Downloaded from orbit.dtu.dk on: Mar 29, 2019

DTU Library

Co-Simulation of Cyber-Physical System with Distributed Embedded Control

Pedersen, Nicolai

Publication date:
2018

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Pedersen, N. (2018). Co-Simulation of Cyber-Physical System with Distributed Embedded Control. Technical University of Denmark.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Co-Simulation of Cyber-Physical System with Distributed Embedded Control

Nicolai Pedersen

Kongens Lyngby 2017

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary (English)

Cyber-Physical Systems (CPS) are integrations of computation and physical processes, with distributed embedded computation units, connected by network, controlling and monitoring a physical plant. The development of physical components is essentially different from the object-oriented software of the computation units. A major challenge developing CPS, is the nonlinear interaction between the discrete domain of the computational units and the continuous domain of the physical process. Model based development of both discrete and continuous systems has significantly benefited from specialized modelling and simulation tools in each domain. However, to realize the full potential of CPS, the abstraction-level of models and simulation has to unify both computation and physical dynamics. A solution to this, is a so called co-simulation where the coupled problem is divided into sub-systems where each constituent model can be solved by its optimum tool/solver in a distributed manner. This enables domain expert to work in domain specific tools while being able to simulate the complete CPS in a holistic manner.

This dissertation provides a solution for doing co-simulation of CPS with distributed embedded control. This research has been conducted in collaboration with MAN Diesel & Turbo (MD&T) using their CPS, consisting of a two-stroke low speed engine with a distributed engine control system, as case study. Adapting a distributed control system to enable co-simulation is not trivial. How the lower layers of the embedded system software has been adapted to enable a deterministic and temporally controlled simulation will be presented. This includes how multiple controllers are compiled to dynamic link libraries that can be executed in parallel by a main process. A method for controlling execution and time progression on each controller has been developed along with a schedul-

ing and network communication solution. To enable co-simulation with tools for modelling physical dynamics, the Functional Mockup Interface (FMI) standard for co-simulation has been implemented in the control system simulation.

The solutions presented are validated through a set co-simulation experiments using the MD&T engine control system and different physical dynamic modelling tools. During the research new applications and requirements to the co-simulation environment was discovered. In large organizations like MD&T, tools, platforms and architecture used by different departments often deviate, making co-simulation and model exchange difficult. In collaboration with the EU Horizon 2020 project; Integrated Tool-chain for the model based design of Cyber-Physical Systems (INTO-CPS), a distributed co-simulation was made possible, that was able to co-simulate sub-systems of any architecture (32/64-bit) and platform (Windows/Linux). Furthermore, when developing safety critical CPS that include a Human Machine Interface (HMI), the human interaction and cognitive assessment is of great importance. However, it is often difficult to obtain quantitative and evidence based data on the human in the loop. With an extension to the co-simulation environment it is possible to connect the control system simulation with the HMI in a hybrid co-simulation. In the hybrid co-simulation scenarios requiring human interaction can be formulated and tracked. The collected data can be used for analyzing the system applicability and intuitiveness, insuring correct and secure operation of MD&T engines.

Validation and verification on hardware and engine test-benches is a major part of the development cost at MD&T. With the possibility of simulating the complete distributed control system, engineers are able to verify more of the component design before moving to the hardware test-bench. Furthermore, by introducing co-simulation, engineers can investigate and validate the holistic system dynamics during development before moving to the Engine test-bench and do model sharing between departments, reducing redundant modelling efforts. This research provides a solution for doing co-simulation of CPS with distributed control and proves that co-simulation can improve the development process, by reducing the amount of design and test loops during the design phase, thereby reducing the overall verification and validation cost.

Summary (Danish)

Cyber-fysiske systemer (CPS) integrer computerbaserede og fysiske processer, hvor distribuerede indlejrede computer-enheder, forbundet via netværk, kontrollerer og monitorerer et fysisk anlæg. Udviklingsprocessen af fysiske komponenter adskiller sig fundamentalt fra den objekt orienterede udvikling af software til computer-enhederne (Controllerne). Den store udfordring ved udvikling a CPS er den ulineære interaktion mellem computer enheder i det diskrete domæne, og fysiske processer i kontinuert domæne. Specialiserede modellerings og simulerings værktøjer inden for hvert domæne, har markant forbedret den model basered udvikling af både diskrete- og kontinuere systemer. Kompleksiteten af CPS gør imidlertid, at abstraktionsniveauet for modellerings- og simulerings værktøjerne, bliver nødt til at forene begge domæner, for at realisere systemernes fulde potentiale. En løsning på dette er co-simulering, hvor det sammenkoblede problem bliver opdelt i delsystemer, som kan løses distribueret og anvende specialiserede værktøjer og solvere. Co-simulering gør det muligt for domæne eksperter at arbejde i domæne specifikke værktøjer, og samtidig have mulighed for en holistisk simulering af det fulde CPS.

Denne afhandling præsenterer en løsning på, hvordan et CPS, med distribueret indlejret kontrol, kan co-simuleres. Forskningen er blevet udført i samarbejde med MAN Diesel & Turbo (MD&T) of deres CPS blevet brugt grundlag for løsningen. MD&Ts CPS er en to-takts lavhastigheds motor med dertilhørende motor kontrol system. At co-simulere et distribueret kontrol system er ikke trivielt og kræver tilpasning. Det vil blive præsenteret, hvordan de nedre lag af det indlejrede systems software er blevet ændret for at kunne blive simuleret på en både deterministisk og tidslig kontrolleret måde. Dette inkludere, hvordan flere controllere kan blive kompileret til dynamisk linkede biblioteker, der kan

eksekveres parallelt af en hovedproces. En metode hvorpå eksekveringen og den tidslige fremdrift kan kontrolleres på hver enkelt kontroller vil blive præsenteret sammen med en løsning på schedulering og netværks kommunikation. For at muliggøre co-simulering af kontrol systemet sammen med andre modelleringsværktøjer, er co-simulerings standarden Functional Mockup Interface (FMI) blevet implementeret.

Løsningen er validret gennem en række co-simulering eksperimenter med MD&Ts motor kontrol system og forskellige fysisk dynamik modellerings værktøjer. Under forskningsforløbet, blev flere krav og anvendelser af co-simulering miljøet identificeret. I større organisationer som MD&T, varierer værtøjer ofte i både platform(Windows/Linux) og arkitektur (32/64-bit). I samarbejde med EU Horizon 2020 projektet Integrated Tool-chain for the model based design of Cyber-Physical Systems (INTOCPS), blev en distribueret co-simulerings løsning udviklet. Denne er i stand til at kombinere delsystemer, der varierer i både arkitektur og platform. Ydermere, er udviklingen af sikkerhedskritiske CPS, som inkluderer et Human Machine Interface (HMI), afhængig af menneskelige interaktion og kognitive forståelse. Det er imidlertid svært at få kvantitativ og evidensbaserede data om den menneskelige faktor. En udvidelse til co-simulerings miljøet gør det muligt at forbinde kontrol systems simulering med MD&Ts HMI, i en hybrid co-simulering. I den hybride co-simulering kan scenarier blive formuleret som kræver menneskelig interaktion. Data fra simulering kan opsamledes of bruges til at analysere systemets anvendelighed, og sikre at MD&Ts motorer bliver opereret, på en korrekt og sikker måde.

Verifikation og validering på hardware og motor test opstillinger bærer en stor del af udviklingsudgifterne hos MD&T. Med muligheden for, at simulere det fulde distribuerede kontrol system, kan udviklere verificere størstedelen af deres design, inden de behøver, at foretage test på hardware opstillinger. Ydermere gør den holistiske tilgang med co-simuleringen det muligt, at analysere og validere det samlede system, uden tilgang til motor test opstillinger. Co-simulering gør det også muligt at dele modeller i mellem afdelinger og med underleverandører, hvilket forbedrer samarbejde, og reducerer redundant udvikling. Resultaterne fra dette projekt har bevist at co-simulering kan optimere udviklingsprocessen af CPS, ved at reducere mængden af design og test løkker og derved den overordnede udgift ved validering og test.

Preface

This dissertation is submitted as partial fulfillment of the requirements for the degree of Doctor of Philosophy (Ph.D.) in Engineering at the Technical University of Denmark, Department of Applied Mathematics and Computer Science, Section for Embedded Systems Engineering. The project has been founded partly by the Danish branch of MAN Diesel & Turbo SE, The Confederation of Danish Industry and the Danish Agency for Science, Technology and Innovation. The research has been supervised by Professor Jan Madsen, Head of ESE Section, and from MAN Diesel & Turbo, Morten Vejlgaard-Laursen, Head of Emission Reduction System, followed by Senior Manager Henrik Rechnagel Olesen, Head of Control & Monitoring.

This dissertation deals with co-simulation of Cyber-Physical Systems (CPS) with distributed embedded control. The CPS at MAN Diesel & Turbo consisting of an Engine Control System and a Two-stroke low speed engine has been used as basis for the research.

The dissertation is a summary report consisting of 7 chapters and a collection of research papers written and published during the period of 2014-2017.

Lyngby, 01-October-2017

Nicolai Pedersen

Acknowledgements

First of all I would like to thank MAN Diesel & Turbo (MD&T) and the Technical University of Denmark (DTU) for giving me the opportunity of pursuing my PhD within the Embedded Systems Engineering (ESE) group.

I especially would like to thank my advisor Professor Jan Madsen, Deputy Director at DTU Compute and Head of the ESE Section. Jan has provided me with valuable guidance and support, always pushing me to define my own ambitions and direction of the research.

Moreover, my gratitude goes to my colleagues at MD&T; Enrique Vidal Sánchez and Marco Fam for their immense support and providing a great working environment. Especially, I would like to thank Tom Bojsen for his indispensable insights to MD&T software and countless hours of assistance, Morten Vejlgaard-Laursen and Henrik Rechnagel Olesen for their confidence and guidance.

I am thankful for my collaboration with the INTO-CPS group at Aarhus University. In-particular to Professor Peter Gorm Larsen, Head of the Software Engineering Group, for a great collaboration and many interesting discussions.

Finally, I would like to thank my family for their love and support; parents Jette Pedersen and Hans Pedersen and my sister Signe Pedersen. Last but not least, I am grateful to my girlfriend Nina Due Halvorsen, for hours of proof reading and keeping my spirits up when things got tough.

Contents

Chapter 1

Introduction

Designing the next generation of distributed embedded control systems, for the control of Cyber-Physical Systems (CPS), require advanced modeling and simulation. Model-based development enables engineers to work at higher abstraction levels, making it possible to identify and verify an optimal solution at an early stage of development. This limits the amount of test and redesign iterations that the system has to go through and thereby the cost of verification and validation, but also prevent damaging expensive equipment or committing to inappropriate hardware or software solutions. Simulation of embedded systems is not trivial due to the dependability of both software and hardware. Furthermore, most embedded systems are uniquely designed for specific purposes and only a subset of components are similar across different systems, which limit the amount of generic simulators available. In recent years, control systems are becoming more distributed, interconnected and with increased algorithm complexity, the control system dynamics are starting to significantly influence the physical dynamics and vice versa. This nonlinear interaction between the discrete domain, of the control system, and the continuous domain of the physical process, is a major challenge when developing CPS. To realize the full potential of a CPS, the abstraction-level of models and simulation has to unify both discrete computation and physical dynamics. Where previously, control systems could be developed with low fidelity models of the physical dynamics and physical plant with simplified control dynamics, it is now desirable to have a holistic model of the coherent system. In practise this means, mixing of Discrete Event

(DE) models of the control algorithms and high fidelity Continuous Time (CT) models of the physical dynamics. Multiple tools and frameworks exists for the development of both DE and CT models each with their specific specialization and validity. The tools and frameworks used by different engineering disciplines deviate significantly in their implementation language, concept and integration of time, mathematical solvers etc. making it very difficult to model the holistic system in a single tool or environment. A solution for unifying the DT and CT domains and overcoming this tool-chain deviations between engineering disciplines, is to simulate each of the sub-systems, of the coupled problem, in a distributed manner, a so called co-simulation. Co-simulation makes it possible for domain experts to work on their sub-subsystem in the tool of preference, while still being able to simulate the complete CPS as one holistic model.

1.1 Background and Research Objectives

This research was conducted in collaboration with MAN Diesel & Turbo (MD&T), developing two-stroke low speed engines for marine vessel population and stationary applications, such as power plants. In recent years, legislation on engine emissions and efficiency, along with a demand for alternative fuel types, is increasing the system complexity and amount of distributed sub-systems for the traditional two-stroke engine. This increased complexity and distribution, makes the cooperation between engineers developing control systems and the physical plant, vital for being able to deliver the future line of MD&T engines with increased efficiency and lower emissions. Currently, low fidelity models of the engine physics are used to develop control system algorithms, and on the other hand simplified control dynamics are used for the engine performance calculations for sizing and commissioning. Furthermore, the development of the control system used to be based on a Software In the Loop (SIL) simulation that only model the upper application layers of individual controllers without valid modelling of the distributed aspect of the engine control system. These challenges were dealt with using Hardware In the Loop (HIL) and engine testbenches for verification and validation. However, with the increased system complexity foreseen in the future, and the cost of both HIL and engine testbenches, it is desirable for MD&T to increase their modelling and simulation efforts to optimize their development process.

Two main objectives were identified as the basis for this research. First objective was to enable modelling and simulation of the distributed embedded control system. The simulation should be deterministic to ensure reproducibility and enable engineers to do e.g. regression testing and verify functionality at an earlier stage of development. The second objective was to enable co-simulation

of the control system together with a high fidelity physical dynamic model expressed in a dedicated tool. Co-Simulation will enable engineers to investigate and validate holistic system dynamics during development and enable model sharing between departments, reducing redundant modelling efforts.

1.2 Co-Simulation Survey

Cyber-Physical Systems integrate both physical and computational processes. The development of physical components is based on CT models which is essentially different from the DE modelling of the object-oriented software components. To realize the potential of CPS, the abstraction-level of models and simulation has to unify both computational and physical dynamics Lee (2008). Co-simulation and co-modelling has been proposed as solutions to overcome the challenges of unifying models and simulations from different domains including DE and CS. Models used in co-modelling describe both physical and computational components in a unified language and simulates the model as one. While this approach has advantages, each domain has its own limitations and particularities making it difficult to find a single language that fits all applications Gomes et al. (2017); Fitzgerald and Gorm (2014). In co-simulation a coupled problem is divided into sub-systems where each constituent model is solved in a distributed manner. This enable each sub-system to be developed in domain specific tools by domain experts and simulated in a global holistic simulation. Coordination and control of the simulation is orchestrated by a co-simulation manager responsible for exchanging data-between sub-systems and the progression of time Fitzgerald and Gorm (2014). Applications within co-simulation has been published in multiple domains. Examples within the automotive industry are Abel et al. (2012); Brezina et al. (2011); Li and He (2011); Mews et al. (2012); Stoermer and Tibba (2014), within energy and grid systems Elsheikh et al. (2013); Vanfretti et al. (2014); Bian et al. (2015); Lin et al. (2011) and HVAC Nouidui et al. (2014); Dols et al. (2016); Hafner et al. (2013), including other domains as well. The maritime industry share many similarities with the other industries especially the automotive, however, there are many significant differences. Besides from the obvious diversity of the physical engine like two-stroke vs four-stroke combustion, cooling system, electrical system and fuel system, also the control system has some key differences. While both systems consist of multiple microprocessors connected by a network, in the automotive industry Electronic Control Units connected by a CAN bus, the control challenges are very different. In modern automotive four-stroke engine the piston revolution range roughly from 800-7000 RPM whereas a low-speed two-stroke marine engine range on average from 50-100 RPM, this makes the control strategies used in the different industries significantly different. Fur-

thermore, the most important difference is the size of the two industries and the consequences that follow. The automotive industry sold 88.1 million units in 2016[1] divided on more than 50 car manufactures as compared to the around 2300 vessels sold by only a very few suppliers. This huge contrast makes the amount of specialized tools, Original Equipment Manufacturers (OEM) and research done in the different fields very unequal. The only publications on this specific topic within the maritime industry is the publications originating from this research Pedersen et al. (2017a); Pedersen et al. (2016); Pedersen et al. (2017b); Pedersen et al. (2015).

Common for the publications mentioned above is that they, either use a multi-domain tool like the commercial MATLAB/Simulink or Ptolemy II Eker et al. (2003); Awais et al. (2013) from UC Berkeley, that can model both DE and CS and connect a single specialized tool through a custom co-simulation interface or use a co-simulation standard for coupling sub-systems. Several co-simulation standards has been developed the past years. Most widely accepted is the High Level Architecture (HLA) Dahmann (1997) and the Functional Mock-up Interface (FMI) Blochwitz et al. (2009).

While multi-domain tools are very powerful, they force developers to commit to the multi-domain tool to drive the co-simulation. This is especially troublesome when dealing with sub-suppliers or large global organizations. The multi-domain tools mentioned, MATLAB/Simulink and Ptolemy II, are state of the art and also offer a vast variety of co-simulation capabilities, meaning the possibility of connecting a simulation from another tool to the host simulation tool. These co-simulation interfaces are either based on standards like the HLA and FMI or custom made interfaces. With the large amount of tools available at MD&T no multi-domain tool was found that offers all the interfaces required. Adapting the tools that are not supported is possible, but would require a large amount of change management and more maintenance then choosing a co-simulation standard, while not committing to a single tool-chain.

The HLA is a platform independent standard interface for distributed co-simulation. It uses a central manager, called the RTI (Run Time Infrastructure), that receives data from various sub-systems and sends them to other sub-systems in the simulation. HLA provide a set of application interface functionality that sub-systems use and is implemented by the RTI. HLA is widely used in the military industry for inoperable distributed simulation of large scale simulations. While the standard is widely accepted it focus mostly on networked data exchange between sub-systems and not on the dynamic dependability and coupling of e.g. an engine and its turbo-charger.

[1] Source: National car data, Macquarie Research, January 2017

In recent years FMI has become a widely accepted standard. FMI was initially developed for the automotive industry, to enable better collaboration with OEMs, an industry which MD&T shares many similarities, including many of the same tools. 95 tools are currently supporting the FMI standard, many of which MD&T departments currently use and could be part of the co-simulated environment in the future. The Functional Mockup interface was chosen as co-simulation standard and will be described further in chapter 4.

1.3 Scope and Limitation

This research aimed to provide an approach for doing co-simulation of Cyber-Physical Systems with distributed control, by analyzing the CPS developed at MD&T. The scope was to investigate how a distributed embedded control system can be simulated in a deterministic manner to enable Co-simulation and how the FMI standard can be implemented. The MD&T system is generic in its architecture and the solutions proposed should be applicable to many other systems. The co-simulation environment was verified by a set of experiments and the development process at MD&T analyzed to investigate what requirements and features are relevant for companies developing CPS.

1.4 List of Publications

The following articles has been peer-reviewed and published during the research:

1. Pedersen, N., Madsen, J., and Vejlgaard-Laursen, M. (2015). Co-Simulation of Distributed Engine Control System and Network Model using FMI and SCNSL. *IFAC-PapersOnLine*, 48(16):261–266

2. Pedersen, N., Bojsen, T., Madsen, J., and Vejlgaard-Laursen, M. (2016). FMI for Co-Simulation of Embedded Control Software. In *Linköping Electronic Conference Proceedings*, number 124, pages 70–77. MAN Diesel & Turbo, Copenhagen, Denmark, Linköping University Electronic Press, Linköpings universitet

3. Pedersen, N., Bojsen, T., and Madsen, J. (2017a). CO-SIMULATION OF CYBER PHYSICAL SYSTEMS WITH HMI FOR HUMAN IN THE LOOP INVESTIGATIONS. *Proceedings of the Symposium on Theory of Modeling & Simulation*, pages 1:1–1:12

4. Pedersen, N., Lausdahl, K. G., Sanchez, E. V., Larsen, P. G., Madsen, J., Vidal, E. S., Lausdahl, K. G., Madsen, J., Pedersen, N., and Larsen, P. G. (2017b). Distributed Co-Simulation of Embedded Control Software with Exhaust Gas Recirculation Water Handling System using INTO-CPS. In *Simultech 2017*, pages 73–82

1.5 Structure of Dissertation

The dissertation will begin in chapter 2 with an introduction of the Cyber-Physical System at MAN Diesel & Turbo to be modelled and simulated. How the distributed engine control system has been adapted to enable modeling and simulation of the software will be presented in chapter 3. Chapter 4 will describe the co-simulation interface chosen, FMI, and how the standard has been implemented in the control system simulation. Platform and architecture deviations within the MD&T tool-chain made a distributed co-simulation necessary, as will be presented in chapter 5. The new simulation approach, opened up for the possibility of simulating the engine control system together with the human machine interface in a hybrid-co-simulation, and perform human in the loop investigations, as illustrated in chapter 6. Chapter 7 will draw conclusions and put the solution into perspective. Chapters 8, 9, 10, 11 gather the publications originating from this research.

Chapter 2

MAN Diesel & Turbo

MAN Diesel & Turbo (MD&T) is the power engineering business unit of the MAN Group, a subsidiary of MAN SE, which has been part of the Volkswagen Group since 2011. MD&T develop two-stroke low speed and four-stroke medium/high speed engines for marine population and stationary applications such as power plants. Besides from the core engine, they also deliver complete marine population systems, from propellers all the way to turbochargers. A separate turbo-machinery department develop and produce compressors, gas and steam turbines. MD&T also has its own service brand named MAN PrimeServ, who provide global 24/7 service of OEM parts. This research focus on the two-stroke low speed engines developed at the Copenhagen office of MD&T. Historically speaking the two-stroke engine production was purchased by MAN AG from the danish ship-yard and diesel engine producer Burmeister & Wien (B&W) in 1980. B&W had roots stretching back to 1846, building the world's first ever ocean-going diesel-powered vessel, M/S Selandia, and the world's largest diesel engine, at the time in 1933. The first turbocharged two-stroke diesel engine was commissioned in 1952 with a design that laid the foundation for the engines built today.

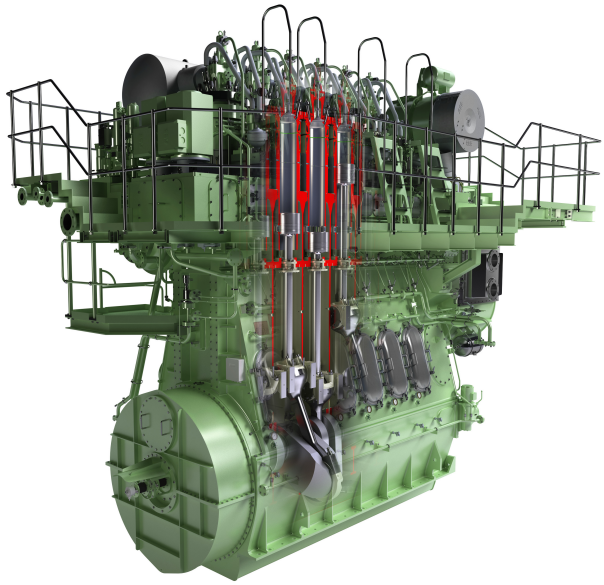

Figure 2.1: MAN Diesel & Turbo 6 cylinder electronically controlled ME-C engine.

2.1 The ME-C Engine

In 2002 the MD&T engines became electronically controlled and the company moved into the field of Cyber-Physical Systems. The line of electronically controlled engines are named ME-C, where the fuel injection and valve opening is electronically controlled by the Engine Control System (ECS). The ME-C engines brake power range from 4,350 kW to 82,440 kW with a stroke length and cylinder diameter from 1,550mm stroke and 30cm diameter to 3,460mm stroke and 98cm diameter over 4 to 12 cylinders. ME-C engines are powered by fuel oil both distillate fuels and residual fuels. MD&T also delivers dual fuel engines able to switch between fuel oil and a gas supply. ME-C-GI engines can run on methane, ME-C-GIE on ethane, ME-C-LGIM on methanol and ME-C-LGIP on LPG (mixture of liquid propane and butane). To comply with the International Marine Organization's (IMO) emission regulations on marine engines, currently the Tier III emission protocol Organization. (2013), engines can be fitted with either a Exhaust Gas Regulation (EGR) system or Selective Catalytic Reduction (SCR) system for reduction of NO_x emission.

2.2 Engine Control System

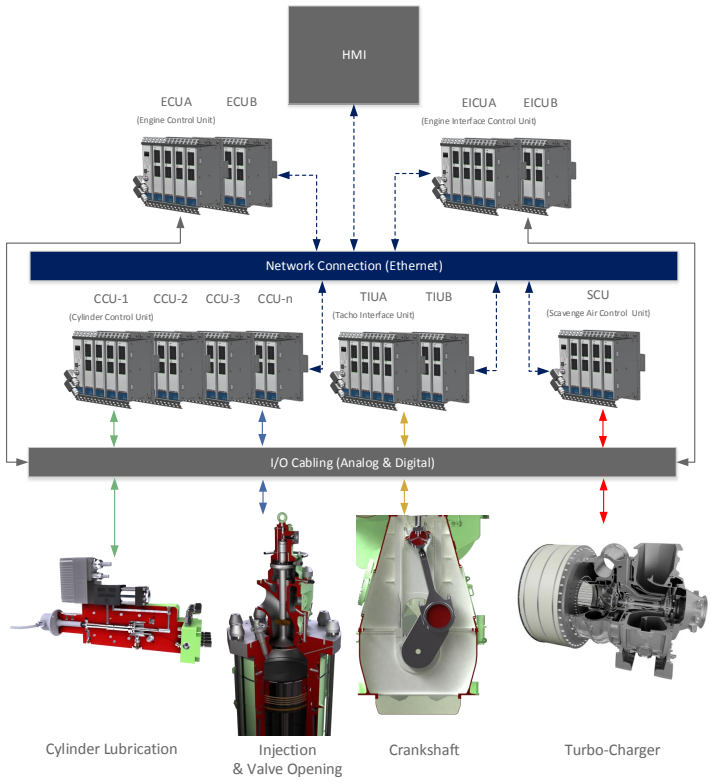

Figure 2.2: The distributed engine control system.

The Engine Control System is a distributed embedded control system consisting of between 14 to 33 individual controllers, depending on the amount of cylinders, auxiliary systems installed and engine version. All controllers are hardware-wise identical but have individual control objectives, determined by the software running on the embedded system. Figure 2.2 illustrates a subset of controllers in the ECS and the components they are controlling. Controllers interact with the engine through sensors and actuators connected by IO cables both analog and digital. An Ethernet network connects all controllers in the ECS and the Human Machine Interface (HMI). The ECU, EICU and TIU controllers are redundant as a fault tolerance measure.

