 Proc. eighth IEEE/ACM/IFIP Int. Conf. Hardware/software codesign Syst. Synth.*, pages 331–332, 2010.

Atiyah Elsheikh, Muhammed Usman Awais, Edmund Widl, and Peter Palensky. Modelica-enabled rapid prototyping of cyber-physical energy systems via the functional mockup interface. *2013 Work. Model. Simul. Cyber-Physical Energy Syst. MSCPES 2013*, pages 1–6, 2013. doi:10.1109/MSCPES.2013.6623315.

IMO. MARPOL : Annex VI and NTC 2008 with guidelines for implementation. Technical report, 2008.

Edward A. Lee, Mehrdad Niknami, Thierry S. Nouidui, and Micheal Wetter. Modeling and Simulating Cyber-Physical Systems. *2015 Int. Conf. Embed. Softw.*, pages 115–124, 2015. doi:doi: 10.1109/EMSOFT.2015.7318266.

Jie Liu, Xiaojun Liu, and Edward A Lee. Modeling Distributed Hybrid Systems in Ptolemy II. *Proc. 2001 Am. Control Conf.*, 6:4984–4985, 2001. doi:10.1109/ACC.2001.945773.

Modelon. FMI Library. URL `http://www.jmodelica.org/FMILibrary`.

Thierry Nouidui, Michael Wetter, and Wangda Zuo. Functional mock-up unit for co-simulation import in EnergyPlus. *J. Build. Perform. Simul.*, 7(3):192–202, 2014. doi:10.1080/19401493.2013.808265.

Nicolai Pedersen, Jan Madsen, and Morten Vejlgaard-Laursen. Co-Simulation of Distributed Engine Control System and Network Model using FMI and SCNSL. *10th IFAC Conf. Manoeuvring Control Mar. Cr. MCMC 2015*, 48(16):261–266, 2015. doi:10.1016/j.ifacol.2015.10.290.

QTronic. FMU SDK. URL `https://www.qtronic.de/en/fmusdk.html`.

Christoph Stoermer and Ghizlane Tibba. Powertrain Co-Simulation using AUTOSAR and the Functional Mockup Interface standard. *Proc. 51st Annu. Des. Autom. Conf. Des. Autom. Conf. - DAC '14*, (March):1–1, 2014. doi:10.1145/2593069.2602975.

Luigi Vanfretti, Tetiana Bogodorova, and Maxime Baudette. Power system model identification exploiting the Modelica language and FMI technologies. *2014 IEEE Int. Conf. Intell. Energy Power Syst. IEPS 2014 - Conf. Proc.*, pages 127–132, 2014. doi:10.1109/IEPS.2014.6874164.

Edmund Widl, Wolfgang Muller, Atiyah Elsheikh, Matthias Hortenhuber, and Peter Palensky. The FMI++ library: A high-level utility package for FMI for model exchange. *2013 Work. Model. Simul. Cyber-Physical Energy Syst. MSCPES 2013*, 2013. doi:10.1109/MSCPES.2013.6623316.