The main objective of the ECS is to ensure an efficient and clean combustion that delivers the population power requested. The Cylinder Control Unit (CCU)

is responsible for calculating the injections profile and performing injection, the opening of exhaust valves and cylinder lubrication. Each cylinder has a dedicated CCU where the orchestration of timing between all cylinders is governed by the Engine Control Unit (ECU). Information about position and velocity of each cylinder is provided to the ECU by the Tacho Interface Unit (TIU), which measures the engine crankshaft position. The Engine Interface Control Unit (EICU) connects the HMI with the ECS. From the HMI, engine running modes, load, emission modes, fuel calibration ect. is controlled. The Scavenge Air Control Unit (SCU) regulate the turbo-charger by balancing sufficient scavenge air and exhaust pressures. Besides from the core engine controllers, also auxiliary systems are controlled by the ECS; the Auxiliary Control Units (ACU) are controlling e.g. hydraulics and start air blowers. If an emission reduction system such as SCR/EGR is installed, dedicated SCR/EGR Control Units and SCR/EGR Interface Units will be installed, ensuring correct emission reduction while maintaining engine performance.

2.2.1 The Triton Controller

The Triton controller is the new generation of MAN Diesel & Turbo ECS controllers. Triton controllers are modular and can be tailored to different engines and control features by adding boards with different functionality.

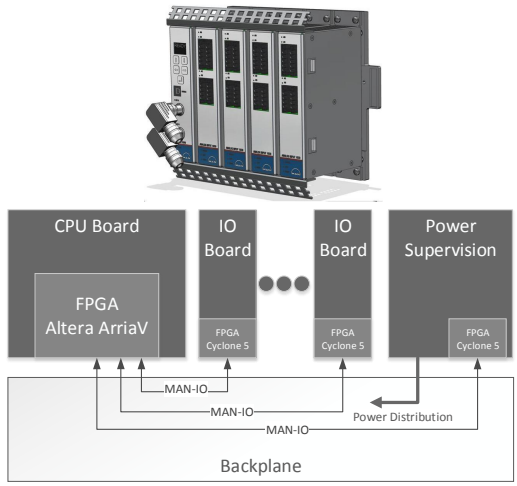

Figure 2.3: The MD&T Triton controller.

As can be seen on figure 2.3, the design is based on a single crate with space for

8 boards connected through a back-plane. There are three main types of boards consolidating a Triton controller. The CPU board is where the software from the control application is executed. IO boards can be either Analog or Digital and handle communications with sensors and actuators installed on the engine. The power supervision board distributes and monitors power to the controller. Each controller in the ECS will contain a CPU and Power Supervision board, but the amount of IO-boards are determined by the control objective of the specific controller (e.g. a Cylinder Control Unit or Engine Interface Unit).

2.2.1.1 CPU Board

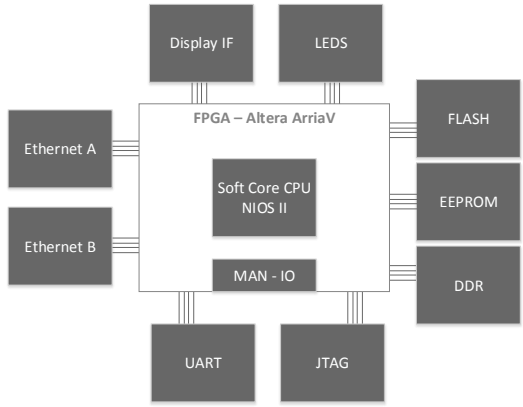

Figure 2.4: Diagram of the Triton CPU board.

The CPU boards main components are illustrated on figure 2.4. The FPGA is a Altera ArriaV containing all the control functionality, a NIOS II soft-core CPU and the MAN IO module that uses a star-typology to connect all the boards with the CPU as center hub. The CPU board has two physical Ethernet ports and runs the Parallel Redundancy Protocol (PRP). To show the status of the systems to an operator, the controller has two LEDs and a small display connected to the FPGA. The board has two flash memories, one containing the production image to be loaded on the FPGA and the other holding the application and driver software. Module information is stored in an EEPROM, this information is e.g. used to tell the software and the rest of the control system what the control objective of the controller is; a cylinder control unit or engine interface control unit ect.. The DDR module is the CPU main memory. The JTAG unit provides a debugging interface that can load images on the FPGA and connect to chip-scope for debugging purpose. Finally the UART interface can be used for terminal access to the system.

2.2.1.2 IO Board

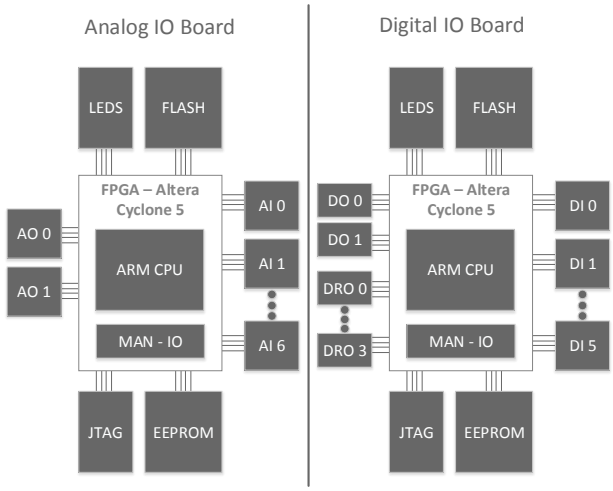

Figure 2.5: Diagram of the Triton IO boards.

The IO boards are divided into either an Analog or Digital board. Each board has an Altera Cyclone 5 FPGA with an ARM CPU containing all control functionality and MAN IO CPU. LEDs are available to show system status and a JTAG module for debugging. The flash contain the FPGA image to be loaded on start-up and EEPROM the module information. Analog IO Boards have 2 analog output and 7 analog input signals. The digital IO Board has 2 digital outputs, 4 digital relay outputs and 6 digital input signals.

2.2.1.3 Power Supervision board

The power supervision board is similar to the IO Board from 2.2.1.2, but without the IO components. Instead it has a power supervision module that monitors the input power and fuses for protecting the controller against power spikes and short circuits.

2.2.2 Software Architecture

The architecture of the software running on the Triton controller, is illustrated on figure 2.6. A Board Support Package (BSP) for the NIOS II soft-core on the

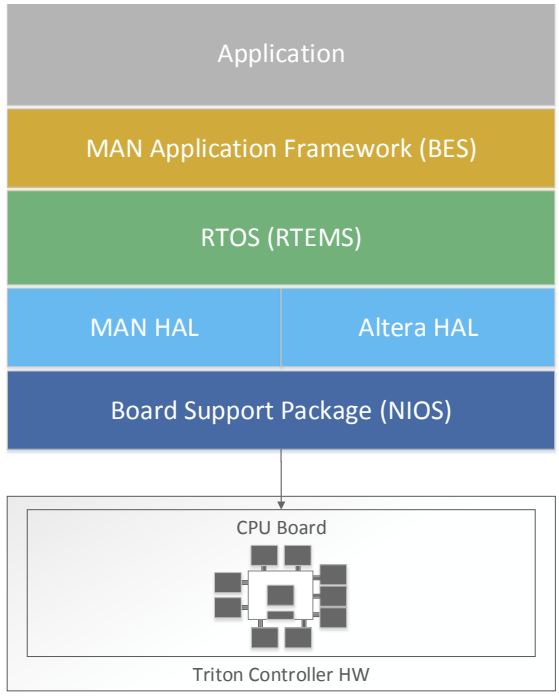

Figure 2.6: Illustration of software architecture.

FPGA provides the low level drivers for interfacing with the hardware. The Altera Hardware Abstraction Layer (HAL) provides access to some of the regular features of the FPGA. The MAN HAL layer is developed at MD&T and provides access to customized features of the hardware which are specific to the MD&T Triton controller. As most embedded systems the software runs a Real Time Operating System (RTOS), the specific operating system used, is the Real-Time Executive for Multiprocessor Systems (RTEMS). RTEMS was designed for real time execution of embedded systems and support nearly all POSIX services. Engineers designing control applications, work directly in C++ code that can be directly deployed to the controller. An application framework between the operating system and the application code has been created to ease the coding efforts required by application engineers. This framework is developed at MD&T, in the department called Basic Electronic and Software (BES), thus the name BES-framework. The BES-framework provides a simplified API for application developers to e.g. create tasks, fix-point conversion, access hardware such as timers, IO's ect.. The application layer is where control algorithms are developed and implemented.

2.2.3 Human Machine Interface

Figure 2.7: The human machine interface.

The Human Machine Interface (HMI) of the MD&T ME-Engine consist of a Engine Control Main Operating Panel (EC-MOP) and a 3rd party propulsion control system with the main throttle handle, important gauges and manual switches for specific pumps and other actuators. In this dissertation only the EC-MOP connected to the ECS is relevant. The EC-MOP, also described in Pedersen et al. (2017a) and chapter 10, is the main HMI for engineers operating the engine. The EC-MOP is a marine approved and certified PC with a touch screen interface located on the engine control room panel. From the EC-MOP operators can carry out engine commands, adjust engine parameters, select running modes and observe the status of the control system. Communication between the EC-MOP and ECS is based on Ethernet Local Area Network (LAN). A MAN Ethernet protocol driver connects the EC-MOP with the Engine Interface Control Unit, which is connected to the rest of the distributed system through the Engine Control Unit. If, for some reason, the EC-MOP is unavailable, the engine can also be operated directly from a local operating panel located on the engine.

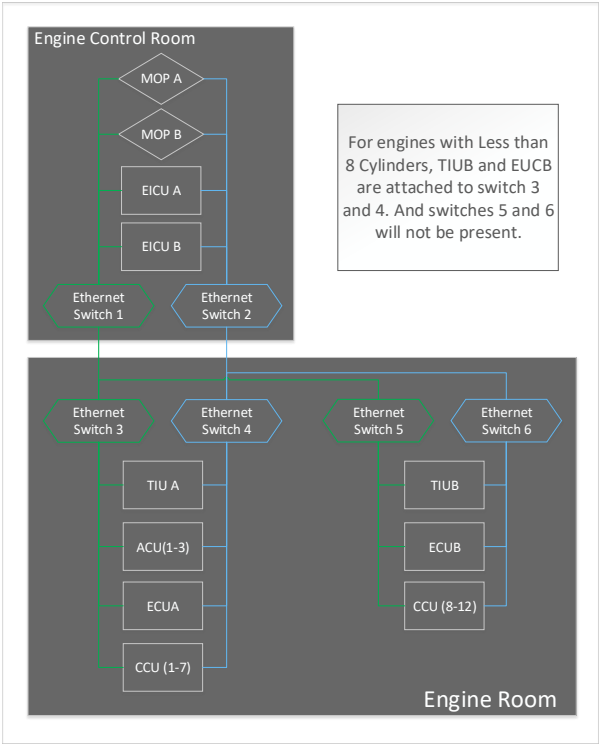

Figure 2.8: HMI and engine control system network typology.

2.2.4 Distribution and Network Typology

The distributed engine control system is connected through LAN cables and switches as seen on figure 2.8. The network is redundant as a safety critical precaution with two network adapters on each controller in the network and two EC-MOP (MOP A & B). To distinguish between the two networks the Parallel Redundancy Protocol (PRP) is used. To ease CPU load a hardware PRP component is embedded on the FPGA Ethernet module. Controllers and switches are installed in electrical enclosures in the engine room and engine control room. In the engine control room the EC-MOPs are located with the Engine Interface Control Units. Depending on the engine size and amount of auxiliary systems, the ECS have two or four switches that connects all the engine specific controllers.

The network communication is illustrated on the Open Systems Interconnection (OSI) model on table 2.1. On the physical layers (layer 1-2), the network is a

Table 2.1: Network Protocols

OSI	Ethernet			
Layers 5/6/7	Application			
Layer 4	CNET (MAN Protocol)	ARP (Address Resolution Protocol)	IEEE 1588 (Distributed Clock Synchronization)	TACHO (MAN Protocol)
Layer 3				
Layer 2	MAC			
Layer 1	802.3			

wired Ethernet LAN, with at Media Access Control (MAC) data link layer. The transport and network layer (layer 3-4) has four main protocols:

- The CNET (Control Network) is the MD&T developed protocol for communication with the control software.
- The Address Resolution Protocol (ARP) is a communications protocol used for mapping network addresses to physical MAC addresses.
- The IEEE 1588 Precision Time Protocol (PTP) is used for distributed clock synchronization between network nodes (controllers). A IEEE-1588 time-stamping component is implemented in hardware on the Ethernet module of the FPGA where it is connected to both the CPU and the hardware clock. This makes high precision synchronization of the system clock possible.
- The TACHO protocol is developed at MD&T and used to communicate the high priority information about the engine crankshaft position from the Tacho Interface Units. The FPGA Ethernet module contain a prioritizing module that splits packages into two connections used by the CPU, one for high priority packets, e.g. Tacho package, and one for all the other packets.

The upper layers (layer 5-7) are implemented in the application and not presented here.

2.3 Development Method

This research focuses on the benefits from simulation of distributed control systems and co-simulation of Cyber-Physical systems, it is therefore important to look at how new control applications are developed at MD&T.

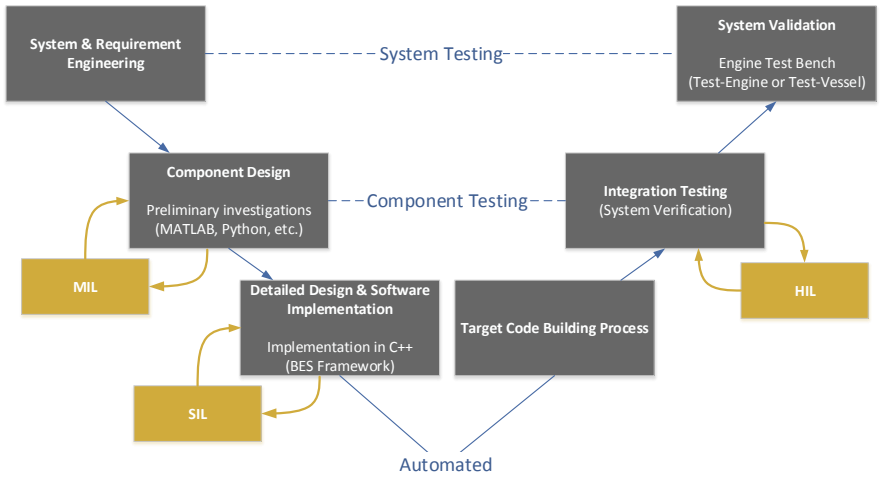

Figure 2.9: V model of development process

When a new subsystem is to be designed or functionality changed a development process is initiated as seen on figure 2.9. The first step is a cross department collaboration on system requirements and high-level design decisions. Once requirements and surrounding system dependencies are clear the specific component can be developed. For doing preliminary rapid modeling of components, multiple tools are available to engineers at MD&T. Many would use e.g. MATLAB/Simulink to do preliminary studies of the system to be designed and using Model In the Loop (MIL) simulation, models can be iterated. Once the component design is believed to be reliable, the control logic is implemented in the C++ BES-framework, described in 2.2.2, and a detailed design implemented. The software can be tested by cross-compiling the code to a PC platform where Software In the Loop (SIL) simulation can be performed, described further in 3.1. Physical dynamic models, to test the control logic against, are formulated in an internally developed tool for continuous time simulation called the Dynamic Simulation Environment(DSE) which extends the bes-framework. The primary focus in DSE is SIL and Hardware In the Loop (HIL) simulation, and the physical dynamic models implemented here are often an abstraction of high-fidelity models developed during the component design. The main advantage

of this approach is that moving from development to production code is automated and applications created during development can be directly compiled to target. Likewise, it is possible to compile the physical dynamic models, build in DSE, to a dedicated controller, called the Engine Simulation Unit (ESU) to be used during HIL testing. For system verification, a Hardware In the Loop platform is available, where the newly developed software runs on the real controller hardware. Here component and integration testing can be done using the engine dynamic models running in the ESU controller. The final step in the development process is to validate the new design on an engine test-bench, either on the test engine at the MD&T research center in Copenhagen or on one of the test vessel collaborating with MD&T.

2.3.1 Challenges with Current Method

While the development process described above, has been efficient for many years, increased system complexity challenge this process. In the transition between component design and software implementation, both control algorithm and physical dynamic models are manually translated to the C++ BES-framework. The DSE was initially intended for building models to run on HIL and by design much lower fidelity than models developed in the preliminary study, using tools such as MATLAB. System verification on the HIL test bench is important for ensuring that the controller is able to execute the control software with regards to temporal integrity and performance overhead. HIL test-benches and the engine test-bench are very limited resources at MD&T, due to the cost and size of the system. The more testing and validation that can be done during the development phase in MIL and SIL simulation the cheaper the complete validation and verification cost will become. By developing an advanced simulation of the embedded control software that is able to simulate a full distributed system, the amount of time spent on the HIL-test benches will be significantly reduced. The advantage of looking into co-simulation is to be able to connect models of high-fidelity to the ECS simulation and thereby do rapid full system validation before moving to the actual engine test bench.

2.4 Chapter Summary

This chapter introduced the Cyber-Physical System being developed at MD&T. A detailed description of the distributed embedded system was presented and the development process at MD&T introduced and challenges discussed. The cost of HIL and engine test-bench validation is immense and with the com-

plexity foreseen in the future, advanced modelling and simulation is required. Overcoming these challenges are some of the results expected as an outcome from this research by enabling co-simulation of the engine control system.

Chapter 3

Simulation of Distributed Embedded Control Systems

Modelling and simulation have become standard practice when studying and engineering complex systems. In engineering the concept of simulation requires both a model and a solver. Where a model is an entity that imitate the key characteristics, behaviors and functions of either a physical, abstract system or process and a solver is a mathematical algorithm that calculates the solution to the model states. When developing Cyber-Physical Systems we normally distinguish between continuous time simulation and discrete event simulation. Continuous simulation utilizes differential equation based models which are solved by ordinary differential equation calculators. A continuous time model represents a continuously changing system and do not require an explicit representation of the state and time relationships. Discrete event simulation utilizes a mathematical/logical model of a system that portrays state changes at precise points in simulated time. Both the nature of the state change and the time at which the change occurs mandate precise description. In general, the physical-part of a CPS is modeled in continuous time, in our case the engine physics, and the cyber-part in discrete time e.g. the engine control system. Currently the control system development at MD&T allows the source code to be cross compiled to a developers PC for SIL testing. This basically provides a discrete model of the embedded software where the application layer is identical to the real system and the lower hardware dependent layers are abstractions. This research aim

to significantly improve the fidelity of this model by introducing deterministic and temporal simulation of multiple controller models in a single coherent simulation.

3.1 Cross-Compiler for PC-x86 Execution

When developing new software it is important to continuously be able to test your work. Engineers at MD&T are working on PC workstation and with the help of a cross-compiler they can execute and test their software locally.

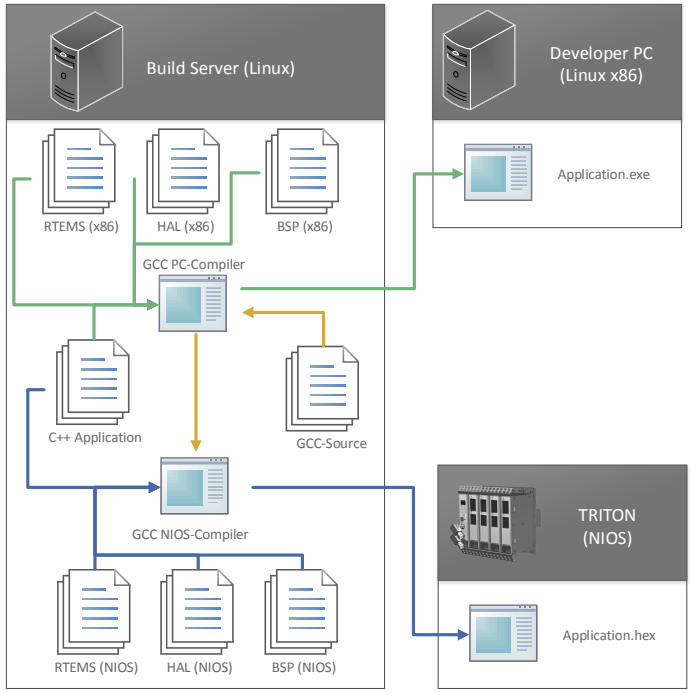

Figure 3.1: Compilation process for both PC and Controller.

The cross-compilation routine is illustrated on figure 3.1, where a build server creates two versions of the GCC compiler using the same GCC source files, one for building applications for an x86 architecture "GCC PC-Compiler", the other for building applications for the Triton controller, with the NIOS II architecture "GCC NIOS-Compiler". Each compiler uses its own version of the Board Support Package (BSP), real-time operating system (RTEMS) and Hardware

Abstraction Layers (HAL). A lot of effort has been put into modeling the PC version of the MAN-HAL layer to give a valid representation of the real system. The application C++ source files are identical for both compilers. This way it is possible to execute and test the same application code on both a developer PC and on target. The PC application is basically a SIL simulation of a single controller software and a lot of functionality can be tested in this environment before moving to the HIL test-bench. There are a lot of aspects that can not be tested in the single controller SIL simulation, such as communication with other controllers and all the temporal aspects of the distributed system. In the following sections we will describe how a multi-controller SIL simulation has been made possible, making the SIL simulation even more powerful and reducing the efforts spent on the HIL test-bench.

Figure 3.1 show how a build server distributes applications to both developer PC and target. Another way to use the compilers is to have them distributed to the Developer PC alongside all the source code. In this way engineers can write code and compile locally without the Build server.

The cross compilers and large parts of the HAL layer has been developed prior to this research. Our research show how the SIL simulation approach can be extended to enable a controlled and deterministic execution of the operating system, interrupts and network events. Where as the current simulation simply lets the operating system execute. We will also describe how a model of the complete distributed system with multiple controllers can be simulated in a deterministic and temporally correct manner.

3.2 Multiple Controller SIL Simulation

This section will describe how the SIL single controller simulation has been extended to handle the multiple controllers of the distributed ECS. One of the main challenges when simulating a distributed system is making sure that all timely aspects are valid both locally and globally.

3.2.1 Single Process Multiple Controllers

We wish to be able to run the complete software, including operating system, of multiple controllers in a single process. This is achieved by exporting the main function of RTEMS and compiling each of the controllers to a dynamic link library, in our case shared libraries, since we are running Linux. Multiple

shared libraries can be loaded by a process running on the Linux kernel and their main function be activated in a dedicated thread.

3.2.2 Manage execution of controllers

The multiple controller simulation has to be deterministic and it must be possible to start and stop execution of individual controllers.

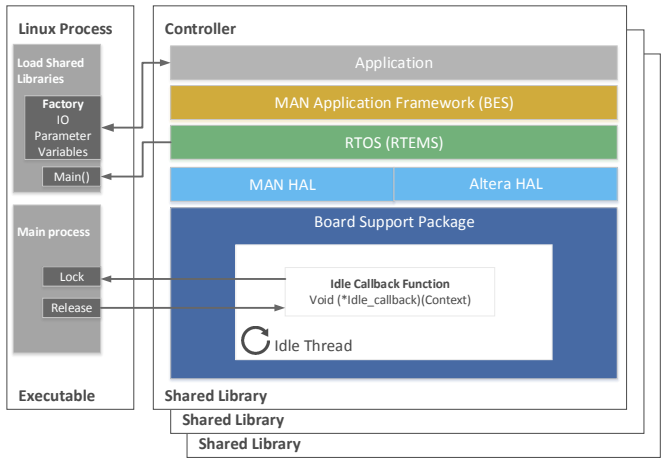

Figure 3.2: Multiple controller simulation with idle-thread callback.

The RTEMS BSP has an Idle thread, which is the task that the system returns to when all other tasks has been executed. It is by overwriting the idle thread function "bsp_idle_thread()" with a function, that takes a function pointer "bsp_idle_thread(uintptr_t context)", we can introduce a blocking callback function from the main process to the controller as illustrated on 3.2. Executing the callback function from the idle-thread will cause a context switching from the Controller to the Linux main process. Here a mutual exclusion (mutex) routine will prevent the context from returning to the controller until the mutex is released, at which point the controller can continue until it once again returns to idle.

In this way we are able start and stop execution on each controller in the simulation. Starting and stopping execution is only relevant if we can interact with the controller. ECS controllers have a tree structure of data-object with either parameter or variable type. Furthermore, it connects to the engine through Analog or Digital Input and Outputs (IO). When building a shared library con-

troller, a custom object factory is created that makes it possible for the Linux main process to create proxies for data-object in the data tree. Parameter, variable, analog and digital proxies are basically pointers to the BES/HAL instance of the data-object, that makes it possible to manipulate the object.

3.2.3 Operating System Clock Progression

It is now possible to start and stop execution on individual controllers and interact with them through proxy IOs, parameters and variables. To get a concept of time in the simulation we need to control the progression of time on each controller. The application code is scheduled by the RTEMS Operating System (OS). On target, the OS clock is ticked by an interrupt from the main hardware clock once every millisecond. In the simulation we can move the OS clock progression from the interrupt context to the idle thread. Here it is possible to execute the OS clock progression every time the program returns from the main Linux process. When the system returns back to idle, meaning that all RTEMS tasks has been executed, we determine that 1 millisecond has passed. Meaning that every time the main process releases the mutex blocking a controller, 1 millisecond of simulated time will progress locally on the specific controller. This is of course a very hard assumption, where we are basically assuming unlimited processing power, meaning that all tasks will finish and never be interrupted before returning to idle. This could cause the simulation results to deviate from a real stochastic execution. However, it ensures a deterministic simulation which is important during control algorithm development and regression testing. Our system is currently dimensioned with sufficient computational power for this to have no significant consequences. And significant testing of computational overhead and stochastic temporal aspects will still be performed on the HIL test-bench.

Now the main Linux process is able to control the individual time on each node with a resolution of one millisecond.

3.3 Interrupt Implementation

One of the ambitions with this project was to enhance the SIL simulation efforts to increase early stage test coverage and reduce time spent on HIL test-benches. With the simulated progression of the OS clock, we are able to test the application and functional layers of the system, but there are many low-level aspects of an embedded system that are also relevant to simulate. Interrupts are sig-

nals to the processor emitted by software or hardware indicating an event that needs immediate action. When an interrupt occurs, the processor will finish its current instruction and immediately switch to the interrupt routine. In the SIL simulation we have no hardware that can produce these interrupts, so they will have to be simulated. This section will show how some of the interrupt routines, important to the ECS simulation, has been implemented and scheduled.

3.3.1 Idle Event Scheduler

Introducing multiple events in the simulation will require some sort of scheduling. Our access to the controller execution is the idle thread as described previously. In the idle thread, we can create an event scheduler called the "Idle Event Handler" that can be used to manipulate the execution and progression of the controller simulation. The OS clock tick is converted to an event that reoccur every millisecond. Interrupt routines will also be implemented as events in the Idle Event Handler, as will be described in the sections below. To ensure temporal execution of events, we need a simulated local concept of time on the controller, from now on called System Time. The System Time is implemented as a free running 64-bit counter, called the system clock, similar to the free running counters available on the FPGA. The system clock is controlled by the idle thread, which means that our re-implementation of the idle thread has full control of the timely aspects in the simulation. The resolution of the system clock was chosen to be 1 µs by analysing the frequency of the events to be scheduled, the resolution is simulated and could be changed if required. Every time an event occur in idle, the system clock is progressed to the time instance of the event e.g. an OS tick will progress the System Time with 1 ms and a higher resolution event with a pulse of 200 MHz will progress the System Time with 5 µs.

On figure 3.3, we see how the Idle Event Handler is implemented as a Class with an event element structure consisting of a Time; the system time of the event, Context; context object of the event origin, and Callback; the callback function the event should execute. The Idle Event Handler is accessible throughout the controller code where modules can subscribe/unsubscribe/update their own events to be scheduled in the idle thread and not the RTEMS scheduler. Every time the system reach the idle thread, the callback to the main process will be executed, blocking the controller and handing over control to the main Linux process. The main process can access the Idle Event Handler on each controller and get the System Time of the next local event. In this way the main process can evaluate local time on each controller and schedule the execution of multiple controllers in global time. When the idle callback is released the system clock is progressed to the time of the event and the event callback is executed.

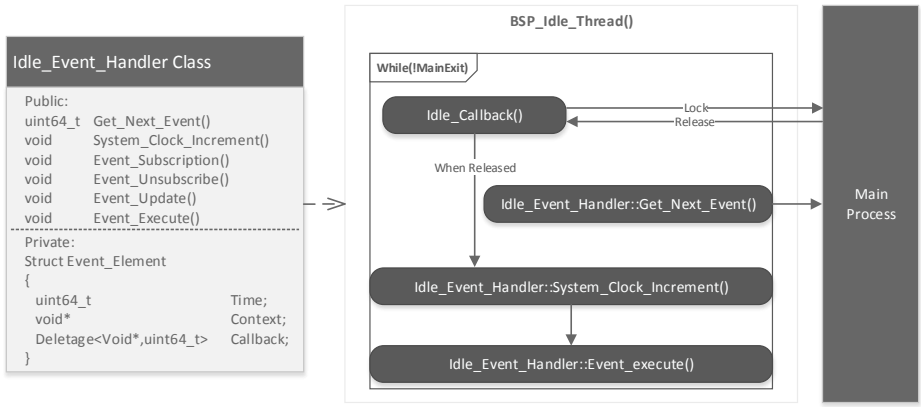

Figure 3.3: Idle Event Handler.

3.3.2 High Precision Timers

The engine control system has many control objectives that require a higher frequency and priority than the 1 ms OS clock progression. Controlling components like the FIVA-valve (Fuel Injection Valve Actuator), a 100 MHz high precision timer is used to create interrupts. In the application, developers can register a callback function to a high precision timer interrupt and define a specific interval or a certain time for the interrupt signal to occur. In hardware this will result in the HAL setting up registers for the timer to create the electronic alerting signal at the correct time. To simulate the same functionality, we have replaced the register setup with an event subscription to the Idle Event Handler in the idle thread. The subscribed event contains the System Time when the interrupt should occur, the full context of the high precision timer class and a callback function that is able to execute the high precision timer delegate from the application.

3.3.3 Time Synchronized Pulse

The Time Synchronized Pulse (TSP) is a finite state machine based on a transition table defining the current, next and error states of a timer interrupt pulse that the user can continuously regulate. TSPs are used e.g. on digital outputs doing fuel injection. From the high frequency Tacho network packages (containing crankshaft position information) and the fuel index requirement, the application will calculate a pulse with a start time (microseconds in the future) and pulse width for when a digital output needs to start injection and for how

long the injection should last. When the calculated injection position of the piston gets closer, the application will continuously try to synchronize the pulse to achieve an optimal injection profile. The TSP state machine monitors the pulse request against the globally synchronized IEEE 1588 clock and moves the DO activating through the different states; Idle → Prepare → Armed → Activated. While the digital output is Armed the pulse can be corrected, but once the DO is activated only the pulse width can be manipulated.

The simulated version of TSP has been implemented similar to the high precision times. When a new TSP is created in the application, an event is subscribed to the Idle Event Handler, instead of writing to registers in hardware. Whenever the application wish to optimize the pulse, the event will be updated in the Idle Event Handler.

The simulated version of the 1588 clock is connected to the system clock controlled by the idle thread. This means that the idle thread has full control of the timely aspects in the simulation, and the 1588 distributed clock synchronization is not relevant to the simulation because the distributed clocks will always be in sync. The TSP state machine does not need to constantly monitor the 1588 clock. We always know the exact time of either an TSP pulse optimization or idle event execution, an therefore also the time when the state machine needs to evaluate its state and make the transition required.

3.4 Simulation Orchestration Manager

For scheduling a temporal execution of multiple controllers a simulation orchestration manager has been developed in the main Linux process. On figure 3.4 the manager has been illustrated. First the controller shared libraries, called nodes, are loaded into the process and stated in individual threads, enabling multi-threaded execution. The idle callback functions are connected to the idle thread and a simulation main loop is started. The simulation loop will wait for all nodes to return to idle where they will execute the callback to the manager. Once all nodes are stopped the manager ask for the next event time of all nodes. The event time closest to the global time will be the next global event time. All nodes who has an event occurring at the next global event time will be executed in parallel. When all nodes are back in idle the next global event time will once again be calculated.

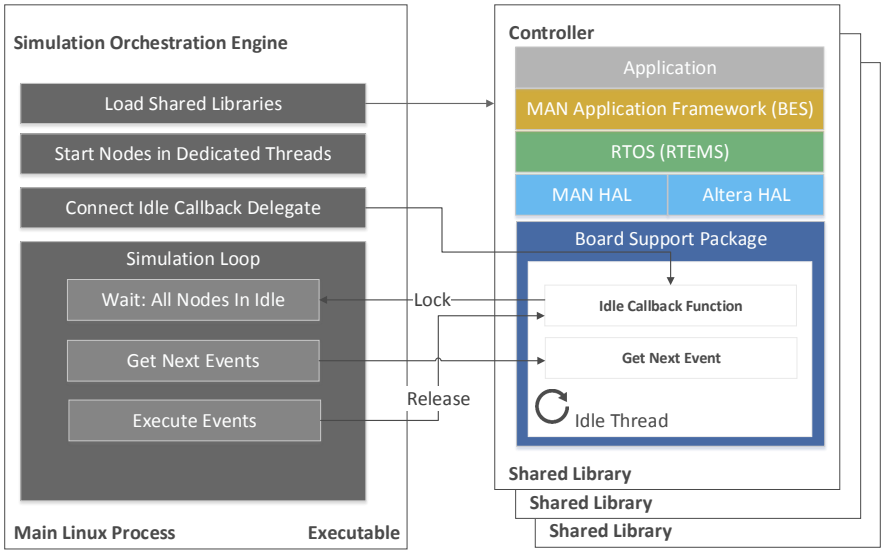

Figure 3.4: The simulation orchestration manager.

3.4.1 Network Simulation

When simulating a distributed system the communication between nodes is important. The main type of communication between nodes in the ECS is the Ethernet network. With the possibility of introducing interrupts and replacing hardware specific layers of the software, it is also possible to model the network. We only need to replace the hardware network driver with a simulated version. With this solution we maintain all the above layers of the network such as the Ethernet Interface and the network protocols, including the MD&T developed Control Net.

As illustrated on figure 3.5, the Ethernet interface is a OS task that contain the network driver base class from which the simulated network driver is derived. The driver inherits from the network receive class (Network RX). Network RX is running as an OS task and protected by a semaphore. The simulation driver uses the class to dispatch packages into the application through the "Process_Package" method when the OS is scheduled to receive packages. The network simulation driver offers the same interface as the hardware driver but instead of connecting to the physical network adapter we connect to a port class located in the simulation manager of the Linux process. The connection is created through a "Link_Callbacks()" method that has been exported from

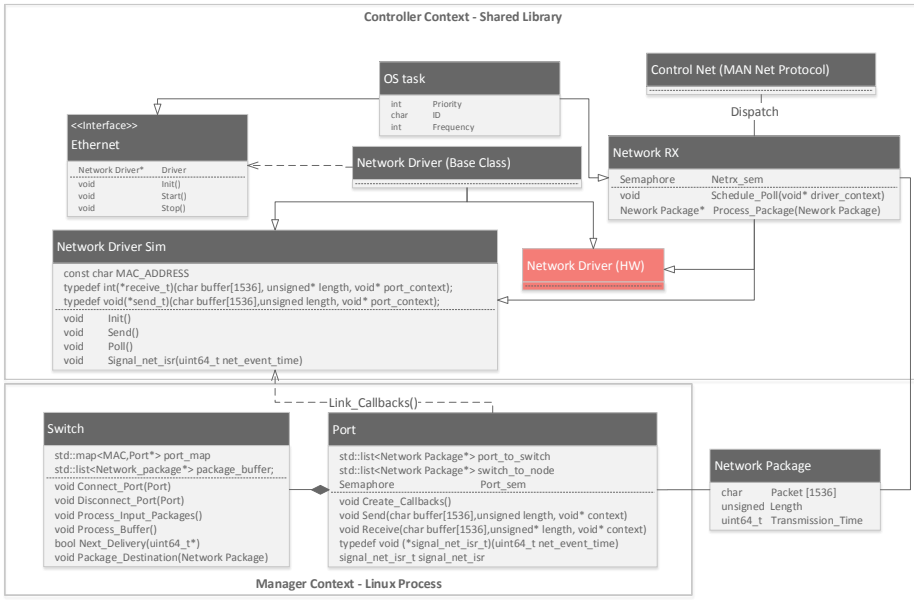

Figure 3.5: Class diagram of network driver.

the controller shared library. This method enable the "Send()" method of the driver to call the "Send()" method of the port, the "Poll()" method to call the "Receive()" method and the port function pointer "signal_net_isr" to call the "Signal_net_isr()" method of the driver. The Port class contains two lists of network packages; one for packages coming from the node to the switch and another from the switch going to the node. The lists are shared resources between the nodes and the switch and have to be protected by the port semaphore.

It is important to notice that the manager and controller are running in two different contexts. The manager is running in the Linux kernel with its own memory space and the controller is running in the RTEMS context within its own thread and predefined memory space. This essentially means that calling the "Send()" method of the port from the controller context will make the controller thread execute a Linux context method. The issue occur when the controller wants to push a package on the "port_to_switch" list, which is in the Linux memory space. We have solved this issue by passing the port object as a *void** though to the network driver. In the same way when the simulation manager tells the controller that network packages are ready to be received, we imitate a network interrupt signal routine by execution the "Signal_net_isr" callback. The callback interrupt is not allowed to execute, as it comes from a Linux context, instead it schedules a high priority OS task to receive network

packages through the Network RX "Schedule_Poll". While doing this we need to disable the OS thread dispatcher to make sure no underlying threads are allowed to execute. The "Signal_net_isr" also subscribe an event to the idle event handler of the controller, in this way the manager can release the idle callback which will execute the newly scheduled network rx task and the driver "Poll()" method will start receiving packages from the port and processing them through the Network RX "Process_Package".

To distribute packages between nodes a virtual switch has been implemented in the Switch class. During the simulation initialization, the ports of all nodes are connected to the switch and added to a map with their MAC address as key. From the simulation orchestration manager the "Process_Input_Packages()" method can be executed which moves the network packages from the Port lists "port_to_switch" to the "package_buffer" list of the switch. The manager can evaluate the transmission time of the network packages in the buffer list and schedule the moving of packages to the port list "switch_to_node" through the "Process_buffer" method, according to their destination MAC address from the Ethernet packet frame. This is followed by signaling the node to schedule receiving of packages through "signal_net_isr".

3.4.2 SystemC Network Simulation Library

In the early stage of this research we focused on an independent model of the network topology connecting nodes through the FMI co-simulation interface, as will be described in further detail in chapter 4. This solution was disregarded in favor of the network driver simulation described above.

The network model utilized the SystemC Network Simulation Library (SCNSL Fummi et al. (2008)) to model the network. The SystemC Network Simulation Library is an extension to SystemC Design Automation (2012) that makes it possible to jointly design HW, SW and network in a single simulation tool. Using the SCNSL simulation kernel, testers and developers would be able to define simulation scenarios where protocols can be replaced and the connection between nodes can be configured to alter propagation speed, delay and priority. SCNSL handles the scheduling of the tasks and tracing of relevant reference and output signals. The aim was to do validation and fault injection to test the limits of the engine control system network.

The library provides 5 key elements, as described in Paper et al. (2015) and chapter 8:

- **Kernel:** The kernel is responsible for the execution of events in the correct temporal order and behaviour of the communication channels with features like propagation delay, byte-rate etc.. Since SCNSL is an extension to SystemC it exploits the SystemC scheduler by mapping network events on standard SystemC events.
- **Node:** A node is the active element of the network, it produces, transforms and consumes transmitted data. The node implementation is decoupled from the network simulation which makes it possible for system designers to e.g. change abstraction level, do fault injection, synthesis and validation.
- **Packet:** In packet-switched networks the packet is the unit of data exchanged among nodes. In general the packet format is highly dependent on the corresponding protocol. SCNSL does not provide a set of protocols and packet formats, but uses an internal format and lets the designer implement the protocol design on the specific node.
- **Channel:** A Channel represents the physical medium which connects two or more nodes. It can be either a point-to-point link or a shared medium. Multiple channel types are supported; Unidirectional, Half/Full-Duplex and shared.
- **Port:** Every node uses ports to send and receive packets.

A model of the engine control system network was created as illustrated on Fig. 3.6. All nodes are connected, by Ethernet cables, to a switch located in either the engine control room or engine room. Cables are represented by channels in SCNSL, where bit-rate and delays can be adjusted between ports. Each controller is implemented as a SCNSL-node-instance and consists of an input and output controller. In this way it is possible for developers to implement different protocols etc. and investigate the effects on the control system. Network packages are delivered to the node through the simulation kernel, activating the input controller thread upon arrival. Switches are implemented in the same manner as node-instances. Standard switches simply forward communication, but through simple customization they can become a key element for investigating network improprieties. The prioritization of the network packages is done using the SCNSL-embedded channels that makes it possible to prioritize packages on kernel level before arriving to the switch.

By implementing the FMI standard in the SystemC simulation kernel it was possible to distribute network packages between controllers implemented as independent sub-systems in the co-simulation. The solution was disregarded mainly because of improprieties with the FMI standard and the workload of implementing already developed network protocols ect. in SCNSL. As will also be

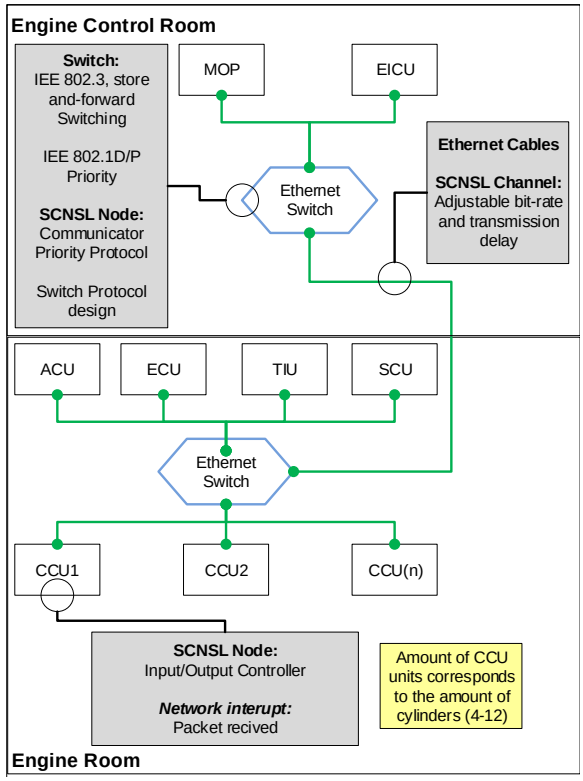

Figure 3.6: Simplified Network model of the engine control system on a MD&T engine.

described in chapter 4, the FMI standard limits the amount of data-types that can be used. Transferring network packages through FMI was done as strings (char*) with predefined size. It is not possible to stack packages in e.g. an array or struct, and each package has to be received and sent through individual $Get()$ and $Set()$ calls, which is not very efficient. The efficiency of the FMI standard is in general very sensitive to the amount of inputs and outputs. The main reason for not moving on with the model however, was the workload required by developers to translate already implemented functionality to SCNSL as compared to the network driver simulation in section 3.4.1. The SCNSL Model developed is not completely disregarded and could be used as mock-up for future investigations, if the ECS network has to undergo significant change. Results from the SCNSL and FMI co-simulation are presented in chapter 8 and Paper et al. (2015).

3.5 Chapter Summary

This chapter aimed to fulfill the first objective from section 1.1; To simulate the distributed embedded control systems with multiple controllers and their communication in a single coherent simulation. We have presented how the application software is cross-compiled to either target or PC making it possible to execute the controller application on a developers PC. Cross-compilation was part of the development process before this research. We showed how the idle thread in the BSP layer of the embedded software architecture can be manipulated to control the execution of the controller instance. By introducing a blocking callback function to a simulation orchestration engine it is possible to control execution in deterministic manner. To ensure a deterministic and temporally correct simulation, an event handler was introduced in the idle thread. All events, such as OS progression and interrupts, that normally would be scheduled by hardware/software timer interrupts, can be subscribed to the event handler and scheduled by the simulation orchestration engine. To get a concept of time and ensure temporal integrity, a simulated system time is introduced as local controller time. The system time is progressed by the event handler, that know exactly at what time each event occur. Making the assumption that all tasks can always finish execution and not be interrupted, we get a representation of time that is deterministic and can be used to execute multiple controllers in a temporally correct manner. A considerable part of a distributed system is communication. In the engine control system the communication is Ethernet based. With the ability of replacing hardware specific functionality and introduce interrupt events in the idle event handler, it is possible to replace the controller network driver. In the simulation orchestration engine a virtual switch was created, distributing network packages between ports connected to the simulated version of the network driver. With a delegate structure, the manager can move packages between the controller and Linux context and signal a network interrupts on the controller which processes packages thought the controller software using original protocols ect. As a result of the solutions presented in this chapter, it is possible to simulate the complete distributed embedded control system and its communication in a deterministic simulation.

Chapter 4

Embedded System Co-Simulation

Designing embedded systems for the control of a physical plant, the required fidelity of the plant model is highly dependent on the complexity of the control algorithms to be designed. Fundamental differences in the development process and tool-chain of embedded control system development and physical dynamic modelling makes it difficult to design both, in a single modeling and simulation environment. At MD&T increased system complexity, require the control system development to use higher fidelity models than previously, to achieve a well performing design. The development philosophy at MD&T has always been, that the right tool should be used for the given task. This makes the amount of tools available for engineers very diverse. Rather than forcing engineers to use a single multi-domain tool, able to simulate the complete system, a co-simulation approach has been investigated.

4.1 Concept of Co-Simulation

In co-simulation, sub-systems of a coupled problem are modelled and simulated in a distributed manner. The goal of co-simulation is to verify as much of the system functionality as possible before committing to a design. The modelling

of each sub-system is performed in domain specific tools and by domain experts without the coupled problem in mind. The coupled simulation exchange data between sub-systems and manage simulation progression without knowledge of the sub-system dynamics. Co-simulation sub-systems are typically weak-coupled with a discrete communication patterns either parallel or serial.

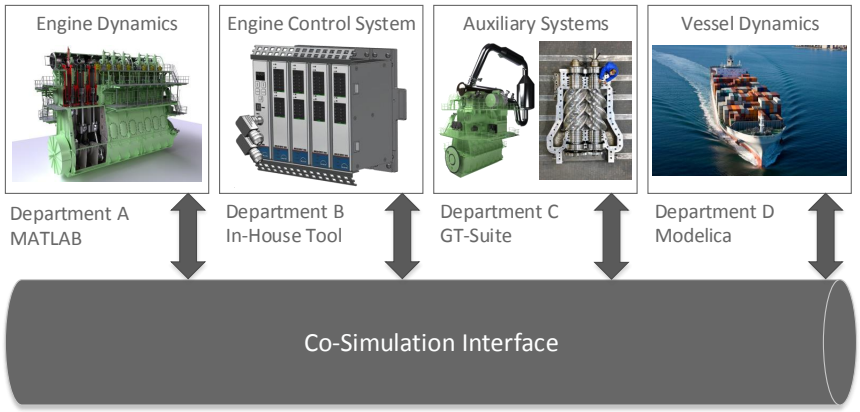

Figure 4.1: Co-Simulation Concept.

Developing engines for marine vessel population include multiple domains. As seen on figure 4.1, both engine dynamics and auxiliary systems such as emission reduction systems, turbo-chargers etc. and sometimes even vessel dynamics has to be taken into account when developing control systems. At MD&T the different domains are developed in separate departments with different tools. The aim of introducing co-simulation is to enable cross department collaboration and to make it possible to share models reducing redundant modelling efforts.

4.1.1 Co-Simulation Approach

As discussed in the survey section 1.2 FMI has been chosen as co-simulation standard. The main advantage of the FMI standard is its diversity and flexibility. FMI is based on C-code, making it platform independent and require nothing more then a C-compiler. Subsystem information is described in a simple manner with an XML model description, making the interconnection configuration between subsystems easy to manage. FMI provides an application interface, with a state machine, that needs to be implemented. This state machine ensure that each subsystem is simulated in a similar fashion and that execution and

communication within the system is temporally correct. Even though a full application interface needs to be implemented, it is only the function calls that are required by the standard, how they are implemented is completely free. Implementing the standard to a custom simulation requires some effort. This process is, however, well documented and exemplified in the FMU SDK from QTronic and the FMI++ library Widl et al. (2013), making the task manageable.

The high level of flexibility come with some constrains. Especially, the way data is exchanged between subsystems is limited. Data types are restricted to "Real","Integer","Boolean", "char", "Byte" and "String". It is not possible to exchange e.g. arrays or any advanced data-objects, which can be inconvenient. In our embedded system software most data-types are fix-point types and similar. This requires a conversion layer between FMI data-types and internal data-types. Furthermore, the data-exchange is based on a Get/Set functionality, when the system and connection amount grows this become a significant execution overhead.

Conclusively, the FMI standard provides the flexibility needed to co-simulate our distributed control system and opens up for interconnection with a vast amount of tools already in our organization. The following section will describe the FMI standard and how it has been implemented.

4.2 The Functional Mock-up Interface

The Functional Mock-up Interface (FMI) is a tool-independent standardized interface for coupling of sub-models. Initiated by Daimler AG under the MODELISAR ITEA 2 ITEA Office Association (2015) European project, the FMI development aimed to improve the exchange of simulation models between suppliers and OEMs. The first version of FMI was published in 2010, followed by FMI 2.0 in 2014. FMI 2.0 supports two ways of using the standard, FMI for model exchange and FMI for co-simulation. In FMI for model exchange sub-models are generated into Functional Mock-up Units (FMU) to be utilized by another modeling and simulation environments. The holistic simulation is then performed in a single tool and solver. FMI for co-simulation is used to couple simulation environments instead of models. This means that both models and solvers are compiled to the FMUs. Each sub-system is solved independently with their own unique solver. Data-exchange between sub-models and synchronization of all sub-system solvers is restricted to discrete communication points and governed by a master algorithm. In this dissertation we will only discuss the FMI for co-simulation standard.

Functional Mock-up Units are the sub-system components of the FMI standard. A FMU is an archive file consisting of a model description XML file (modelDescription.xml) describing how the sub-system is connected to the rest of the simulation, a compiled C dynamic link library of the FMI application interface is wrapped around the actual sub-simulation to be executed. Besides from the model description and the C library the FMU can also contain documentation and any other resources, like libraries or configuration files, if needed by the simulation.

4.2.1 Model Description

The model description is a XML schema that contains all the information about the co-simulation setup, required to connect to the sub-system (slave). Fig. 4.2 show the top-level of the schema, it should be noticed that not all components of the schema are required (Illustrated on figure 4.2 where boxes and connections with – are required and - - optional). This dissertation will only discuss the required configurations, for further information please refer to FMI documentation Blochwitz et al. (2012).

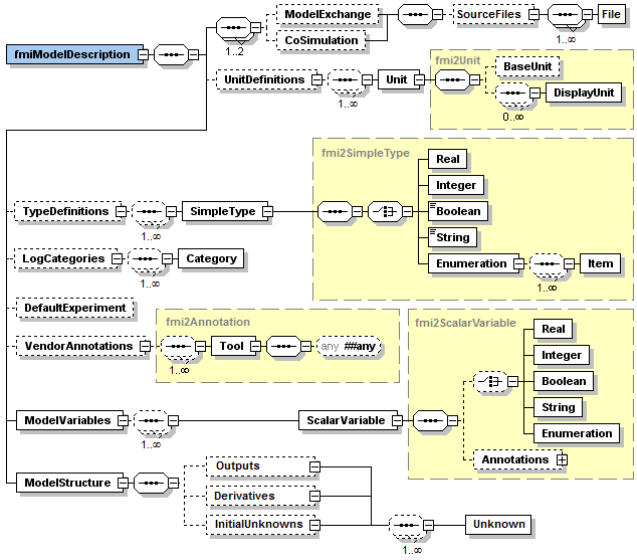

Figure 4.2: Model Description Schema.

All FMUs have to specify their FMI version, Model name and a Globally Unique IDentifier (GUID). The essential part of the schema is the ModelVariables, here all the variables exposed by the FMU to the simulation manager

(master) are defined. The ModelStructure is necessary for handling the order of outputs exposed, possible derivatives and states for linearization, together with the unknowns during initialization. An important part of the schema for co-simulation is the fmiModelDescription→CoSimulation→Attributes see Blochwitz et al. (2012). Here the capability flags, that define the capabilities of the co-simulation subsystem, are defined. The capabilities available are:

fmiModelDescription-CoSimulation-attributes:

- **modelIndentifier**
  Unique name for the manager to distinguish between FMUs.

- **needsExecutionTool**
  If the FMU need a tool to execute e.g. if the tool needs to access the MATLAB MEX compiler to execute.

- **canHandleVariableCommunicationStepSize**
  If the slave solver can handle variable step size.

- **canInterpolateInputs**
  If the slave is able to interpolate continuous inputs.

- **maxOutputDerivativeOrder**
  Maximum order of output derivatives.

- **canRunAsynchronuously**
  If the slave allows for asynchronously call of the step function.

- **canBeInstantiatedOnlyOncePerProcess**
  If the FMU can only exist in a single instance per process.

- **canNotUseMemoryManagementFunctions**
  If the slave does not allow memory management functions.

- **canGetAndSetFMUstate**
  If the manager can change the state of the slave.

- **canSerializeFMUstate**
  If the FMU can be serialized.

- **providesDirectionalDerivative**
  If the slave directional derivative can be calculated at communication point.

These capabilities are used by the FMI manager to orchestrate the holistic simulation between multiple FMUs.

4.2.2 Application Interface

FMI provides a standardized application interface that defines how to execute a FMU from a simulation manager. The purpose of the interface is to have both textual and binary representation of a FMU that is similar across several sub-systems. The interface is delivered as 3 header files:

fmi2TypesPlatform.h
Platform specific type definitions for all input and output arguments of the functions. Instead of C types, alias types are defined to simplify porting. Input and output types are limited to $Real[double]$, $Integer[int]$, $Boolean[int]$, $Char[char]$, $String[const\ char*]$ and $Byte[char]$.

fmi2FunctionTypes.h
Type definition of data types and all function prototypes.

fmi2Functions.h
The function prototypes of the API to be implemented in and exported from a dynamic link library, and accessed in the FMI simulation manager. The most important functions to be implemented are illustrated on figure 4.3 and will be described further below.

The FMI standard is contains a state machine illustration, as seen on 4.3, that describe the calling sequence from the simulation manager (master) and the sub-system FMU (slave).

Instantiated
The simulation manger calls the $fmi2Instantiate$ function to bring the FMU in the $Instantiated$ state. Here all initial and approximated variables are determined and set with the $fmi2Set$ functions. Simulation configuration parameter are set by the $fmi2SetupExperiment$.

InitializationMode
Inputs and initially unknown variables are calculated and set by the $fmi2Get$ and $fmi2Set$ functions.

SlaveInitialized
This state is initiated when the slave has been initialized. It is here that the actual step calculations are performed. All inputs and outputs are updated when the step function $fmi2DoStep$ is executed. Depending on the outcome of the step progress, the following state is either "stepComplete","stepFailed" or "stepCanceled" which are self-explanatory.

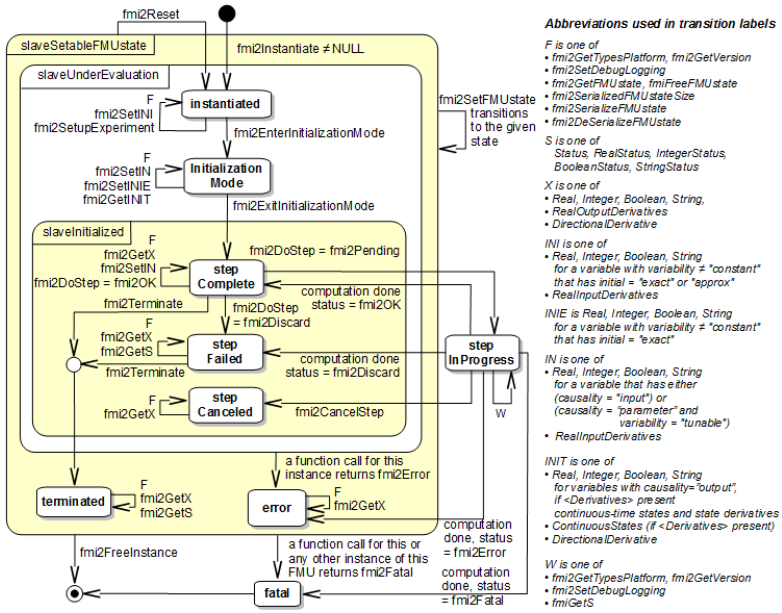

Figure 4.3: State Machine Co-Simulation.

terminated
: The simulation is terminated and the final results of the simulation can be retrieved

The application-interface is compiled as a dynamic link library either a linux shared library or microsoft DLL. The binary or multiple binaries for cross compatibility are placed in the binaries directory of the FMU archive file.

4.3 FMI for Embedded System Software

In order to co-simulate our engine control system together with any of our physical dynamic tools we need to implement the FMI standard into the SIL simulation of the ECS described in chapter 3.

4.3.1 Engine Control System FMI Implementation

The FMI application interface introduce a set of function prototypes to be implemented and exported through a dynamic link library. In this section we describe how the interface has been implemented within the engine control system simulation. The actual implementation of each function will not be described but an overview of how the implementation connects to the ECS simulation will be provided.

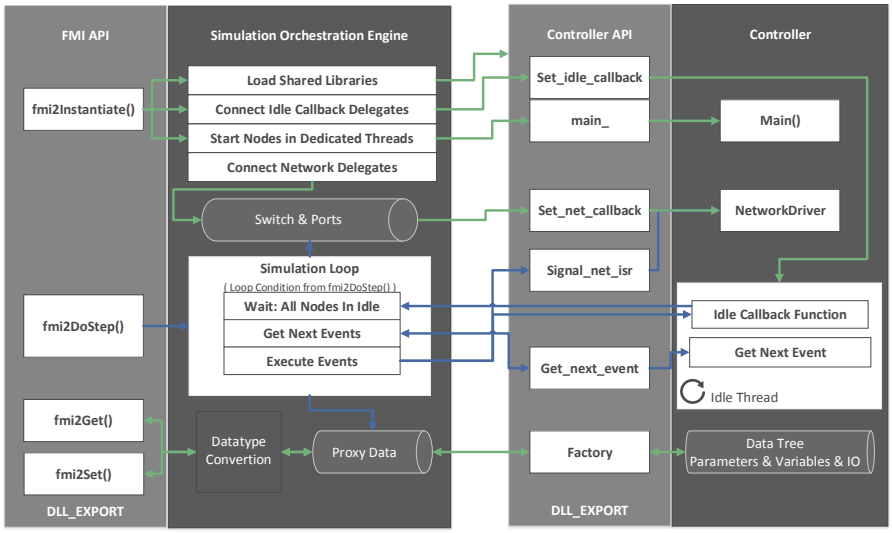

Figure 4.4: FMI Implementation.

As seen on figure 4.4, the Simulation Orchestration Engine (SOE) from section 3.4 has been compiled to a shared library that exports the FMI API. This library is the main library in the FMU. The $fmi2Instatiate()$ function is the one loading the shared libraries of all the controllers in the ECS simulation. Each controller is started in a dedicated thread by calling their main function through the Controller API exported from the controller shared library. Idle callbacks and network ports are connected as described in chapter 3. The $fmi2DoStep()$ is the function that progress the simulation and is called between every discrete communication point. From the FMI manager algorithm, the current time and communication time step will be passed on to the SOE simulation loop. The simulation loop will simply advance with the communication time step and perform the events on all controllers that need to be executed, by accessing their idle event handler and updating the network communication. Between every step in the FMI simulation, communication between sub-systems should be

performed. The FMI standard introduce a Get/Set functionality, where a *Get*() and *Set*() method has to be implemented for each FMI datatype (Real, Integer, Boolean, Char, String, Byte). Like most embedded systems the controller data-types are implicitly defined to make sure they fit the hardware. To translate between FMI data-types and controller data-types a conversion layer has been implemented. To map the FMI variables to ECS data-tree parameter, IO or variables, we have put an extra constraint on the model description by requiring the name of a scalar-variable to correspond with the unique ID of a data-object in the data-tree of controllers. As can be seen on the model description xml in listing 4.1, the *ScalarVariables* in the *ModelVaribles* tag, has a *name* attribute corresponding to a data-object ID in the ECS which maps to the valueReference attribute used by FMI. When the FMI API calls the *Get*() or *Set*() method with a valueReference, we can use the modelDescription.xml to get the ECS ID. With the ECS ID it is possible to ask the controller what data-type the specific ID has. Using a template implementation, a standard C map can be created mapping between FMI an controller data-types e.g. controller types; Fixpoint-16, float or double can be mapped to a FMI datatype Real. It is also the ECS ID which is used to create the proxy instance of a data-object in the SOE through the Factory of the Controller API.

```
<?xml version="1.0" encoding="utf-8"?>
<fmiModelDescription
  fmiVersion="2.0"
  modelName="ECS"
  guid="{xxxx}">

<CoSimulation
  modelIdentifier="ECS"
  canHandleVariableCommunicationStepSize="true"/>

<ModelVariables>
  <ScalarVariable name="ECS_ID" valueReference="0">
    <Real start="1"/>
  </ScalarVariable>
</ModelVariables>
</fmiModelDescription>
```

Listing 4.1: ModelDescription.xml

4.4 Co-simulation of SCR Heating Model and ECS

With the co-simulation setup, it is now possible to express physical dynamics in a tool for continuous time simulation and co-simulate it with the ECS simulation. In paper Pedersen et al. (2016) and chapter 9, a simple reactor heating model of the Selective Catalyst Reduction (SCR) system was created in Ptolemy II

Liu et al. (2001); Brooks et al. (2010) and co-simulated with the Engine Control System simulation as presented below.

4.4.1 SCR Heating Model

To comply with emission regulations, set forth by the IMO (Organization., 2013), for reduction of NO_x, a SCR emission-reduction system can be installed on the vessel. The SCR system will redirect the exhaust gas from the exhaust manifold through a vaporiser and mixing chamber, where a reducing agent (ammonia/urea) is injected into the gas. This gas mixture is then passed through the SCR Reactor of catalyst chambers where the selective catalytic reduction process will be performed to significantly reduce the amount of NO_x, as seen on figure 4.5.

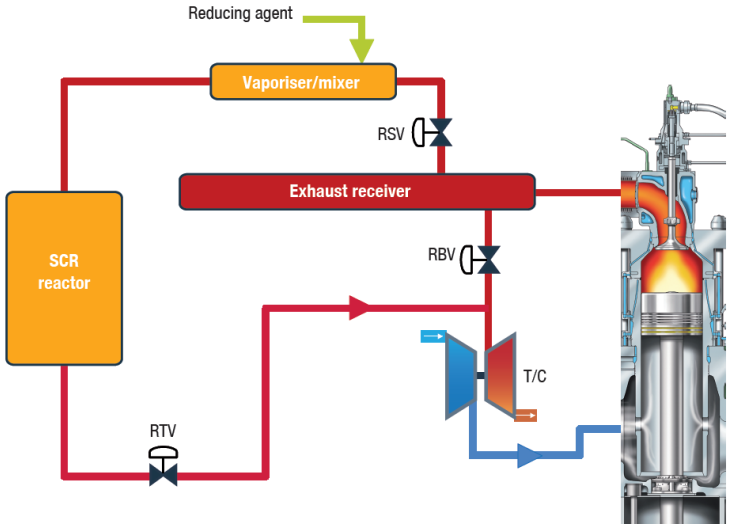

Figure 4.5: Air-path model of the Selective Catalytic Reduction.

The control system has to balance the air flow between the SCR system for emission reduction and the turbine-inlet of the turbo-charger for providing sufficient scavenge-air pressure for the scavenge and combustion processes. To regulate this balance the control system has three valves for the distribution of exhaust gas. The Reactor Sealing valve (RSV) connects the exhaust receiver manifold with the vaporiser and mixing chamber and the reactor. The Reactor Bypass Valve (RBV) can redirect exhaust gas directly to the turbine-inlet. For the SCR process to work properly the reactor has to reach an optimal temperature range

between 350 °C and 450 °C. To heat up the reactor, the Reactor Throttle Valve (RTV) can reduce or increase the amount of gas leaving the reactor and thereby control the heating of the reactor and balance the pressure to the turbine-inlet.

The control algorithm for the RTV valve, uses the error calculation between the reactor input and output temperature as a reference-residual signal for controlling the valve position. In this simplified model, the heating of the reactor is modelled as a time delay and the resulting SCR output temperature passed back to the SCR Controller. We will show that it is possible to co-simulate and investigate the dynamic interaction between a physical dynamic model and the actual control software.

The output temperature can be modelled as a 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 control system simulation.

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

$$\dot{M}_n = \dot{M}_{n-1} + \frac{L - \dot{M}_{n-1}}{1 + \tau_{Scavenge} \cdot T} \tag{4.1}$$

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.

$$\tau_{out} = (\dot{M}_n \cdot RTV + \tau_{reactor}) \cdot 3600 \tag{4.2}$$

Finally the output temperature is calculated as:

$$Tscr_{out_n} = Tout_{n-1} + \frac{Tscr_{in} + Tout_{n-1}}{1 + \tau_{out} \cdot T} \tag{4.3}$$

This is of course a simplified approach, however it proves to show, that it is possible co-simulate the actual control system software together with a thermodynamic model executing in a different tool.

4.4.2 Co-Simulation Configuration

On figure 4.6 the configuration of the co-simulation is illustrated. The open-source simulation framework Ptolemy II was chosen due to its heterogeneous

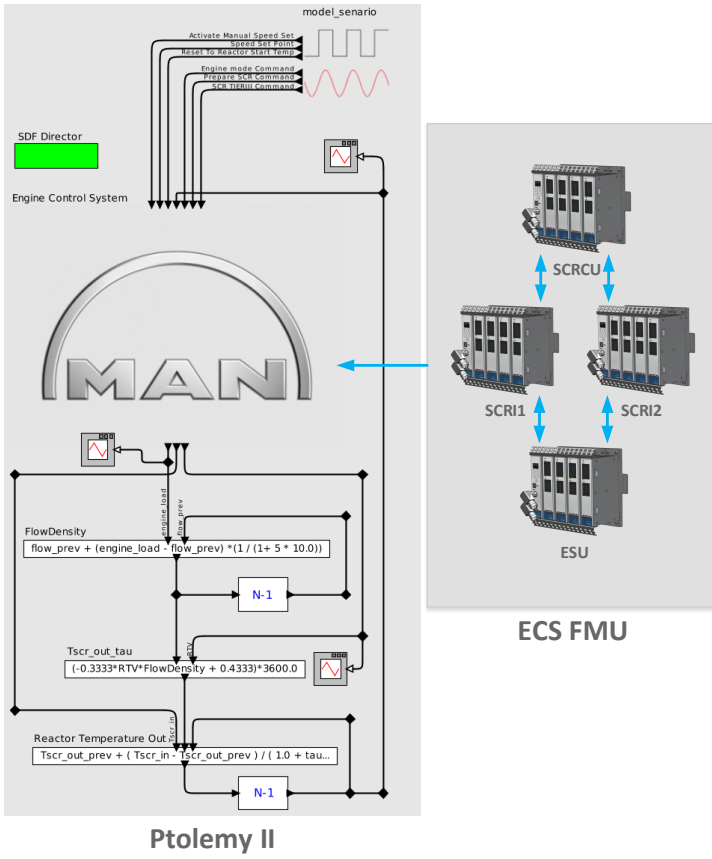

Figure 4.6: Configuration of Ptolemy II and ECS co-simulation.

actor-oriented design and comprehensive support for different software components including a FMI-interface as described in Broman et al. (2013). The ECS FMU described in chapter 4.3 was imported as a co-simulation actor and automatically configured by Ptolemy using its model description. The graphical user interface of Ptolemy, "*Vergil*", was used to implement the equations from above and connected to the FMU inputs and outputs. A simulation scenario, manipulating the control system, is likewise described in Vergil and connected to the input ports of the FMU. The scenario sets the internal reactor start temperature, engine speed set point and imitates operators interaction with the control system through the HMI EC-MOP. After 700 seconds a simulated EC-MOP command is sent to the SCR controller to activate the SCR control strategy.

A Synchronous dataflow (SDF) director was chosen to execute the simulation. The SDF director is appropriate since we have a predictable and regular execution of the FMU. At regular communications points inputs/outputs are updated in a predefined order.

In the ECS FMU four controllers have been included. The SCR control strategy is implemented in the SCR Control Unit (SCRCU) and the SCR Interface Units (SCRI1 & SCRI2) connects to the engine through I/O's. To have a full simulation of the SCR system, more models than the reactor heating model is required. As described in 2.3 an Engine Simulation Unit (ESU) containing multiple engine models is available for doing SIL and HIL testing. The ESU has been used to provide the missing models required for doing a complete SCR simulation. Ptolemy connects to the SCRCU, where it recived the engine load and manipulate the speed set-point, reactor start temperature and engine commands. The input temperature is given to the Ptolemy model from the SCRI1 and the RTV valve opening from SCRI2.

4.4.3 Co-Simulation Results

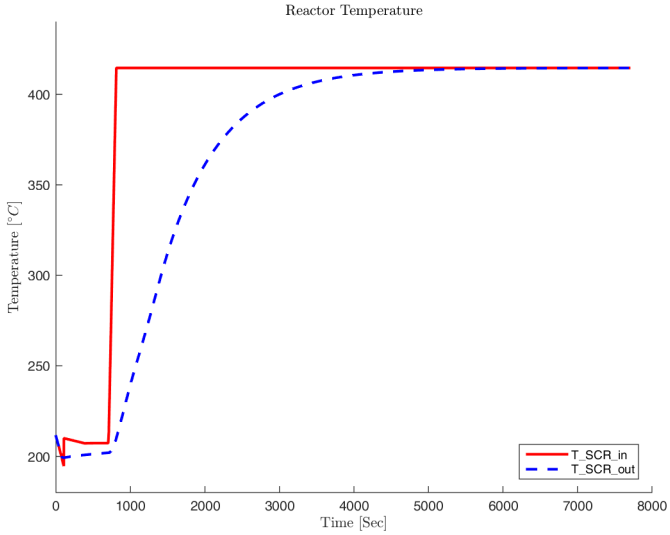

Figure 4.7: The in- and output temperature of the simulated SCR heating.

The result of the SCR reactor heating is presented on figure 4.7. Here we see how the SCR start command is sent at 700 seconds and the SCR reactor out temperature starts to increase. The simulation has the expected low-pass

behaviour and takes approximately 1.5 hours to heat up, which is expected.

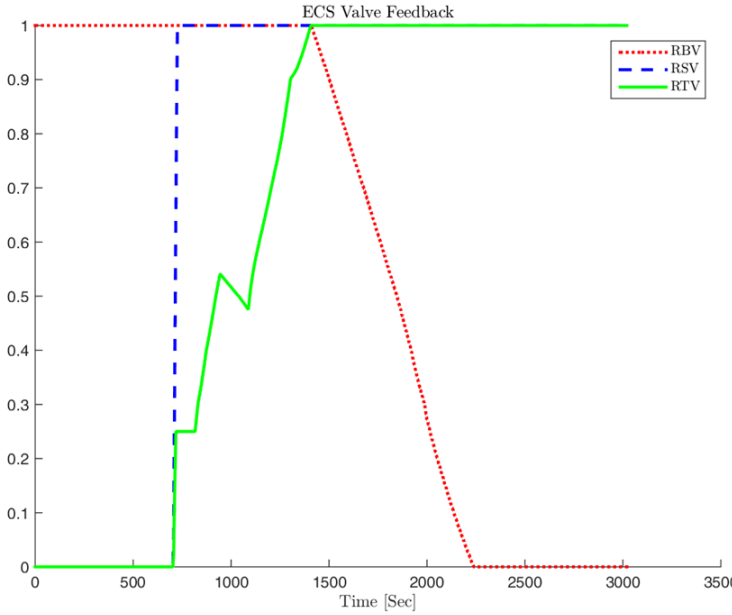

Figure 4.8: Valve feedback from the SCR simulation.

The valve control signals from the ECS simulation is plotted on figure 4.8[1]. When the SCR activation occurs, the RTV and RSV valves start to open. The RTV valve is clearly trying to heat the reactor by building up pressure while balancing the flow to the turbocharger. This actuation is filtered from the temperature output 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.

We clearly see that the co-simulation is working as intended and it is possible to implement part of the physical dynamic modelling in a tool like Ptolemy. With this setup it is possible to implement higher fidelity models than those developed for HIL in the ESU, in a tool appropriate for physical dynamic modelling.

[1] In the published paper Pedersen et al. (2016) the legends on Figure 10 are mixed up. The results presented on figure 4.8 are correct.

4.5 Chapter Summery

This chapter discussed the reasons for choosing the Functional Mockup Interface as co-simulation standard. It was described how the standard was implemented in the engine control system simulation. The FMI application interface was wrapped around the Simulation Orchestration Engine (SOE) and compiled as a shared library with the FMI methods exported. The FMI step function provides the loop conditions for the ECS main simulation loop, and controllers are loaded as shared libraries within the SOE library. Inputs and outputs to the sub-system are accessed through FMI Get() and Set/(methods in the SOE library and mapped to proxy instances of controller variables. A conversion layer has been implemented converting internal controller data-types to FMI data-types and vice versa. An example using a SCR reactor heating model expressed in the tool Ptolemy was co-simulated together with the ECS FMU. Results showed that the co-simulation worked as intended and proved that it is possible to have the control software simulated together with a physical dynamics model expressed in a distributed simulator. As stated in the objectives from section 1.1, the co-simulation environment will enable engineers to better share models and do more holistic system investigation and validation, utilizing a wider range of tools. It is expected that the co-simulation approach will make it possible to validate new systems and features at an earlier stage and reduce the time spent on the expensive test-benches.

Chapter 5

Distributed Co-Simulation

Distributed simulation is the practise of executing a single simulation on a collection of loosely coupled processors, e.g. multiple PCs interconnected by LAN or WLAN. As compared to parallel simulation where the simulation is tightly coupled e.g. like the ECS simulation from chapter 3, where each controller is given an individual thread so the Linux kernel can distribute the execution of multiple controllers to multiple processors. Distributed simulation is often used for analysis of complex systems such as transportation networks, next generation internet, military and professional training etc. Dealing with large systems that are time consuming to simulate, distributed simulation can reduce execution time and make it possible to run large simulations utilizing more memory. Distributed simulation are mostly connected by network interfaces that are both platform and architecture independent, meaning that processes that are otherwise incompatible can be simulated together, e.g. a Linux 32-bit process and a Windows 64-bit.

5.1 Platform and Architecture Challenges

A MD&T control system development is performed in the SIL environment using the Dynamic Simulation Environment (DSE) for developing physical dynamic models to test new control algorithms. The DSE and SIL environment can be

compiled to the 32-bit target or cross compiled to a Linux 32-bit object to be executed on the developer Linux PC. The DSE was developed with the main purpose of enabling physical dynamic models to be executed in the SIL and HIL environment. While DSE is powerful and important for the HIL testing it is by design limited to low fidelity modelling, since models have to be able to run real-time on the target hardware. With the increasing system complexity, it is desirable to connect modern modelling tools for performing co-simulation with the ECS and high fidelity physical dynamics models. In large organizations, department tool-chain and development environments will often deviate. At MD&T software and control system development is done in Linux where as detailed engine performance modelling is done on Windows in tools such as GT-Suite and MATLAB/Simulink along side in-house developed tools. This deviation in platform is a challenge for the newly developed co-simulation environment. As previously discussed, it is not desirable to force a single tool-chain upon all development, but rather have a co-simulation approach. To be able to co-simulate a high-fidelity model, developed by performance departments in e.g. Window 64-bit MATLAB/Simulink, together with the ECS SIL simulation in 32-bit Linux, we need to be able to do a distributed co-simulation where sub-systems can be distributed to the correct platform.

5.2 INTO-CPS

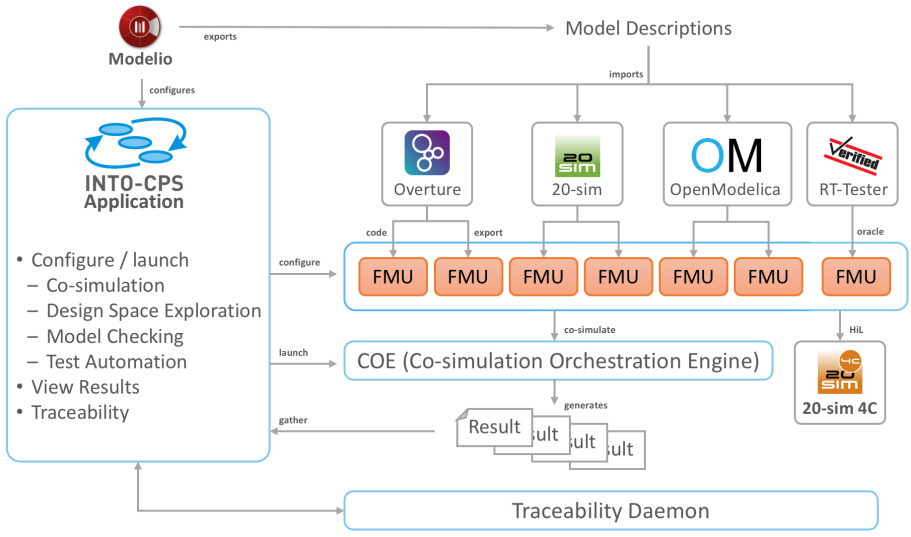

Figure 5.1: INTO-CPS Tool Suite.

The Horizon 2020 project INTO-CPS ("Integrated Tool-chain for the model-based design of Cyber-Physical Systems") Fitzgerald et al. (2015,0) aim to create an integrated tool chain for model based design of Cyber-Physical Systems Larsen et al. (2016). The tool-chain is illustrated on figure 5.1, supporting everything from requirements to final realization through multidisciplinary and collaborative modelling of CPS Bandur et al. (2016). At the core of the tool-chain is the Co-Simulation Orchestration Engine (COE), which is a fully FMI 2.0 co-simulation compliant manager supporting both fixed and variable step size simulations. With the INTO-CPS COE being a FMI compliant manager it is possible to use COE for co-simulation of the ECS simulation FMU.

5.2.1 Co-Simulation Orchestration Engine

The COE is developed in a combination of Java and Scala, which makes it multi-platform and provides the simulation service through HTTP. Currently, two methods for time-stepping are implemented; one for fixed time steps, and one for variable time steps. The COE is capable of switching on stability checking as well as using parallelism Thule and Larsen (2016). In addition to the baseline tools incorporated inside the tool-chain, a number of other modelling and simulation tools have been tested with the COE. This includes both commercial tools such as Dymola, Modelon, SimulationX and Unity as well as additional open source tools such as 4Diac. While the COE is multi-platform, it does not directly support mixed-architecture (combinations of 32bit and 64bit architectures) or mixed-platform (combinations of e.g. Windows and Linux) simulations as required at MD&T.

5.3 Distributed Co-Simulation Orchestration Engine

In collaboration with the INTO-CPS team, a project was initiated to enable their COE to perform a multi-architecture and platform co-simulation, able to mix 32bit and 64bit code on both Windows and Linux. To realize such a distributed simulation, two processes with inter-process communication is required by the host system, where one of them acts as the simulation master.

An extension to the COE was developed which is capable of both simulating across architectures and platforms. A solution, where an extension point in the COE allowed a custom factory to be used for FMU instantiation, was chosen. An overview of the extension was realized as shown on figure 5.2. The COE

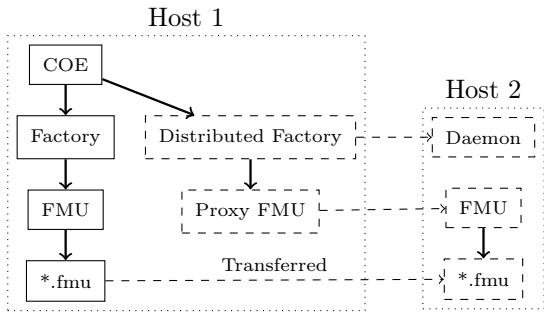

Figure 5.2: Distributed Extension Overview.

uses the distributed factory to instantiate FMUs that require execution with a different host configuration, either architecture or platform deviation.

5.4 Co-Simulation of Engine Control System and Physical Dynamic Tools

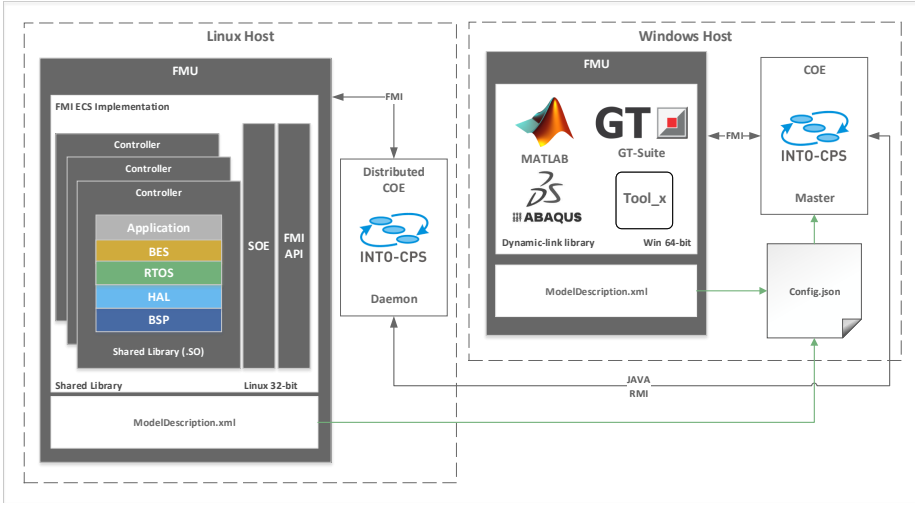

Figure 5.3: Distributed Co-Simulation using INTO-CPS.

Using INTO-CPS with the extended Distributed Co-simulation Orchestration Engine (DCOE), it is possible to co-simulate the ECS simulation together with the Windows tools available at MD&T compliant with the FMI standard. As

seen on figure 5.3, the distributed COE runs a deamon service on the Linux host that can connect to the master COE on the Windows host. The JSON configuration file on the Windows host describes the distribution of FMU across all available host machines and using Java RMI Oracle Corporation (2004) the sub-system FMUs and communication is distributed.

5.5 Distributed Co-Simulation of EGR Water Handling System

At MD&T a new cleaning system for NO_x emission reduction, the Exhaust Gas Recirculation (EGR) Water Handling System (WHS), was developed using the traditional development approach. During final system validation on the engine test bench it was discovered that the new control strategy did not work properly. The control strategy was developed in the SIL environment with physical dynamics expressed in the DSE extension and tested on the HIL test bench with the DSE model running in the Engine Simulation Unit (ESU). During the preliminary investigations a MATLAB model of the new systems was developed. When moving to the detailed design this model was translated to the SIL environment as a DSE model. During this translation, some abstractions were made to ensure that the model could execute real time for HIL testing. In this section we will present the results from the publication Pedersen et al. (2017b) also presented in chapter 11, where it was investigated if the MATLAB model, developed in 64-bit Windows, could be co-simulated together with the 32-bit Linux ECS simulation. The hypothesis was that the higher fidelity model developed in MATLAB, could have foreseen the defects of the control strategy that occurred during the expensive engine test bench validation.

5.5.1 EGR Water Handling System

The EGR system works by redirecting the exhaust gas from the exhaust manifold to the scavenge-air intake manifold, thereby lowering the in-cylinder oxygen (O_2) level due to the high concentration of carbon dioxide (CO_2) in the exhaust gas. The exchange of O_2 with CO_2 leads to a decrease of combustion speed, resulting in lower peak temperatures during combustion. Furthermore, it also results in a higher in-cylinder heat capacity of the gas which also lowers the combustion temperature. Lower combustion temperatures and especially lower peak temperatures result in lower formation of thermal NO_x during the combustion process. The recirculated exhaust gas is very polluted compared to the residual ambient scavenge-air. The gas has to be cleaned to prevent Sulphur (SO_2) and

other particles from damaging the engine. The gas is cleaned by spraying it with water and by cooling the gas to collect the polluting practicals. A Water Handling System (WHS) provides the water used for cleaning the exhaust gas in the EGR unit. An EGR blower controls the flow of gas to a mixing chamber. In the EGR Unit the gas is sprayed with water and cooled to form mist, that can be collected by a Water Mist Catcher (WMC). Water from the EGR unit is drained to a Receiving Tank Unit (RTU) and recirculated to the EGR unit. Part of the recirculated water is led to a Water Treatment Unit (WTU) to be cleaned and returned to the EGR unit. The surplus of water originating from the combustion process is drained from the WTS as bleed-off water and discharged to the sea. The residuals from the cleaning process are discharged to the sludge tank. Depending on engine load and ambient conditions the combustion process will accumulate water in the system, which must be discharged as bleed-off water. If discharged to the sea, the bleed-off water must meet the quality criteria required by Organization. (2013), presently defined in the 2015 Guidelines for Exhaust Gas Cleaning Systems, MEPC 259 (68). Bleed-off water, which does not meet the discharge criteria or cannot be discharged to sea due to local restrictions, is drained to a drain tank for delivery at port.

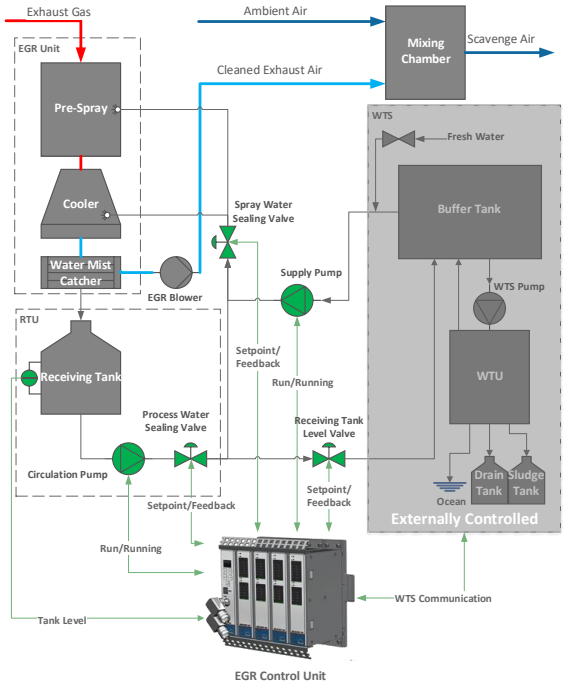

Figure 5.4: EGR Water Handling System.

The WHS system is illustrated on figure 5.4 together with the EGR Control Unit (EGRCU) responsible for controlling the cleaning process. The EGRCU controls and monitors the WHS, with the objective of providing clean gas to the scavenge air manifold. There are two main water loops to be controlled. The recirculation loop, where the water from the EGR unit is sent to the RTU and back again by the 'Circulation pump' via the 'Process Water Sealing Valve' and 'Spray Water Sealing Valve'. The other loop is where part of the water from the recirculation loop is sent via the 'Receiving Tank Level Valve' to the WTS. The water from the WTS is sent back to the recirculation loop with the 'Supply Pump'. The WTS receives the processed water from the RTU and is collected in the buffer tank. The WTS is a separate system, provided by a MD&T OEM, that cleans the process water in the buffer tank and discharge any residual waste water in either the sludge/drain tank or, if the water quality parameters are fulfilled, sends the water overboard.

The objective of the control system is to maintain a clean water supply for the cleaning process and a stable water level in the 'Receiving Tank'. During start up and shutdown of the WHS the actuation timing of the components has a direct impact on the water level. During running mode, the water level is controlled by the 'Receiving Tank Level Valve' and compensates for deviations in the water flow due to e.g. engine load, exhaust gas and scavenge air pressure changes.

5.5.2 Results of Traditional Development Process

The traditional development approach was used to create a control strategy for the WHS. The control strategy consist of a PI controller that regulates the process water tank level and a state machine for actuating valves and pumps according to a number of states for starting, running and stopping the system. After development in the SIL environment using a DSE model of the WHS physics was finished, the system was tested on the HIL test bench with the real EGRCU controller and the DSE model running in the ECU as illustrated on figure 5.5.

As final validation a test session was performed on the engine test bench. The test proved that the PI controller worked as intended, however, an unsuspected situation occurred when stopping the WHS system.

On figure 5.6 the results from the engine test bed running the initial control strategy is plotted. We see that after 100 seconds the EGR control system is started by an operator which orders the WHS system to prepare for EGR operation. 50 seconds after the state-machine has finished starting the different

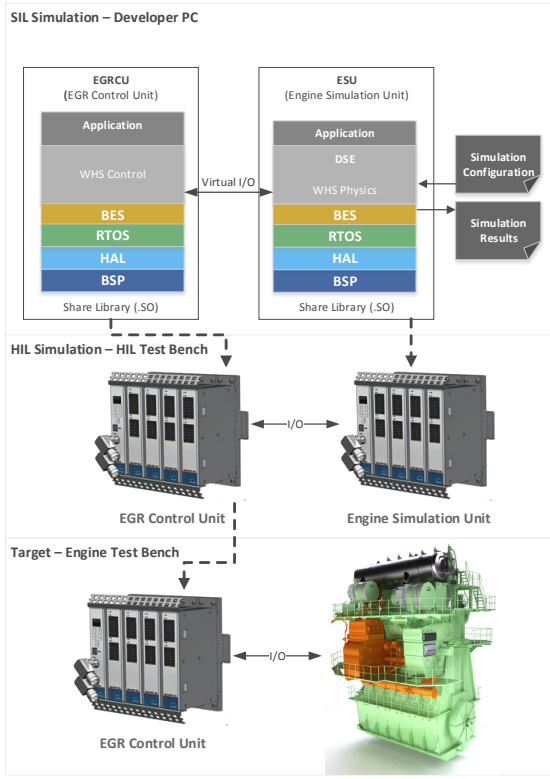

Figure 5.5: Traditional validation process.

pumps and opening valves, Process Water Receiving Tank (PWRT) control is fully engaged. The bottom plot on figure 5.6 show how the Receiving Tank Level Valve (RTLV) is controlled for stabilizing the PWRT level and redirecting water from the process circuit to the WTS for cleaning. The top plot on figure 5.8 shows the level of water in the PWRT. The water level became stable after a transient period, proving that the PWRT control worked correctly during WHS operation. The EGR system was ordered to shut-down by an operator after 600 seconds, and we see that WHS state-machine starts to empty the tank to reach a stable offline level around 20-25 %. However, at 676 seconds a behaviour not seen in either the SIL or HIL validation was observed. When the WHS system is started, water will flow and accumulate gradually in the WMC. An equilibrium will be achieved due to increased water pressure resulting in a consistent flow through the WMC (without increased water accumulation in the WMC as a consequence). During shutdown, the control system will reach

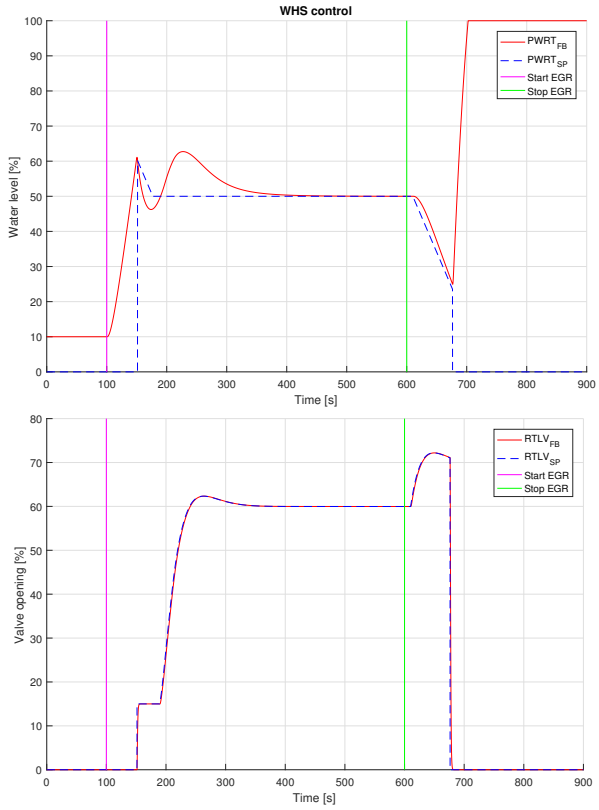

Figure 5.6: WHS Control results.

a desired water level in the PWRT and RTLV control will be stopped, at this point the accumulated water in the WMC starts to flow to the PWRT tank. As seen on figure 5.6, the amount of accumulated water in the WMC is so large that it overfills the PWRT.

From the engine test bench validation, it was discovered that the controller actuating the RTLV was working properly, but the state-machine was not properly handling the emptying of the WMC. Engine test bench sessions are very costly and one of the ambitions with this research was to enable a more efficient development process, where high fidelity physical dynamic models can be used during control system development to holistically validate the system at an earlier stage. The normal development approach would be to extend the DSE model to correctly represent the accumulation of water in the WMC and make the changes required to the control system. However, with the development of the distributed co-simulation environment it is now possible to reuse the MAT-

LAB model developed during the initial investigations. This model included element such as water accumulation in the WMC, but they were disregarded when translated to the SIL DSE environment. The DSE is central to development because it is designed for the target platform and directly enable validation on the HIL test bench. Keeping this in mind it is rational to have a higher fidelity modelling tool for SIL investigations and DSE models for HIL validation. The purpose of the HIL test is not to test functionality already verified in SIL, but to ensure computational overhead and investigate temporal aspects. The distributed co-simulation solution is generic and allows for multiple well known modelling tools in the physical domain to be used without concerns of platform and architecture compliance.

5.5.3 Distributed Co-Simulation Configuration

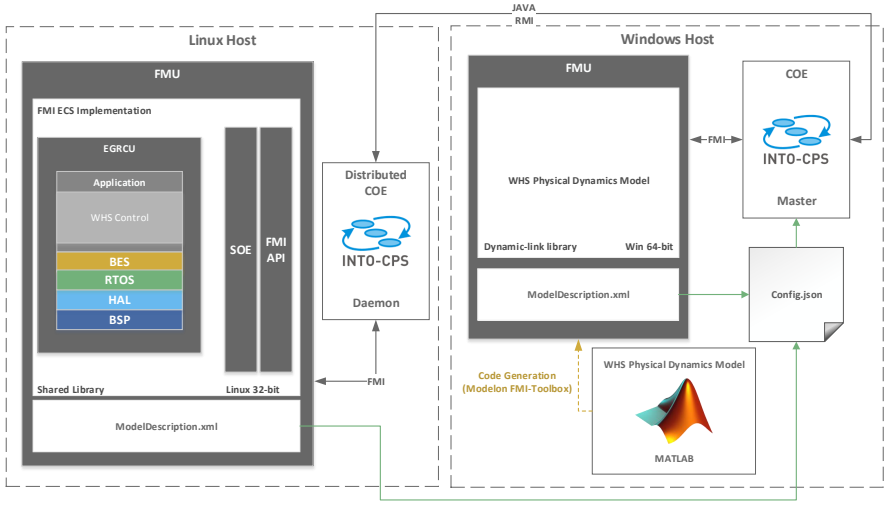

Figure 5.7: Co-Simulation Configuration.

The WHS distributed co-simulation setup is illustrated on 5.7. On the Windows host the master Co-simulation Orchestration Engine (COE) is running and the DCOE-deamon on the Linux host. A JSON configuration file describes where the FMUs are located and on which host-ip they should be executed. The configuration file also contains information about connections between the inputs and outputs of the FMUs, parameters and simulation algorithm: variable/fixed time step. The WHS phyisical dynamic model from MATLAB is code generated into an FMU using the Modelon FMI toolbox for MATLAB/simulink. The tool-

box compiles the MATLAB model to a 64-bit DLL including the FMI-API and auto-generates the model description XML defining the interface to the FMU. The ECS FMU with the EGRCU controller is created as described in chapter 4 and compiled into a Linux 32-bit shared library. The simulation can access the SOE for scheduling and access variables of the WHS control component. The distributed co-simulation is initiated through the COE and results delivered in CSV format on the Windows host.

5.5.4 Results of Distributed Co-Simulation

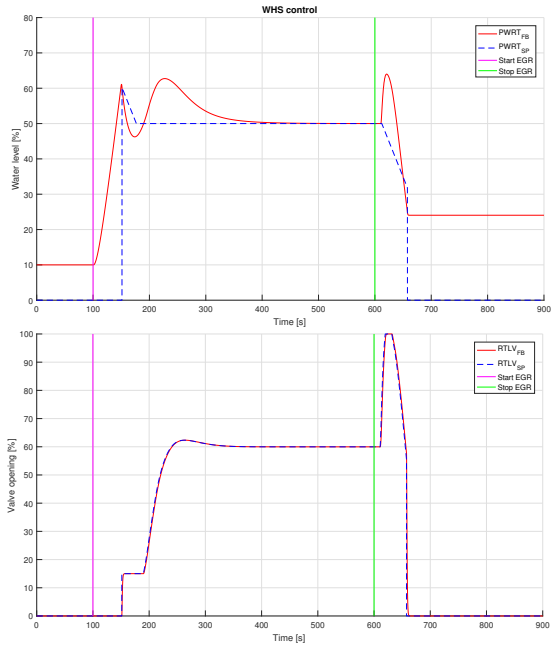

Figure 5.8: Results of the distributed co-simulation of the WHS.

Using the distributed co-simulation, where the DSE model of the WHS physical dynamics have been replaced with at MATLAB model, control engineers at MD&T were able to develop a new control strategy handling the WHS correctly. The simulation results of the new controller is presented on figure 5.8. Here we see that the PI controller for the PWRT level is still working properly and that the new state machine is now able to handle accumulation of water in the WMC. This prevents the water level in the PWRT from overflowing, but instead stabilize at the desired level of approximately 20-25%.

This experiment proves that with the higher fidelity model formulated in MATLAB developers where able to create working control strategy. Had the distributed co-simulation been used for initial control development, the issues seen on the test engine would likely have been discovered at an earlier stage, saving money and time.

5.6 Chapter Summery

In large organizations like MD&T multiple tools are available on different platforms and architectures. This platform and architecture deviation is a challenge when trying to enable cross department model sharing. The Integrated Toolchain for the model based design of Cyber-Physical Systems (INTO-CPS) was used as co-simulation manager and in collaboration with the INTO-CPS development team, an extension to their co-simulation orchestration engine was created that enable distributed co-simulation across both platform and architecture. Using INTO-CPS with the newly developed distributed co-simulation orchestration engine, it was possible to help the development of a new EGR Water Handling System (WHS). The traditional development process was initially used to create a control strategy for the WHS control. The initial solution did not successfully handle the shutdown of the system, which was discovered during the integration test on the engine test bench. Using the distributed co-simulation environment it was possible for engineers to express the physical dynamics of the WHS in a higher fidelity model created in 64-bit MATLAB on Windows and co-simulate that together with the 32-bit ECS Linux simulation. They where then able to replicate the results experienced on the engine test bench and correct the control strategy on design level using the MIL/SIL distributed co-simulation. It is argued that, if the distributed co-simulation environment had been available during the first phase of development, the improprieties of the initial control strategy would have been discovered during the design phase. Resulting in a significant cost saving on validation and verification of the system.

Chapter 6

Hybrid Co-Simulation

The development of, especially, safety critical Cyber-Physical Systems is highly dependent on human interaction and cognitive assessment. Despite this dependency, the human in the loop is seldom an integrated part of CPS development or tool chain. Most Human Machine Interfaces (HMI) are not representative before connected to the actual hardware, making their development delayed compared to the system development. Furthermore, real hardware and test setups are often limited resources, especially when dealing with large or expensive equipment. The combination of limited resources and delayed development makes it difficult to thoroughly investigate human interaction and, therefore, to design systems tolerant towards user error and misuse. This chapter presents an extension to the co-simulation environment, where the HMI hardware can be connected with the ECS SIL simulation in a hybrid co-simulation. The environment makes it possible to investigate human interaction before new software is released, making the HMI and control system development more concurrent.

6.1 SW/HW Co-Simulation - HMI to ECS connection

The HMI for the ECS is called the Engine Control Main Operating Panel (EC-MOP), as presented in section 2.2.3. The EC-MOP is connected to the ECS through an Ethernet LAN network. As presented in section 3.4.1, a virtual switch has been implemented in the simulation orchestration engine, where ports to the controllers network driver are connected and Ethernet packages distributed on a MAC Address level. This solution ensures that network communication during simulation is identical to the real network communication, meaning that packages going to the EC-MOP are already part of the simulation, however until this point has been disregarded. To enable communication between the ECS simulation and the EC-MOP, all we need to do is connect the virtual switch with the physical network adapter of the PC connected to the EC-MOP.

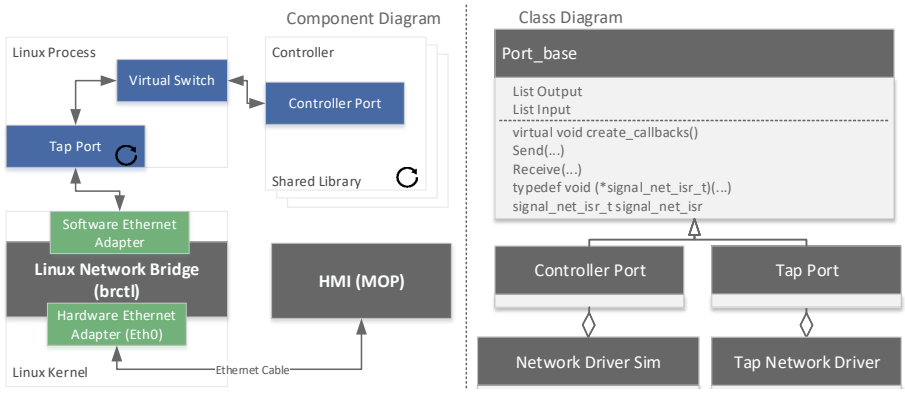

Figure 6.1: ECS Simulation with Tap Driver and Network Bridge to EC-MOP.

An additional Port class has been created, that makes it possible to add either a controller port or a so called Tap port. A Tap Port contains the same components and functionality as the controller port described in 3.4.1. It implements the same callback delegates as the controller port, but instead of connecting to the network driver of the controller it connects to a tap network driver implemented in the SOE of the main Linux process. The tap network driver connects the port to a tap interface, a software network adapter that only exists in the Linux kernel. The interface works as a regular network adapter, where the kernel exchanges full Ethernet frames from and to the network driver instead of a regular wire. The *signal_net_isr* method is allowed to execute directly, since the tap port is located within the Linux context, as opposed to the controller

network driver that runs in the controller context. To connect the Tap interface software Ethernet adapter with the physical Ethernet adapter connected by LAN to the EC-MOP, a Linux software network bridge is created as seen in figure 6.1. Multiple Tap interfaces and port instants can be created, making it possible to connect the virtual switch to not only the EC-MOP and ECS simulation but also other physical hardware. It would be relevant to connect a subset of Triton controller to the remaining SIL controllers of the ECS by introducing an additional "proxy" controller with the purpose to redirect IO data through the virtual switch. This will be included in the future work and will not be covered here.

It should be noticed that connecting real hardware will require that the system is real-time compliant. The previously presented co-simulations were allowed to run as fast as possible but the hybrid co-simulation has to execute real-time if hardware is connected. This is achieved by letting the process sleep after execution for the remaining amount of time in the simulation time step. Furthermore, connecting a real Ethernet interface destroys the determinism of the co-simulation, meaning that simulation reproducibility is no longer ensured and the system is no longer appropriate for e.g. regression tests.

6.2 Hybrid Co-Simulaiton Configuration

With the virtual switch extension to the co-simulation it is now possible to connect both MIL, SIL and HIL systems in a holistic co-simulation. On figure 6.2, the hybrid co-simulation has been illustrated with three main stakeholders of the MOP. The marine engineer/operator is the main user of the HMI interacting with the system according to bridge commands and in response to information from the operating panel, alarm system, illustrations, etc. The control engineer developing control algorithms is also a stakeholder. The control engineer is responsible for developing the correct interaction possibilities and create alarms for new components. The work of the control engineer is, therefore, dependent on an understanding of the cognitive assessment of the operator. Lastly, the Graphical User Interface (GUI) developer is responsible for the user experience and the graphical representation of the system. All stakeholders are interconnected and the communication and joint understanding between them is important. If for example the operator needs to respond to an alarm, it is equally important that the alarm text, formulated by the control engineer, is explanatory and that the user interface, made by the GUI developer, sufficiently attracts the attention of the operator. With the hybrid co-simulation it is possible to create simulation scenarios in a physical dynamics model that propagate through the control system to the operating panel and require user

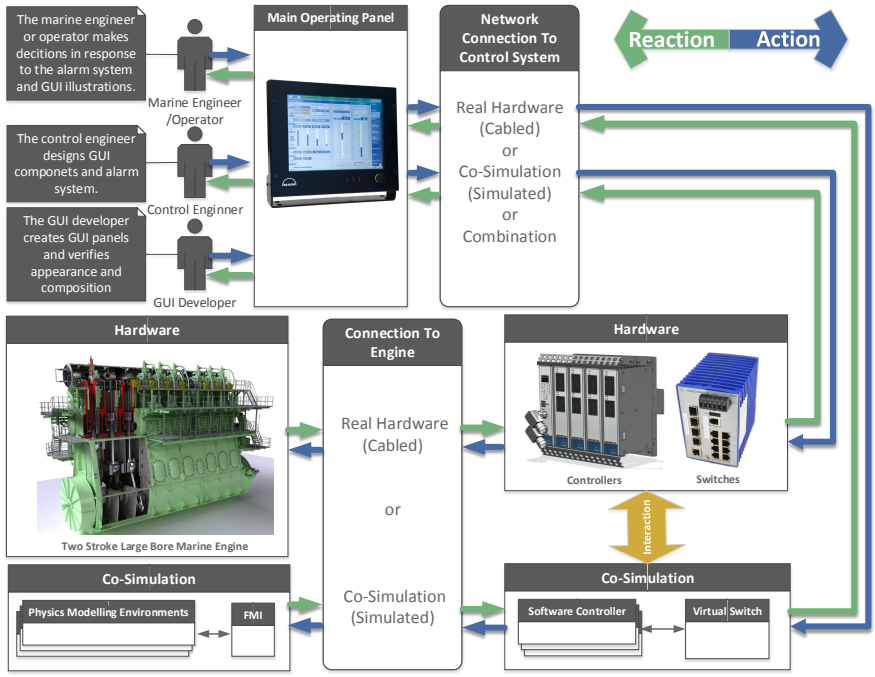

Figure 6.2: Hybrid Co-Simulation setup.

interaction. In the SOE it is possible to track and store the operator interaction when exposed to specific scenarios. Utilizing the collected data, the human in the loop can be investigated during system development to gain quantitative and evidence based data for designing safety critical CPS with human-machine interaction.

6.3 Human In the Loop Investigation

This section presents an experiment that was published in Pedersen et al. (2017a), and is available in chapter 10. Here a thermodynamic model of the ME engine air-path with an exhaust gas bypass is co-simulated in an internal MD&T tool together with the ECS simulation connected to the EC-MOP.

Figure 6.3 illustrates components with the dominating dynamics in the air-path model, which focused on the mass flows and pressures through the system. The

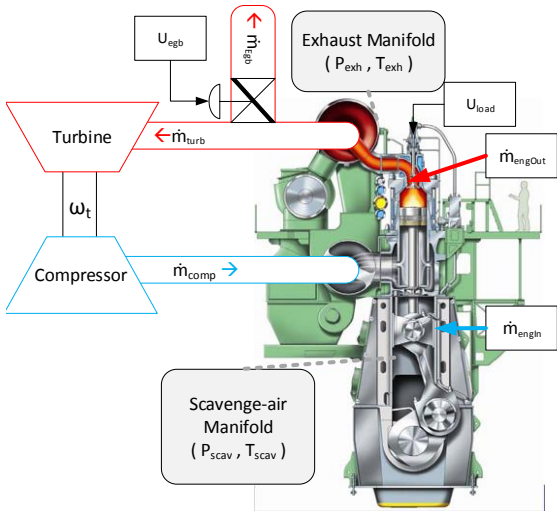

Figure 6.3: Engine air-path model.

model has previously been published and validated in Alegret et al. (2015) and is based on research done by Wahlstrom and Eriksson (2011); Hansen et al. (2013). In Alegret et al. (2015) the model also include an EGR system. The EGR is not activated in the scenario presented here and will therefore not be covered. A turbocharger consists of a turbine and a compressor connected by a common shaft. The exhaust gas from the engine drives the turbine generating power transferred through the shaft to the compressor forcing ambient air into the scavenge-air manifold. The mass flows from the compressor $\dot{m}_{comp}$ and turbine $\dot{m}_{trub}$ are dependent on pressure ratio and turbocharger velocity described in the performance maps provided by the turbocharger manufacturer. The power difference between the turbine and the compressor is used in the state equation expressing the turbocharger velocity ω_t. The scavenge air and exhaust gas manifolds are modeled as control-volumes based on the ideal-gas law and mass conservation with state equations describing the pressures P_{scav} and P_{exh}. The engine is modeled as a flow through a restriction $(\dot{m}_{engIn}, \dot{m}_{engOut})$ with a cylinder temperature T_{exh} based on the Seiliger cycle. For further details of the system refer to Alegret et al. (2015). The Exhaust Gas Bypass (EGB) is not presented in Alegret et al. (2015) and will be described in detail here. The objective of the EGB is to control the power delivered from the exhaust to the turbine and provide energy for downstream systems such as waste heat recovery. This is achieved by redirecting the exhaust flow from the turbine inlet to the exhaust pipe by a EGB-valve. The mass flow leaving through the EGB $\dot{m}_{egb}$ is

modeled as flow through a restriction.

$$\dot{m}_{egb} = U_{egb}\ A_{egb} \frac{p_{exh}}{\sqrt{R_e T_{exh}}} \sqrt{\frac{2\gamma_e}{\gamma_e - 1}\left[\frac{p_{egb}}{p_{exh}}^{\frac{2}{\gamma_e}} - \frac{p_{egb}}{p_{exh}}^{\frac{\gamma_e+1}{\gamma_e}}\right]}, \tag{6.1}$$

where A_{egb} is the maximum EGB-valve orifice and U_{egb} the valve position, γ_e is the ratio of specific heat, R_e is the exhaust-gas constant and P_{egb} is the back pressure from the subsequent system.

The model inputs are the EGB valve setpoint U_{egb} and engine load U_{load}, a non-dimensional power defined as a percentage of the maximal power available for the specific engine. The remaining components are assumed to have little impact on the air flow dynamics and their dynamics disregarded. All cooling in the system is assumed ideal meaning that the temperature of the gas leaving and entering a manifold is assumed to have the exact same temperature. Heat transfer is also neglected, meaning that there is no temperature drop, e.g. from the cylinder to the exhaust gas receiver.

6.3.1 Simulation and Results

The aim of the hybrid co-simulation is to create an environment where a scenario engaging the human operator can be formulated and user interacting recorded. Scenarios that are potentially hazardous can be created and simulated without endangering operators or expensive equipment.

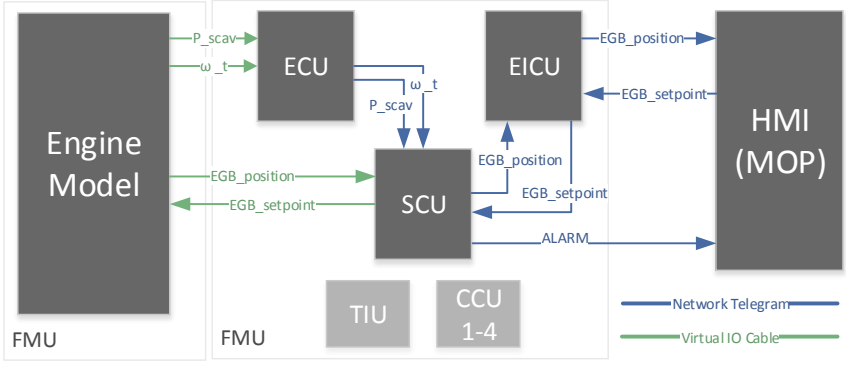

Figure 6.4: The two FMUs and the MOP are illustrated with relevant cable and network connections.

A proof-of-concept scenario has been defined focusing on the EGB control. Closing the EGB-valve will cause the turbine to receive the full power of the exhaust gas. This increases the angular velocity of the common shaft and, thereby, the mass flow from the compressor, causing the scavenge-air pressure (P_{scav}) to raise. Opposite when the EGB-valve is opened, the flow to the turbine decreases and P_{scav} drops. In certain situations both scenarios can be very undesirable. If the turbine velocity increases too much the turbine may be destroyed or even explode with extreme danger to the crew. If P_{scav} drops significantly the engine may suffocate and forcing operation to be stopped. These are of course extreme situations, where the safety-critical system has not been working. With functioning safety systems these scenarios would cause the alarm system to notify the operator and if no action is taken send a slow-down or shut-down command to the ECS bypassing the operator. A slow-down and shut-down command severely limits the maneuverability of the vessel and are very undesirable measures. In the scenario presented here, the simulation is configured as seen in Figure 6.4. The turbocharger velocity ω_t and the scavenge air pressure P_{scav} are provided by the engine model to the Engine Control Unit (ECU) through virtual IO analog cables in the SOE as described in chapter 3. The EGB model is connected to the Scavenge-air Control Unit (SCU) with a control set-point $EGB_{setpoint}$ and actual valve position $EGB_{position}$. Within the ECS simulation data is transferred by network telegrams through the virtual switch between the controllers and the EC-MOP. The Tacho Interface Unit (TIU) and the four Cylinder Control Units (CCU) shown on figure 6.4 are required to properly simulate the system and are connected to the EC-MOP as well. The control signal representing engine load U_{load} is normally a command from the bridge, here it will be set at 40% load and provided internally in the engine model.

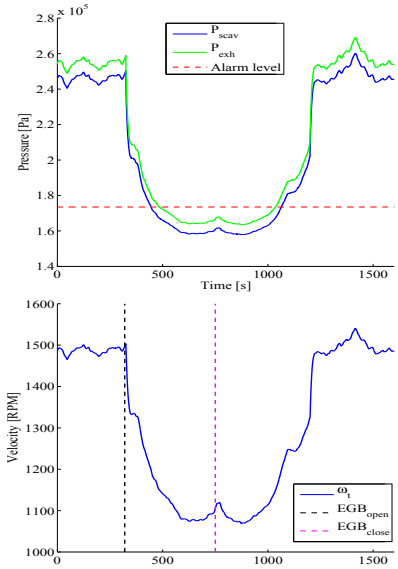

Figure 6.5: Plot of pressures and turbocharger velocity during the simulation scenario.

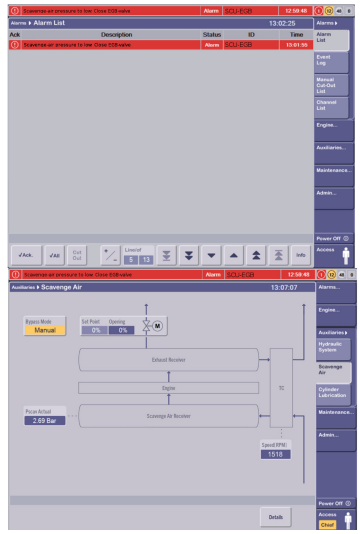

Figure 6.6: Top: Alarm panel, Bottom: Scavenge Air panel

The simulation results are presented on figure 6.5. The scavenge-air and exhaust pressures start at a stable level around 2.5×10^5 Pa, but after 320 seconds the EGB-valve is opened by the engine model. This causes the turbine velocity to drop dramatically and as a consequence, both P_{scav} and P_{exh} will decrease to a level below the alarm limit of 1.75×10^5 Pa. An alarm will be triggered on the SCU and displayed on the EC-MOP. The operator will see the alarm appear on the top bar of the EC-MOP application and in the alarm list panel, as displayed on the top of figure 6.6. The simulation will track the alarm and record when and how the operator responds. To handle the scenario correctly the operator should navigate to the Scavenge Air panel as seen on the bottom of figure 6.6. Here he has the option of either changing the EGB-valve setpoint to 0% or set the EGB-controller in automatic mode which will likewise close the valve. Both actions will cause an increased exhaust gas flow to the turbine and P_{scav} will return to a stable level. On figure 6.5 it is also shown how the MOP-command has ordered the EGB-valve to close at 750 seconds, causing ω_t to increase and returning the pressures to a stable level. The effects of closing the EGB valve are delayed due to the dynamics of the system.

This simple simulation scenario proves that it is possible to expose an operator with at specific situation created with a thermodynamic model interacting with

the real software of the control system connected to the HMI. With a hybrid co-simulation environment like this, it is possible to thoroughly investigate human interaction with in cyber-physical systems to be used for development insights otherwise difficult to come by.

6.4 Chapter Summery

As a result of this research an additional application of the co-simulation environment was identified. When developing safety critical cyber-physical systems the human interaction and cognitive assessment is of great importance. Quantitative and evidence based data on the human factor is difficult to collect. With an extension to the co-simulation environment, it is possible to connect the human machine interface and include the Human in the loop in a hybrid co-simulation. An extension to the virtual switch in the ECS simulation makes it possible to connect a Linux software network adapter called a Tap as a port. The Tap exchanges full Ethernet frames with the virtual switch and thereby connects to the controller network. Using a Linux network bridge the Tap adapter can be connected with the physical network adapter of the developer PC, which can then connect to the HMI. Using the hybrid co-simulation it was possible to create a holistic simulation consisting of a physical dynamic model co-simulated with the ECS software simulation connected to the MD&T HMI. An experiment is presented, where a scenario is formulated in the physical dynamic engine model that require the attention of the human in the loop. The experiment showed a potentially hazardous scenario where the operator has to interact with the control system through the HMI to get the system back to stable operation. The operator interaction was tracked by the simulation and the data recorded can be used later for analyzing the applicability and intuitiveness of the system. With this hybrid co-simulation, it will be possible to base HMI development on real quantitative data, it can also be used as an education tool for engine operator training.

Chapter 7

Conclusion

This dissertation aimed to improve model based design of Cyber-Physical Systems with distributed embedded control through co-simulation. The first objective was to enable simulation of an entire distributed control system in a single coherent and deterministic simulation. With the engine control system at MAN Diesel & Turbo as case study, a simulation approach of distributed embedded systems was presented. The embedded system software was first cross-compiled to an x86 PC application by adapting the RTOS, BSP and HAL layers of the software. Compiling the controller software into a dynamic link library and overwriting the main function makes it possible to load multiple controller instances into a main process and executing their main function in individual threads. Control of execution on each controller can be made possible by re-implementing the idle thread in the BSP of the controller, enabling the main process to start and stop execution. In the idle thread, an event handler can be created, which makes it possible to subscribe events such as operating system clock progression and interrupts. Introducing a system time clock in the event handler controlled by the timing of the events it was possible to have an aspect of time on each controller. Through a custom factory exported from the control library, delegates from the main process could be created and a Simulation Orchestration Engine (SOE) built. The SOE is able to interact with each controller and orchestrate a temporally correct execution of the complete simulation. Communication between the controllers in the engine control system was based on Ethernet. A virtual switch was created in the SOE where port to

controllers could be attached and Ethernet packages distributed. A port implements a send and receive method through the factory, for transferring Ethernet packages to and from the simulated version of the controller network driver. The port also implement an interrupt delegate for signaling the controller to process packages. With these adaptations to the control system software it was now possible to simulate the complete distributed system in a single coherent software in the loop simulation.

The second objective was to enable co-simulation of the distributed control system simulation together with a tool for expressing high fidelity physical dynamics. The standardized co-simulation interface, the Functional Mockup Interface (FMI), was chosen due to its flexibility, wide acceptance in the industry and compliance with a vast amount of tool already used at MD&T. FMI can be implemented in the control system simulation by compiling the SOE to a dynamic link library that exports the FMI application interface. The controller libraries was loaded in the SOE by the FMI instantiate method and the FMI step function provides loop conditions for the main simulation loop in the SOE. Data exchange was done through the FMI Get/Set methods where FMI data types are converted to controller data types using a template implementation. Access to variables in the ECS controller data tree is created through the factory providing proxies in the SOE. With the FMI implementation it was possible to co-simulate the ECS together with any other sub-system compliant with the standard. During this research the ECS has been co-simulated with both Ptolemy II, an in-house physical dynamic tool and MATLAB/Simulink using INTO-CPS as manager. As proof of concept, a SCR reactor heating model was expressed in Ptolemy II and co-simulated with the ECS, published in Pedersen et al. (2016). The results from this simulation proved that it was possible to co-simulate the distributed control system software together with a physical dynamics model in a tool optimal for building high fidelity models.

In large organizations like MD&T, multiple tools are available and tool-chains used by different departments often deviate. Besides from using different tools, also development platform and architecture deviate. At MD&T most engineers developing physical dynamic models will work on a Windows platform with an 64-bit architecture, whereas the control system is developed in 32-bit Linux. This discrepancy makes it difficult to properly share model and do co-simulation. Using the Integrated Tool-chain for the model based design of Cyber-Physical Systems (INTO-CPS) as co-simulation manager and collaborating with the INTO-CPS development team to create an extension to their co-simulation orchestration engine, it was possible to do distributed co-simulation across both platforms and architectures. The co-simulation was distributed by having a Distributed Co-simulation Orchestration (DCOE) running as a daemon on the distributed host, with communication between the DCOE and the COE, running on the main host, realized using Java Remote Method Invoca-

tion. With the distributed co-simulation it was possible to aid development of a new system being developed at MD&T called the EGR Water Handling System (WHS), published in Pedersen et al. (2017b). WHS is a process plant used for cleaning exhaust gas recirculated back to the intake-manifold. The WHS had been developed using the traditional approach and during system validation on the engine test bench it was realized that the control strategy did not properly handle shutdown of the system. Using the distributed co-simulation environment it was possible for engineers to co-simulate the 32-bit ECS Linux simulation together with at high-fidelity model of the WHS dynamics, created in 64-bit MATLAB on Windows. The MATLAB model included dynamics that had been disregarded in the DSE model, used for SIL and HIL. It was possible to replicate the results seen on the engine test bench and correct the control strategy on design level using the MIL/SIL distributed co-simulation. Had the distributed co-simulation environment been available during the initial design of the WHS, it is very likely that the model driven approach would have enabled developers to create a functioning control strategy before doing test on the expensive test-engine.

As a result of this research a new application of the co-simulation environment was identified. Developing cyber-physical systems with a human machine interface, the human interaction and cognitive assessment is of importance for the safety critical aspects of the design. It is often very difficult to obtain quantitative and evidence based data on the human factor. With an extension to the virtual switch, a Linux software network adapter could be implemented that connects the MD&T HMI and the ECS simulation through a Linux network bridge. With this hybrid co-simulation it was possible to create scenarios that require operator interaction in a physical dynamic engine model propagating through the ECS to the HMI. By tracking the human interaction it will be possible to collect data to be used for analyzing the system applicability and intuitiveness insuring correct operation of MD&T engines. An example of a potentially hazardous scenario was presented in Pedersen et al. (2017a), where operator interaction was needed to stabilize the system.

This dissertation contributes with a solution for co-simulation of cyber-physical systems with a distributed control system. A detailed description of how embedded system software can be adapted to enable deterministic simulation was presented. Being able to simulate an entire distributed control system, with the real software and network communication on a single PC, is very powerful and enable engineers at MD&T to spend more time on design and less time on the limited amount of HIL test benches. The FMI co-simulation standard enable the ECS to be simulated together with multiple other specialized modelling and simulation tools. At MD&T increasing system complexity require higher fidelity models of the engine physical dynamics than currently expressed in the modelling tool used during control development DSE. With co-simulation it is

possible to use tools such as MATLAB, optimal for efficiently developing high-fidelity models. System validation on engine test benches is extremely expensive and significant cost saving is expected by using co-simulation for reducing the amount of design and verification loops. Model sharing between departments at MD&T has been difficult due to the deviation in tool-chain, platform and architecture. With the extension to the INTO-CPS enabling distributed co-simulation, model sharing has been done significantly easier. It is expected that the distributed co-simulation environment will reduce the amount of redundant modelling effort and enable both physical system developers to get a better insight in the control system and vice versa. Furthermore, since the FMI is an open standard, it is expected that closer collaboration with OEMs would be possible in the future.

Human error and misuse is difficult to guard against and understanding of the cognitive assessment of an operator can be very beneficial in this aspect. The hybrid co-simulation environment can provide data otherwise hard to obtain, when developing human machine interfaces. Another way of guarding a system against human error is proper training and education. At the MAN PrimeServ Academy Copenhagen, marine engineers are educated in using the different MD&T systems. However, the emulators and test-engines available to students are very limited resources due to their size and immense cost. With the hybrid co-simulation environment the cost of a simulator would be reduced to a single PC per student. The environment can provide more sophisticated thermodynamic models than the HIL models currently used in the academy. Partners around the world, educating marine engineers, would benefit significantly from using this hybrid co-simulation environment as an education tool, which in return would benefit MAN Diesel & Turbo and vessel owners with better operated engines.

In conclusion, this research has improved the design phase of the development process at MAN Diesel & Turbo. Resulting in significant cost savings on system verification and validation with a simulation environment prepared for future challenges.

This research has proved the relevance of simulating embedded control systems as opposed to pure emulation. The work required to adapt the embedded system software (RTOS, BSP and HAL) to enable simulation is significant but manageable and we believe applications within embedded system simulation would significantly increase if the developers of e.g. RTOS had more focus on making simulation available. The hybrid co-simulation environment was not part of the initial scope of this research, but provides multiple new applications and products from its mixture of MIL/SIL/HIL abilities. Educating the people operating CPS is of huge importance and is becoming a large industry in itself. The hybrid co-simulation environment could be an important addition to the education of

marine-engineers. Compared to the emulators currently available, it is possible to provide e.g. cloud based remote education of operators and also education of the MD&T service crew around the world. The co-simulation environment could also be used as an online monitoring tool installed on vessels. Giving that the fidelity of the engine model is sufficient, different signals could be monitored and a residual analysis serve as indicator for worn parts or malfunctioning systems. Finally, the co-simulation environment could also be used for fault diagnostics or condition based maintenance where data from vessels, including engine and sensor data, control system states, weather conditions ect. can be feed to the holistic model and recommendation to clients provided. These and many more applications such as virtual reality etc. could be developed based on the co-simulation environment provided by this research.

Chapter 8

Paper A: Co-Simulation of Distributed Engine Controls System with Thermodynamic Models using FMI & SCNSL

Declaration of co-authorship

According to the Ministerial Order on Doctoral Degrees, Section 5(4), if a dissertation or parts of it are based on previously published papers which are the result of collaboration, a declaration signed by each of the joint authors including the author of the dissertation, and setting out the amount and character of the author's contribution to the work, must be submitted with the dissertation.
Co-authorship is defined according to the Vancouver guidelines, see http://www.icmje.org/index.html

Regarding the following publication:
Nicolai Pedersen, Jan Madsen, Morten Vejlgaard (2015).
Co-Simulation of Distributed Engine Control System and Network Model using FMI & SCNSL

which will be included in the following dissertation:
Co-Simulation of Cyber-Physical System with Distributed Embedded Control

The degree of Nicolai Pedersen's contribution to the publication based on the following scale:

A. has contributed to the collaboration (0-33%).
B. has contributed substantially (34-66%).
C. has to a high degree carried out the work independently (67-100%).

Declaration in each element.	A,B, or C
1. Formulating the scientific idea based on theoretical assumptions to be clarified, including formulation of the question to be answered through analytical work and research plans.	C
2. Planning of experiments and analyses, design of the experimental methods in a way that the questions asked under point 1 can be expected to be answered.	C
3. Involvement in analytical work with respect to the concrete experimental studies and investigations.	C
4. Presentation, interpretation and discussion of the results.	C

Co-authors signature:			
Date	Name	Title	Signature
29/9-2017	Jan Madsen	Professor	
29/9-17	Morten Vejlgaard Larsen	Manager	

Dissertation author's signature

Date: 29/9-2017 Signature:

Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online
CONFERENCE PAPER ARCHIVE

IFAC-PapersOnLine 48-16 (2015) 261–266

Co-Simulation of Distributed Engine Control System and Network Model using FMI & SCNSL

Nicolai Pedersen [*,**] **Jan Madsen** [*]
Morten Vejlgaard-Laursen [**]

[*] *Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark.*
[**] *MAN Diesel & Turbo, Teglholmsgade 41 DK-2450 Copenhagen SV*

Abstract: Increased legislations on engine pollution and efficiency, in the maritime industry, has resulted in drastic changes of engine layout and the amount of interconnected systems. With increased complexity and a more distributed control system, co-simulation and network modelling tools become even more desired. This paper investigates how to co-simulate a distributed cyber-physical system and its surrounding network. The Functional Mock-up Interface (FMI) for co-simulation is used as a standard for interconnecting models and solvers. A network model connecting all subsystem components is created using the SystemC Network Simulation Library (SCNSL). Combining FMI and SCNSL makes it possible to validate the interconnection between both physical and computational components together with the network typology. The result is a more complete simulation environment with better opportunities for investigating proper system behaviour in terms of dynamics and temporal execution. A master algorithm for doing the networked co-simulation routine is proposed, and a proof of concept example is presented, showing one of the many potentials of the environment. With the more comprehensive simulation environment, developers will be able to achieve improved modelling and validation processes, resulting in better applications.

© 2015, IFAC (International Federation of Automatic Control) Hosting by Elsevier Ltd. All rights reserved.

Keywords: Functional Mockup Interface, SystemC Network Simulation Library, Cyber-Physical Systems, Co-Simulation, Simulation Master.

1. INTRODUCTION

When designing the next generation of networked embedded control systems for advanced physical processes, modelling and simulation is essential. Model-based design enable engineers to work at a higher abstraction level, making it possible to select and verify the optimal solution at an early stage of development. Thereby prevent damaging expensive equipment or committing to inappropriate hardware or software solutions. With electronically controlled engines even small variations in the control behaviour has significant impact on the combustion process. Detailed simulations of the engine process combined with high fidelity models of the control system are therefore required. The International Maritime Organization (IMO) often introduce new legislations (Organization., 2013) causing drastic changes to the engine layout. These have caused ships to e.g. have power rating and emission reduction systems, introducing a whole new dimension to the engine design. Further new fuel types and dual fuel engines are emerging, thus also increasing the system complexity. A result of the increased complexity is a more distributed control system with additional network communication. In real-time systems the network performance is of great importance and faults can have significant consequences. At MAN Diesel & Turbo (MDT) these new challenges call for a more collaborative model development than previously. Physical performance modelling will become more dependent of the control system dynamic and the control systems development will require access to comprehensive and detailed engine models. It is therefore, more then ever, desired to develop a co-simulation [1] tool, enabling departments to share models across solvers and environments.

This paper demonstrate how the Functional Mock-up Interface (FMI) (Blochwitz et al., 2012) can be used to co-simulate a cyber-physical system, such as the MDT two-stroke diesel engine with its engine control system. FMI was the result of the ITEA2 project MODELISAR to improve the design of embedded software in vehicles, organized by Daimler AG. FMI has mainly been used in the automotive industry (Abel et al., 2014; Stoermer and Tibba, 2014). Recently, academic applications within energy grid systems (Elsheikh et al., 2013), control systems(Exel et al., 2014) and HVAC systems (Nouidui et al., 2014) are starting to emerge. The interface does not provide a master for scheduling and execution, and only few examples have been proposed (Bastian et al., 2013; Broman et al., 2013), and using High Level Architecture (HLA) compliant Run Time Infrastructure (RTI) in (Awais et al., 2013a,b). A master algorithm will be introduced to organize the co-simulation and a proof of concept example is presented. We propose to use, the SystemC Network Simulation Library (SCNSL)(Fummi et al., 2008) to model the network connecting the sub-systems. The network model can be used

[1] Co-Simulation is the practice of modelling and simulating different subsystems that form a coupled problem in a distributed manner.

2405-8963 © 2015, IFAC (International Federation of Automatic Control) Hosting by Elsevier Ltd. All rights reserved.
Peer review under responsibility of International Federation of Automatic Control.
10.1016/j.ifacol.2015.10.290

to simulate and test different communication scenarios in respect to latency, transit-time, assertion ect..

Connecting FMI with other simulation-tools have previously been done, e.g. with SIEMENS SIMIT 7 to include Shared Memory Gateway (Exel et al., 2014) and integrating Functional Digital Mock-up (FMDU) with FMI has been suggested in (Enge-rosenblatt et al., 2013). Many tools support FMI out of the box or through third party add ons; Dymola, MATLAB/Simulink, SimulationX, GT-Power amongst others. Network modelling within a single tool is common, however combing co-simulation through FMI with a network simulation tool such as the SCNSL has not previously been done.

This paper will start by introducing the cyber-physical system in section 2. Section 3 outlines the FMI 2.0 standard for co-simulation, followed by a description of the SCNSL network model in section 4. In section 5 the implementation of a master algorithm is presented. Results from a proof of concept example is shown in section 6. Finally in section 7 conclusions will be drawn.

2. CYBER-PHYSICAL SYSTEM

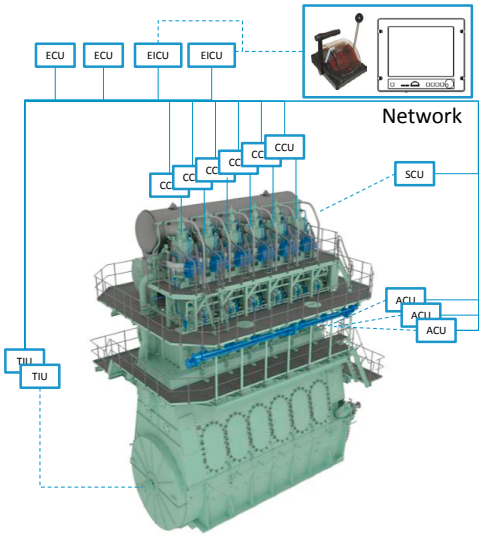

Fig. 1. Illustration of the MAN Diesel & Turbo Engine with distributed engine control System.

The system investigated is the standard layout of a MAN Diesel & Turbo two-stroke diesel engine as seen on Fig. 1, acronyms used can be seen in table A.1. The physical layer is presented by the sensors and actuators connecting the engine with the controllers. The cyber layer is the communication between controllers together with the HMI(alarm system, operating panels, etc.). The engine is electronically controlled by a engine control system, consisting of several embedded controllers. Each controller composes a power module, I/O chassis, DC/DC converter and a FPGA. All controllers are hardware-wise identical, but the software running on the FPGA-core, determines the specific control objective. On Fig. 1, 16 controllers are shown performing 6 different functions/control objectives. The crankshaft position is registered by the TIU, this information is processed by the ECU which schedules tasks for the CCU, SCU and ACU e.g. ingestion timing, wastegate control, blowers, hydraulics and lubrication. The EICU is the interface between the operator and the control system, from here change of engine load, running modes, etc is controlled. Due to fault tolerance measures, many of the controllers are redundant, TIU and EICU on Fig. 1.

The engine is modelled using various commercial tools and self developed environments. These environments are not compatible with the control system, which results in a significant deviation between the thermodynamic models used for control system development and control logic used when developing thermodynamics models. Another challenge of simulating the control system is the identical nature of the software architecture across controllers. Each application has to follow a strict singleton structure, ensuring only one instance of a class, which renders it impossible to compile more than one controller in the same executable.

To properly co-simulate the entire system, an interface is required. The interface will have to support multiple platforms and have a clear and organized way of connecting model inputs and outputs. Furthermore it is required that the binaries of the controller software, is handled in a segregated manner to overcome the singleton structure, FMI does exactly this. Using a standardized simulation routine, the interface permits multiple platform compatibility and optimal opportunity for integration with third party commercial tools. FMI also introduce a XML-schema for describing the subsystems and their inputs, outputs and relevant variables. Furthermore, the binaries of each subsystem are packet in dynamic-link libraries, which solves the issue with the singleton nature of the controller software.

On Fig. 2 the system has been divided into three layers. The physical layer with sensors and actuators, cyber layer with communication across controllers and the top layer, test & development, where simulation and test scenarios are defined. FMI is illustrated as the grey area connecting the boarder between the physical- and the cyber-layer. SCNSL is at the cyber level directing all the communication between components and with the user.

With FMI, a complete co-simulation of the control system with various thermodynamic models is made possible. Deterministic simulations are very appropriate when verifying algorithms and unit-testing reliable parts of a system. Cyber-physical, especially real-time control systems, are very dependent and sensitive to communication both between hardware components and computational units. Communication is often LAN or WAN based and does not guarantee determinism witch may lead to inappropriate system behaviour, due to assertion and varying time delays on reference signals. These challenges are commonly solved by ensuring sufficient overhead in the computation scheduling. The correct system behaviour is then verified through comprehensive stress tests on actual hardware. Though immediate sufficient, this approach is not very effective and increased overhead does not guarantee correct execution. Additionally only few failures caused by communication faults are captured through hardware tests.

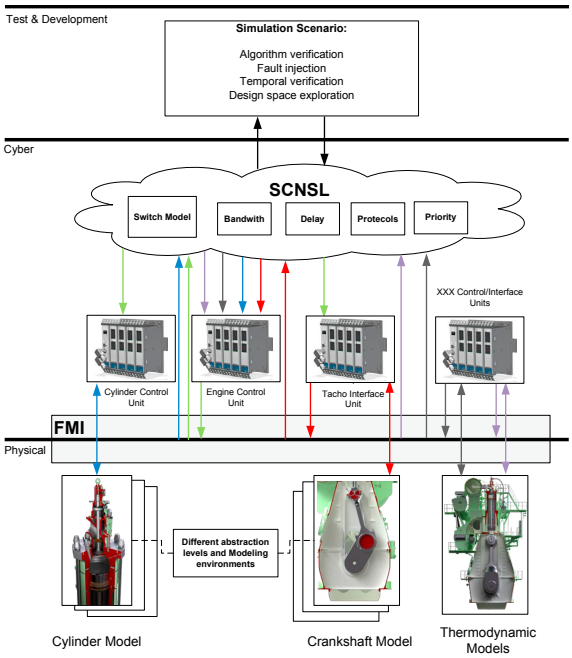

Fig. 2. Cyber Physical System

Introducing a network simulation model, makes it possible to, at an early stage of development, simulate network latency, delay and alternative protocols to determine the optimal overhead on the computational units, minimizing the chance of incorrect temporal execution.

With SCNSL it is possible to model a network representation of the communication between nodes in the engine control system. SCNSL makes it easy to implement and replace protocols and the connection between nodes can be configured to alter propagation speed, delay and priority.

Using the SCNSL simulation kernel, testers and developers can easily define simulation scenarios, do verification and fault injection to test the limits of the system. SCNSL handles the scheduling of the tasks and tracing of relevant reference and output signals.

Combining SCNSL and FMI it is possible to simulate the actual control software with any abstraction of the thermodynamic engine models, combined with a realistic representation of the network connection.

3. FUNCTIONAL MOCK-UP INTERFACE FOR CO-SIMULATION

FMI 2.0 for co-simulation, is a standardized interface for coupling simulation tools, in a co-simulation environment. Each subsystem is solved independently by its individual solver and with data exchange occurring in-between calculations at so called discrete communication points. A Functional Mock-up Unit (FMU) is a subsystem implementing the FMI standard assembled as an archive file containing all the necessary components required to utilize the unit.

Two main components are required to created a FMU; a description schema and an application-interface.

3.1 Description Schema

The description Schema is a XML-file which contain all the information required to connect the subsystems to the co-simulation. Fig. 3 show the top-level of the schema, it should be noticed that optional components are marked with dashed lines, and only their root level is shown on Fig. 3.

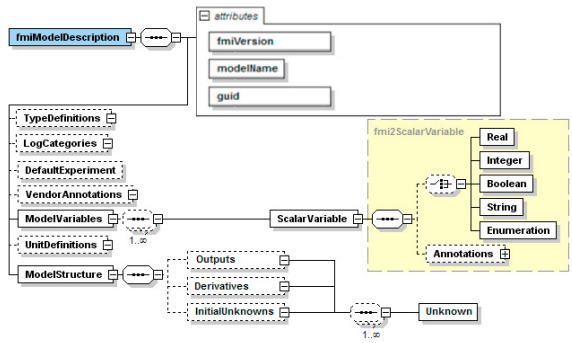

Fig. 3. FMU description schema. Solid-lines are required components, dashed-line are optional. (Blochwitz et al., 2012)

FMUs are required to specify their FMI version, model name and a globally unique identifier (GUID). The essential part of the schema is the ModelVariables, where all FMU variables exposed to the environment, are defined. The ModelStructure is necessary for handling the order of outputs, derivatives and states, together with the variables which are unknown during initialization.

3.2 Application-Interface

FMI defines a number of executable C-function-prototypes that are to be implemented in the FMU. It insures that the definition of types are identical across the entire environment by prepending a prefix to all functions and having type definitions for inputs and outputs. The standard also define data and function types for the interface, which must be used both when compiling FMUs and the FMI master.

The application-interface should be compiled as a dynamic-link library either a Linux shared library or Microsoft DLL. The binary or multiple binaries for cross compatibility are archived in the FMU file together with the description schema.

4. NETWORK MODEL

The network model is created using the SCNSL extension to SystemC. Even before SystemC became an IEEE standard in 2006 (Sys, 2012) it had proven highly efficient when it comes to modelling of virtual platforms. SystemsC is a system-level modelling language that consists of a set of C++ classes and macros which provide an event-driven

simulation. SystemC is continuously being improved in order to achieve better modelling capabilities and ease advanced embedded development. Some of these improvements have become extensions such as Transition Level Modelling, Analog/Mixed-Signal (Grimm and Fraunhofer, 2014; Einwich, 2010), Asynchronous SystemC (Koch-Hofer et al., 2007), SystemC Verification Library and finally SystemC Network Simulation Library (SCNSL) (Fummi et al., 2008) used in this project.

4.1 SystemC Network Simulation Library

The SystemC Network Simulation Library as an extension to SystemC makes it possible to jointly design HW, SW and network in a single simulation tool. The library provides 5 key elements:

- **Kernel:** The kernel is responsible for the execution of events in the correct temporal order and behaviour of the communication channels with features like propagation delay, byte-rate etc.. Since SCNSL is an extension to SystemC it exploits the SystemC scheduler by mapping network events on standard SystemC events.
- **Node:** A node is the active element of the network, it produces, transforms and consumes transmitted data. The node implementation is decoupled from the network simulation which makes it possible for system designers to e.g. change abstraction level, do fault injection, synthesis and validation.
- **Packet:** In packet-switched networks the packet is the unit of data exchanged amongst nodes. In general the packet format is highly dependent on the corresponding protocol. SCNSL does not provide a set of protocols and packet formats, but uses an internal format and lets the designer implement the protocol design on the specific node.
- **Channel:** A Channel represents the physical medium which connects two or more nodes. It can be either a point-to-point link or a shared medium. Multiple channel types are supported; Unidirectional, Half/Full-Duplex and shared.
- **Port:** Every node uses ports to send and receive packets.

4.2 MAN Diesel & Turbo Network Model

A Simplified model of the engine control system network is illustrated on Fig. 4. All nodes are connected, by Ethernet cables, to a switch located in either the engine control room or engine room. Cables are represented by channels in SCNSL, where bit-rate and delays can be adjusted between ports. Each controller is implemented as a SCNSL-node-instance and consists of an input and output controller, implementing the current protocol designers would like to investigate. Network packages are delivered to the node through the simulation kernel, activating the input controller thread upon arrival. Switches are implemented in the same manner as node-instances. Standard switches simply forward communication, but through simple customization they can become a key element for investigating network improprieties. The prioritizing of the network communication is done using the SCNSL-embedded channels that makes it possible to prioritize packages on kernel level before arriving to the switch.

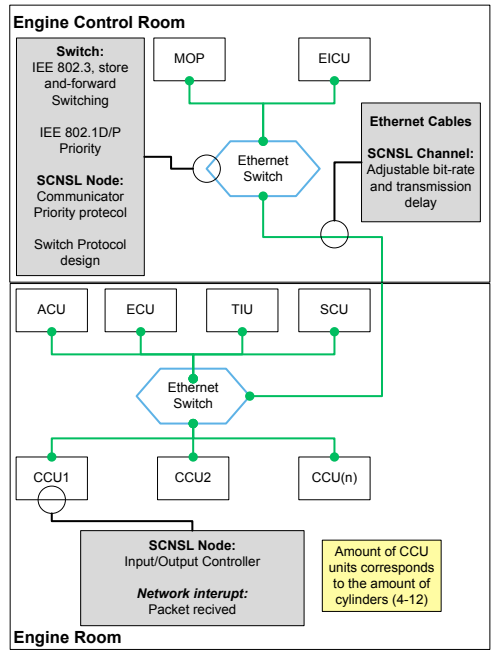

Fig. 4. Simplified Network model of the engine control system on a MDT engine. Acronyms are found in table A.1

5. CO-SIMULATION MASTER

Co-simulation is the discipline of jointly simulating models developed with different tools, where each tool solve parts of a modular coupled problem. Data exchange between subcomponents, is restricted to discrete communication points. In between communication points the subsystems are solved independently. To coordinate temporal execution between subsystems, a simulation master is required. FMI does not provide a master algorithm and no standard FMI co-simulation master has been widely accepted. The master developed for this project implements FMI and links all the FMUs used. Since FMI only support C, and the MDT control software is developed in C++, a wrapper had to be written. The master utilises the SystemC simulation kernel and implements the SCNSL network model as described in Sec. 4. The master is the executable of the simulation environment and here the simulation scenario is defined and executed.

5.1 Simulation Sequence

The co-simulation master is divided into three parts; initialization, simulation and shut-down. A sequence diagram of the master algorithm is seen on Fig. 5 and described below.

Initialization The initialization must be realised in a specific order to correctly instantiate the simulation. First step is to create the network model, here all nodes of

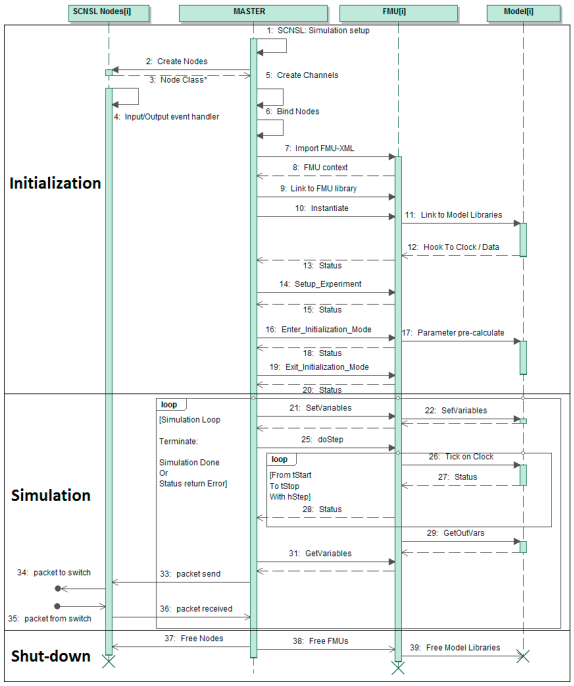

Fig. 5. Calling sequence of FMI Co-Simulation C functions and SCNSL network model.

the network are instantiated. Channels between nodes are created and bound according to the scenario to be tested. When the network is ready the FMUs can be loaded. This is done by first importing the description XML-schema and then loading to the dynamic-link library. The FMUs can then link to the Model libraries, either control software or physical model. After successfully linking to a library, the application will have access to relevant data and the clock of the specific solver [2] . Last is the FMU-initialization, here the inputs and initially unknown variables are set.

Simulation The simulation is a loop terminated either when the scenario is completed or if a FMU returns an error state. The loop step-size corresponds to the distance between discrete communication points. At next communication point all variables are updated, followed by a step on each FMU. On the controller a step corresponds to a relative amount of ticks on the operating system clock. The simulation step and ticks have to match, so time is consistent across nodes. When the step has finished all variables are updated so the parameter bank on the master matches the FMUs. The SCNSL model is simulated with the same step size as the FMU step. If a FMU-output has been updated, a packet will be sent through the network model to the appropriate node. At the first communication point following the reception of a packet, the relevant FMU variable will be updated.

[2] On MAN control software the idle thread of a RTEMS operating system, scheduling the controller-software, is overwritten to provide a hook for the main clock.

Shut-down Here all memory is freed, threads are closed and links released.

6. RESULTS

A proof of concept is presented to show one of the many opportunities resulting from the enhanced simulation environment. The example is configured as illustrated on Fig. 6, and consist of seven FMUs, six of them controllers and one a physical model. A SCNSL-model is representing the network between all the control-nodes connected by a switch. The physical connections between the crankshaft model and controllers are considered immediate and data-exchange is carried out at every communications point. The master has a communication point every 0.5 millisecond, and the network channels have a bite-rate of 100 Mbit/s with priority on packages arriving to the switch. The simulation scenario starts by the crankshaft model

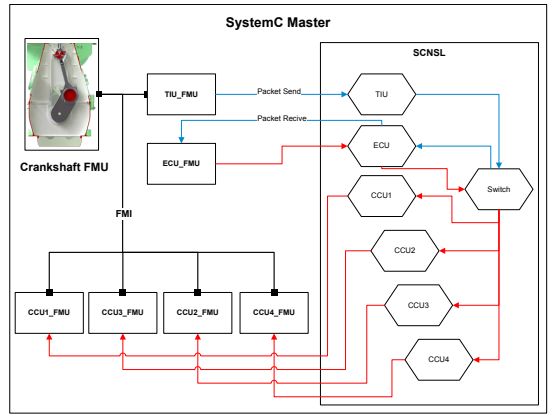

Fig. 6. Example of the simulation configuration

affecting the TIU_FMU. The TIU_FMU reinterprets the I/O signal into a telegram that is sent from the TIU through the network model. The switch receives and forwards the telegram to the ECU node that delivers it to the ECU_FMU at next communications point. The ECU_FMU uses the TIU telegram to calculate the engine speed and piston positions. This information is used to calculate injection commands etc. and the commands are sent through the network to all of the four cylinders CCU1-4.

On the top of Fig. 7 the network communication timing is illustrated. The TIU node receives a signal from the crankshaft model and immediately sends the telegram through the network model. Shortly after the telegram is received by the ECU. At the next communication point the ECU sends its commands to the four cylinders CCU1-4. Chronologically the commands reach the CCUs as fast as the switch allows, in time for the next TIU activation. The bottom image of Fig. 7 show the same system, but with a 50% lower bit rate from the ECU to CCUs. This results in the ECU not being able to send its commands to all the cylinders before a new TIU signal is received.

This is a very simple example of what is possible with the combination of FMI and SCNSL. Many more aspects of timing in the system can be challenged, and the effects of

Fig. 7. Top: Timing diagram of well behaving system. Bottom: Inappropriate timing due to lowering of communication speed

network faults on the entire system can be seen directly within a single environment.

7. CONCLUSION

This paper presented a method for co-simulation of a distributed engine control system with thermodynamic models and their surrounding network. To organize the co-simulation, the FMI for co-simulation standard was used, and to model the network the SystemC network simulation library. A master algorithm for doing the simulations was presented and a simple example of how to investigate assertion showed how easy the systems can be used. The combination of network and co-simulation makes for a more complete environment where developers can investigate both communications performance and its dynamics effects across the entire system. Having a standardized simulation interface eases communication across departments and OEMs, making it easier to verify third part solutions. The result being better models and control algorithms leading to better solutions when facing new challenges.

REFERENCES

(2012). Ieee standard for standard system c language reference manual. doi:10.1109/IEEESTD.2012.6134619.

Abel, A., Blochwitz, T., Eichberger, A., Hamann, P., and Rein, U. (2014). Functional mock-up interface in mechatronic gearshift simulation for commercial vehicles.

Awais, M.U., Palensky, P., Elsheikh, A., Widl, E., and Matthias, S. (2013a). The high level architecture rti as a master to the functional mock-up interface components. *2013 International Conference on Computing, Networking and Communications, Icnc 2013, Int. Conf. Comput., Networking Commun., Icnc*, 315–320. doi:10.1109/iccnc.2013.6504102.

Awais, M.U., Palensky, P., Mueller, W., Widl, E., and Elsheikh, A. (2013b). Distributed hybrid simulation using the hla and the functional mock-up interface@ait.ac.at. *Iecon Proceedings (industrial Electronics Conference), Iecon Proc*, 7564–7569. doi:10.1109/iecon.2013.6700393.

Bastian, J., Clau, C., Wolf, S., and Schneider, P. (2013). P.: Master for co-simulation using fmi.

Blochwitz, T., Otter, M., Åkesson, J., Arnold, M., Clauss, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., et al. (2012). Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *9th International Modelica Conference*.

Broman, D., Brooks, C., Greenberg, L., Lee, E.A., Masin, M., Tripakis, S., and Wetter, M. (2013). Determinate composition of fmus for co-simulation. *2013 Proceedings of the International Conference on Embedded Software, Emsoft 2013, Proc. Int. Conf. Embedded Softw., Emsoft*, 1–12. doi:10.1109/emsoft.2013.6658580.

Einwich, K. (2010). Systemc ams extensions. 1. doi:10.1049/ic.2010.0170.

Elsheikh, A., Awais, M.U., Widl, E., and Palensky, P. (2013). Modelica-enabled rapid prototyping of cyber-physical energy systems via the functional mockup interface. *2013 Workshop on Modeling and Simulation of Cyber-physical Energy Systems, Mscpes 2013, Workshop Model. Simul. Cyber-phys. Energy Syst., Mscpes*, 6623315. doi:10.1109/mscpes.2013.6623315.

Enge-rosenblatt, O., Clau, C., Schneider, A., and Schneider, P. (2013). Functional digital mock-up and the functional mock-up interface two complementary approaches for a comprehensive investigation of heterogeneous systems.

Exel, L., Frey, G., Wolf, G., and Oppelt, M. (2014). Re-use of existing simulation models for dcs engineering via the functional mock-up interface. 1–4. doi:10.1109/ETFA.2014.7005261.

Fummi, F., Quaglia, D., and Stefanni, F. (2008). A systemc-based framework for modeling and simulation of networked embedded systems. *Proceedings - 2008 Forum on Specification, Verification and Design Languages, Fdl'08, Proc. - Forum Specif., Verif. Des. Lang., Fdl*, 49–54. doi:10.1109/fdl.2008.4641420.

Grimm, C. and Fraunhofer, K.E. (2014). Analog and mixed signal modelling with systemc-ams.

Koch-Hofer, C., Renaudin, M., Thonnart, Y., and Vivet, P. (2007). Asc, a systemc extension for modeling asynchronous systems, and its application to an asynchronous noc. *Proceedings - Nocs 2007: First International Symposium on Networks-on-chip, Proc. First Int. Symp. Netw.-on-chip*, 295–303. doi:10.1109/nocs.2007.12.

Nouidui, T.S., Wetter, M., and Zuo, W. (2014). Functional mock-up unit for co-simulation import in energyplus. *JOURNAL OF BUILDING PERFORMANCE SIMULATION*, 7(3), 192–202. doi:10.1080/19401493.2013.808265.

Organization., I.M. (2013). *MARPOL : Annex VI and NTC 2008 with guidelines for implementation.* International Maritime Organization London, third edition 2013. edition.

Stoermer, C. and Tibba, G. (2014). Powertrain co-simulation using autosar and the functional mockup interface standard. 1. doi:10.1109/DAC.2014.6881372.

Appendix A. NOMENCLATURE

Table A.1. Acronyms

Acronym	Definition
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
SCNSL	SystemC Network Simulation Library
MDT	MAN Diesel & Turbo
ECU	Engine Control Unit
EICU	Engine Interface Control Unit
TIU	Tacho Interface Unit
ACU	Auxiliary Control Unit
CCU	Cylinder Control Unit
SCU	Scavenge Air Unit
MOP	Main operating panel

Chapter 9

Paper B: FMI for Co-Simulation of Embedded Control Software

Declaration of co-authorship

According to the Ministerial Order on Doctoral Degrees, Section 5(4), if a dissertation or parts of it are based on previously published papers which are the result of collaboration, a declaration signed by each of the joint authors including the author of the dissertation, and setting out the amount and character of the author's contribution to the work, must be submitted with the dissertation.
Co-authorship is defined according to the Vancouver guidelines, see http://www.icmje.org/index.html

Regarding the following publication:
Nicolai Pedersen, Tom Bojsen, Jan Madsen, Morten Vejlgaard (2016).
FMI for Co-Simulation of Embedded Control Software

which will be included in the following dissertation:
Co-Simulation of Cyber-Physical System with Distributed Embedded Control

The degree of Nicolai Pedersen's contribution to the publication based on the following scale:

A. has contributed to the collaboration (0-33%).
B. has contributed substantially (34-66%).
C. has to a high degree carried out the work independently (67-100%).

Declaration in each element.	A,B, or C
1. Formulating the scientific idea based on theoretical assumptions to be clarified, including formulation of the question to be answered through analytical work and research plans.	C
2. Planning of experiments and analyses, design of the experimental methods in a way that the questions asked under point 1 can be expected to be answered.	C
3. Involvement in analytical work with respect to the concrete experimental studies and investigations.	C
4. Presentation, interpretation and discussion of the results.	C

Co-authors signature:			
Date	Name	Title	Signature
25/9-2017	Tom Bojsen	BSc Eng	Tom Bojsen
29/9-2017	Jan Madsen	Professor	Jan Madsen
28/9-17	Morten Vejlgaard Larsen	Manager	Morten Vejlgaard Larsen

Dissertation author's signature

Date: 29/9-2017 Signature: Nicolai Pedersen

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

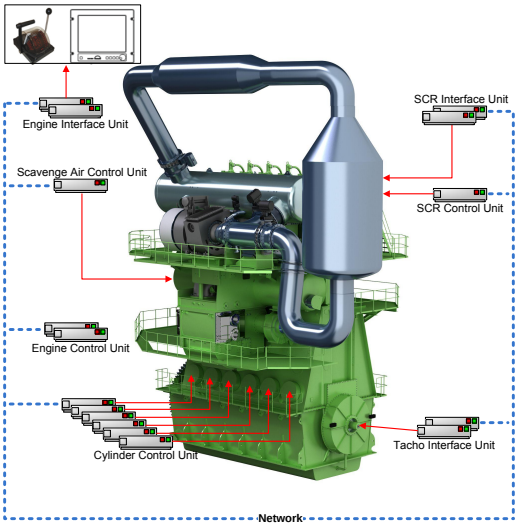

Figure 1. An MAN Diesel & Turbo two-stroke low-speed diesel engine with the SCR and the engine control system illustrated

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

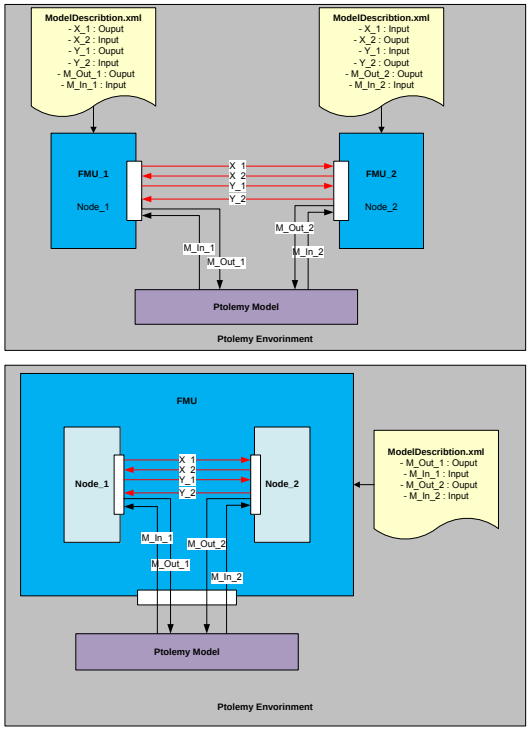

Figure 3. It is possible to change the level of configuration complexity exposed to the user. The top figure shows how each control system node can be packed in an FMU for maximal configuration flexibility. The bottom figure shows how multiple nodes can be packed and configured in a single FMU for a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

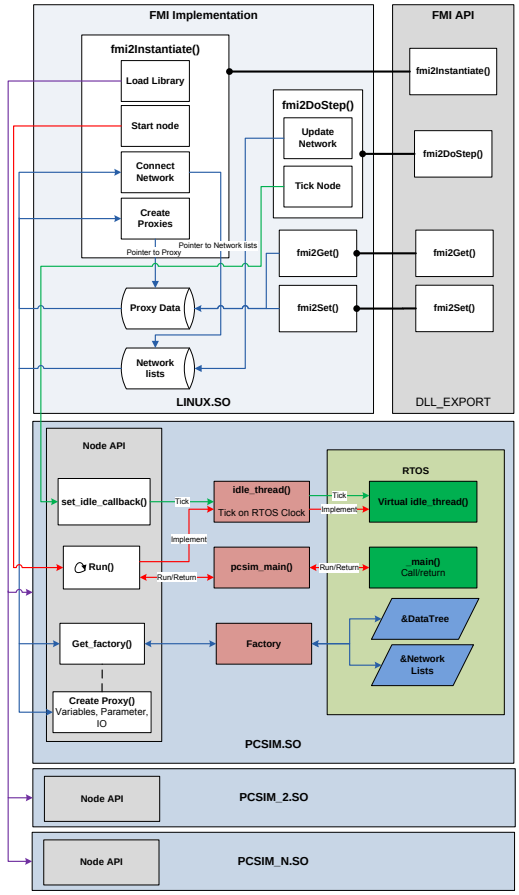

Figure 4. The implementation of FMI on the MAN engine control system

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

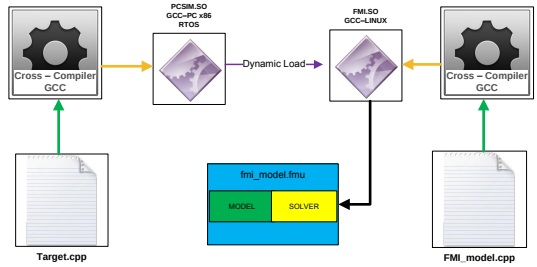

Figure 5. The compiling routine from target to functional mock-up unit.

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

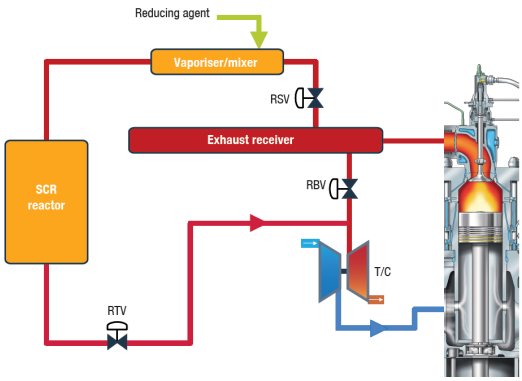

Figure 6. Diagram of the SCR system

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

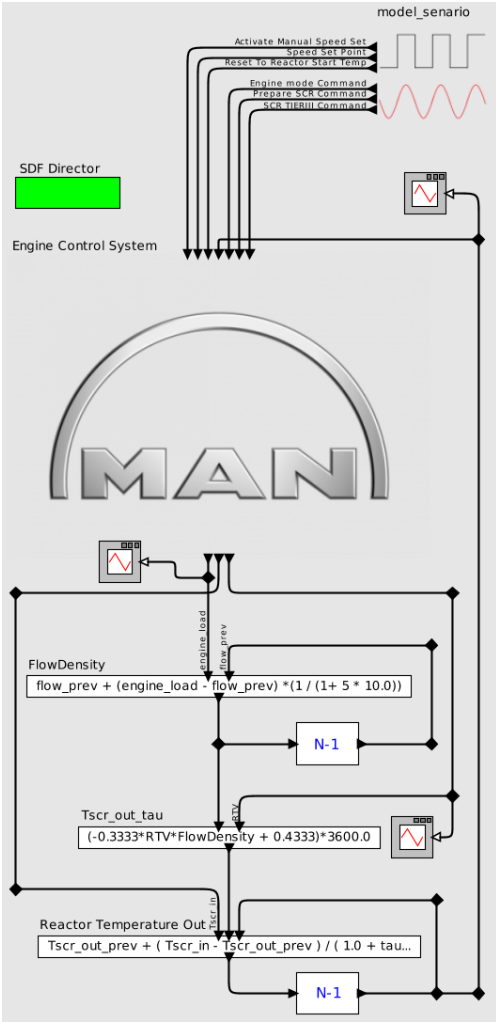

Figure 7. FMU import in Ptolemy II and simple physical model implementation

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

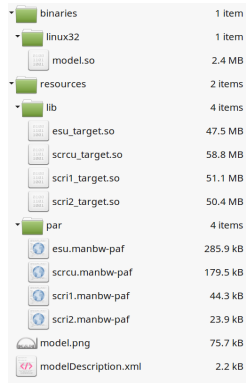

Figure 8. The use-case example of a functional mock-up unit containing the MAN SCR control nodes

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*"

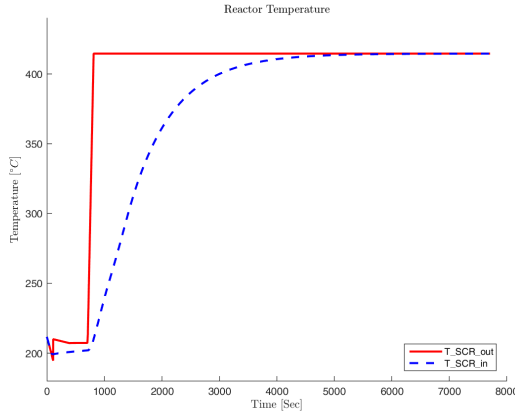

Figure 9. The in- and output temperature of the simulated SCR heating

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.

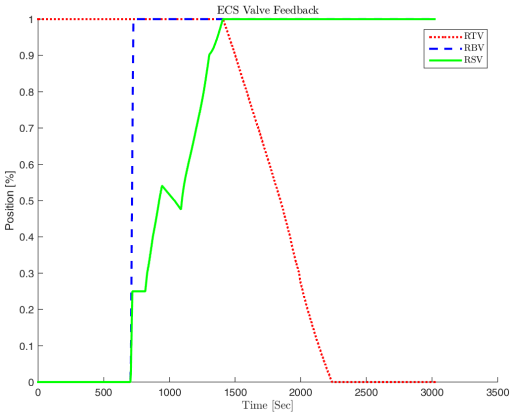

Figure 10. Valve feedback from the SCR simulation

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.

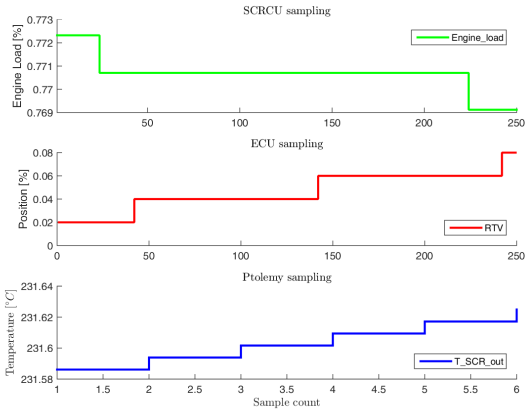

Figure 11. Illustration of the different sub-system sampling frequencies

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.

Chapter 10

Paper C: Co-Simulation of Cyber Physical Systems with HMI for Human In the Loop Investigations

Declaration of co-authorship

According to the Ministerial Order on Doctoral Degrees, Section 5(4), if a dissertation or parts of it are based on previously published papers which are the result of collaboration, a declaration signed by each of the joint authors including the author of the dissertation, and setting out the amount and character of the author's contribution to the work, must be submitted with the dissertation.
Co-authorship is defined according to the Vancouver guidelines, see http://www.icmje.org/index.html

Regarding the following publication:
Nicolai Pedersen, Tom Bojsen, Jan Madsen (2017).
CO-SIMULATION OF CYBER PHYSICAL SYSTEMS WOTH HMI FOR HUMAN IN THE LOOP INVESTIGATIONS

which will be included in the following dissertation:
Co-Simulation of Cyber-Physical System with Distributed Embedded Control

The degree of Nicolai Pedersen's contribution to the publication based on the following scale:

A. has contributed to the collaboration (0-33%).
B. has contributed substantially (34-66%).
C. has to a high degree carried out the work independently (67-100%).

Declaration in each element.	A,B, or C
1. Formulating the scientific idea based on theoretical assumptions to be clarified, including formulation of the question to be answered through analytical work and research plans.	C
2. Planning of experiments and analyses, design of the experimental methods in a way that the questions asked under point 1 can be expected to be answered.	C
3. Involvement in analytical work with respect to the concrete experimental studies and investigations.	C
4. Presentation, interpretation and discussion of the results.	C

Co-authors signature:			
Date	Name	Title	Signature
25/9-2017	Tom Bojsen	BSc Eng	
29/9-2017	Jan Madsen	Professor	

Dissertation author's signature

Date: 29/9-2017 Signature:

CO-SIMULATION OF CYBER PHYSICAL SYSTEMS WITH HMI FOR HUMAN IN THE LOOP INVESTIGATIONS

Nicolai Pedersen
Department of Embedded Systems Engineering
Technical University of Denmark
Richard Petersens Plads, 2800 Kgs. Lyngby , Denmark
nicp@dtu.dk

Tom Bojsen
MAN Diesel & Turbo
Teglholmsgade 35
2450 Copenhagen, Denmark
tom.bojsen@man.eu

Jan Madsen
Department of Embedded Systems Engineering
Technical University of Denmark
Richard Petersens Plads , Building 324
DK-2800 Kgs. Lyngby , Denmark
jama@dtu.dk

ABSTRACT

The development of safety critical Cyber-Physical Systems (CPS) is highly dependent on human interaction and cognitive assessment. Despite this dependency, the human in the loop is seldom an integrated part of CPS development or tool chain. In this paper we propose a hybrid co-simulation environment, where hardware, software and models can be interconnected, making it possible to connect a human machine interface with a software representation of a control system and thermodynamic engine models. Scenarios that require user interaction can be formulated and propagate from engine physics through the control system to the engine operator. The environment makes it possible to investigate human interaction during system development and gain more quantitative and evidence based data for designing safety critical CPS with human-machine interaction.

Keywords: Co-simulation, HMI, FMI, Cyber-physical system, Embedded systems.

1 INTRODUCTION

Most Human Machine Interfaces (HMI) are not representative before connected to the actual hardware, making their development delayed compared to the system development. Furthermore, real hardware and test setups are often limited resources, especially when dealing with large or expensive equipment. The combination of limited resources and delayed development makes it difficult to thoroughly investigate human interaction and, therefore, to design systems tolerant towards user error and misuse. This paper proposes an environment, where engineers can choose to connect both hardware, software and models in a hybrid co-simulation. The environment makes it possible to connect the HMI to a software representation of the control system before it is released to the hardware platform, in which case HMI and control system devel-

SpringSim-TMS/DEVS 2017, April 23-26, Virginia Beach, VA, USA
©2017 Society for Modeling & Simulation International (SCS)

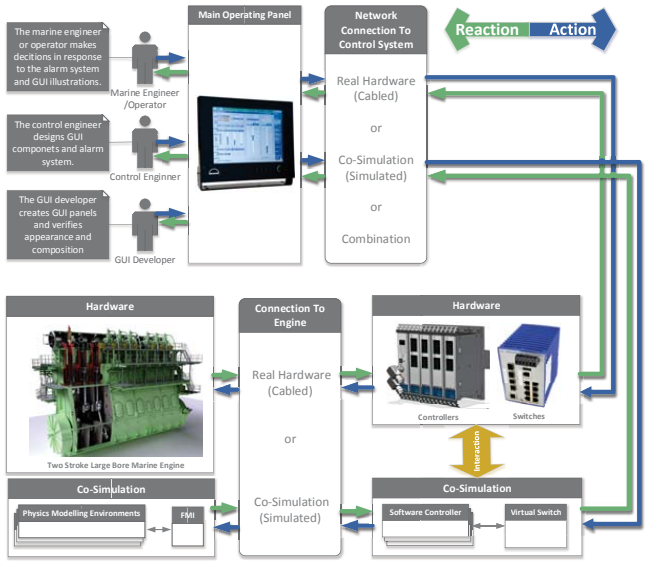

Figure 1: Co-simulation connection options.

opment becomes more concurrent. Furthermore, the control system software can be combined with physics models through the Functional Mockup Interface (FMI) co-simulation standard. It is possible to investigate the human behavior during system development by tracking the user interaction. The hybrid co-simulation is based on the work being done at MAN Diesel & Turbo, where the physics of a large bore two-stroke marine engine is connected with the surrounding distributed control system and the human machine interface.

The HMI is shown in Figure 1 together with the three stakeholders. The marine engineer/operator is the main user of the HMI interacting with the system according to bridge commands and in response to information from the operating panel, alarm system, illustrations, etc. The control engineer developing control algorithms for the control system is also a stakeholder.

The control engineer is responsible for developing the correct interaction possibilities and creates alarms for new components. The work of the control engineer is, therefore, dependent on an understanding of the cognitive assessment of the operator. Lastly, the Graphical User Interface (GUI) developer is responsible for the user experience and the graphical representation of the system. All stakeholders are interconnected and the communication and joint understanding between them are important. If for example the operator needs to respond to an alarm, it is equally important that the alarm text, formulated by the control engineer, is explanatory and that the user interface, made by the GUI developer, sufficiently attracts the attention of the operator.

The hardware platform for the HMI is called the Main Operating Panel (MOP), it is a touch-screen interface connected to the engine control system through the Ethernet. The switches connecting the MOP with the controllers are in the co-simulation environment replaced with a SW implementation, which is termed a virtual switch. The virtual switch is responsible for directing all network traffic between both simulated controllers and real HW. To simulate an embedded controller is not trivial, how the Real Time Operating System (RTOS) has been adapted to enable this will be described in Section 3. HW controllers and SW controllers can be connected by introducing an additional "proxy" controller with the purpose to redirect IO data through the virtual switch, this will not be covered in the present paper. Finally, the control system can be connected to either the physical engine or to the engine models. The latter is achieved by implementing the co-simulation version of the FMI standard (Blockwitz et al. 2012).

One of the aims with the proposed environment is to track the human interaction, when the user is presented with specific scenarios. The interaction provides significant information to system developers regarding operator behavior and may also be useful, when educating operators and marine engineers.

A comprehensive survey of the current state of human in the loop CPS efforts has been performed in (Nunes et al. 2015). In (Gopalakrishna et al. 2017), dealing with machine-learning for human in the loop CPS, a new way of determining the accuracy of an output based on a relevance score taking into account the variability and bios of the human perception. Similar for most research in Human CPS (Gopalakrishna et al. 2017, Nunes et al. 2015, Lieber and Fass 2011), is the acknowledgement of multidisciplinary technical challenges and argument that the traditional development process, where human impact is not an integrated part of the process, will not be sufficient for future products. Multiple co-simulation frameworks and tools for investigating CPS are available. Commercial tools like MATLAB, Dymola and GT-Suite are widely known but limited in their interconectivity. The Ptolemy project from Berkeley (Eker et al. 2003, Awais et al. 2013) and INTO-CPS (Larsen et al. 2016) aims to create a streamlined tool chain for the multidisciplinary development of CPS. More specific projects like (Zhang et al. 2013) and (Zeller et al. 2010) work with embedded systems using tools such as SystemC to produce co-simulations that can supplement HIL testing and result in faster prototyping. An advanced co-simulation environment for development of HMI including the human factor was presented in (Sixto et al. 2015). Here a new HMI aiding efficient human behaviour in electrical vehicles was developed through a set of clinical user experiments, with good results. The environment comprises a number of high-end automotive tool some connected by standard interfaces others adapted to the environment, with no information regarding how the tool and simulation were connected. Significant work has been aimed at co-simulation of different aspects of CPS development, however, very limited research has been put into how to connect the human in the loop within a co-simulation environment.

This paper starts with an introduction of the HMI in Section 2 followed by a description of how the control system has been adapted to enable co-simulation in Section 3. Section 4 describes the virtual switch, which has been developed for connecting the HMI and the simulated control system. The FMI standard used to connect the control system with thermodynamic engine models is explained in Section 5. Finally, the thermodynamic models used in the proof-of-concept simulation are explained in Section 6. A simulation scenario and the results following from it are presented in Section 7.

2 MAIN OPERATING PANEL

The Main Operating Panel (MOP) is the main HMI for engineers operating the engine. The MOP is a marine approved and certified PC with a touch screen interface located on the engine control room panel. From the MOP the engineer can carry out engine commands, adjust engine parameters, select running mode and observe the status of the control system.

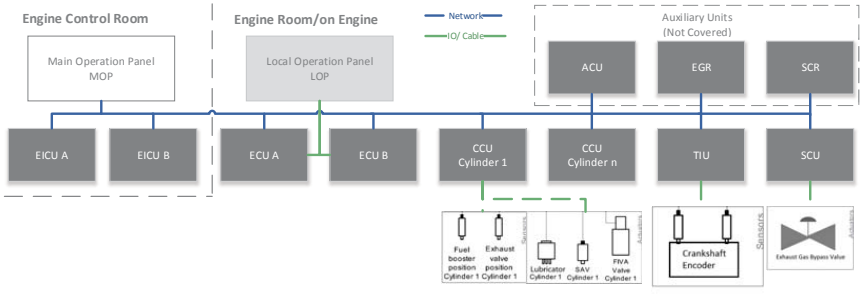

Figure 2: Engine control system communication diagram.

The communication between the MOP and engine control system controllers is Ethernet based. A MAN Ethernet protocol driver connects the MOP with the Engine Interface Control Unit (EICU) as seen in Figure 2. The EICU is connected to the rest of the distributed system through the Engine Control Unit (ECU), both EICU and ECU units are redundant for safety reasons. If for some reason the MOP is unavailable, the engine can also be operated directly from a local operating panel located on the engine. Auxiliary systems such as blowers, hydraulic pumps, exhaust gas recirculation and selective catalytic reduction units will not be discussed in this paper. Every cylinder has a dedicated Cylinder Control Unit (CCU), which controls the actuators responsible for fuel injection, valve opening/closing, lubrication, etc., according to sensor feedback. Timing of the engine is governed by the ECU, which receives the crankshaft position by the Tacho Interface Unit (TIU). The results presented in this paper are based on an example, where the operator manipulates the exhaust gas bypass valve through the Scavenge air Control Unit (SCU).

3 SOFTWARE CONTROLLER

Embedded controllers installed on MAN Diesel & Turbo engines are multipurpose controllers, this implies that they are hardware-wise identical only the software executed on the target determines the specific control objective. The controllers interface with sensors and other computational units through network and cabling and interact with the system through actuators. The controller contains a CPU module with an FPGA-based embedded system running a RTOS. Our strategy for doing software in the loop simulations of the embedded controllers is to replace the embedded board support package (BSP) with an x86 platform version and to rewrite part of the functionality. This approach is partly discussed in (Pedersen et al. 2016). As a consequence the simulation does not include the behavior of the embedded processor instead it will include the target code executed on an x86 platform. Further abstractions and deviations will be presented below.

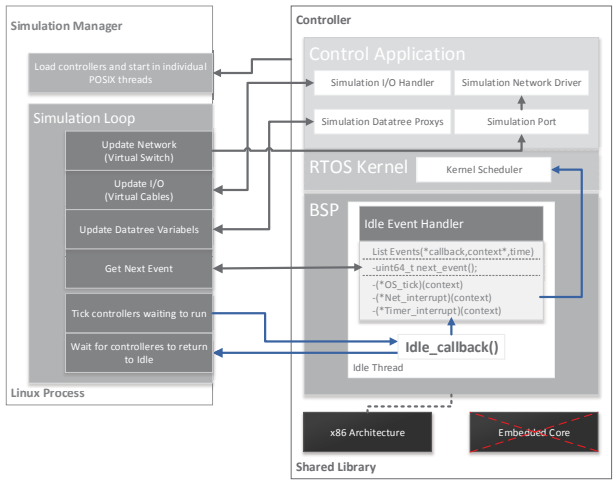

Figure 3: The software model of the embedded system and connection to the simulation manager.

3.1 System Clock

By modeling the controller software with an x86 BSP it is possible to execute the model on a regular developer PC, however, controlling the execution is not possible. To gain control of the execution requires access to the clock of the embedded system. Overwriting the BSP_idle_thread of the RTOS and introducing a blocking idle_callback function to the simulation manager enable us to lock execution every time the

system is in idle. In this way a hook to the system clock is provided. When the controller reaches idle, the execution is stopped and started again by releasing the callback that resembles a clock tick. To get a concept of time we utilize that the OS execute every millisecond. By assuming that the system has sufficient computational power to return to idle between every tick, we know that a millisecond has passed between each OS tick. This guaranties a common perception of time across multiple controllers and a temporal execution of the model.

Remark. *Note, that we are basically assuming unlimited processing power so all tasks will finish and never be interrupted before returning to idle. This could cause the simulation results to deviate from a real stochastic execution. However, it ensures a deterministic simulation which is important during control algorithm development and regression testing. Our system is currently dimensioned with sufficient computational power for this to have no significant consequences. The purpose of the co-simulation is not to replace HIL testing but to aid engineers before doing hardware test.*

3.2 Idle Event Scheduler

Overwriting the idle thread not only enables us to control the system clock, it also makes it possible to introduce simulation functionality on the controller. An event scheduler is introduced in the idle thread to execute external events not scheduled by the OS itself. Events consist of a call-back function to the interrupt that needs to be executed, and the execution time for when the event needs to occur. In this way we are able to schedule events such as network interrupt, IO observer timers and other high precision timers with a resolution down to one microsecond on individual controllers. The event scheduler communicates with the simulation manager and tells when the next event is due. This makes it possible for the manager to orchestrate the simulation and maintain a common perception of time across the entire system.

3.3 Variables and IOs

On the target application all variables, parameters and IOs are organized in a component-oriented data tree structure with unique IDs. We can create proxies for variables, parameters and IO channels by using a factory method design and by rewriting part of the application functionality. This provides a get/set functionality that will effect the source on the specific controller. The IO cabling is achieved by the simulation manager by connecting virtual cables using the module and port typology of the embedded target. Communication between controller input and output is done on a microampere level, simulating a real cable and activating the conversion layers of the software.

3.4 Simulation Manager

The simulation manager illustrated in Figure 3 is responsible for orchestration of execution and data exchange. Each controller is compiled to a shared library and dynamically loaded by the manager, where it is assigned an individual thread. The simulation is a loop, where the network is updated through the virtual switch presented in Section 4. The factory method enables the manager to update data tree variables and IOs. The manager can access the idle event handler and get information about when a specific controller needs to execute. Controllers that need to execute can be stated by releasing the semaphore, blocking their idle callback. The controllers run in parallel and return to the manager, when returning to idle. The global time is updated on the entire system and loop repeated.

A simulation without hardware is allowed to run as fast as possible but the simulation has to run in real-time if hardware is connected. This is achieved by letting the process sleep after execution for the remaining

amount of time of the simulation time step.

Remark. *Note, that simulating in real-time requires all models in the system to be able to run in real-time. Highly complex physics models might not be able to fulfill this requirement. All models presented in this paper are real-time compliant with the desktop hardware available to engineers at MAN Diesel & Turbo.*

4 VIRTUAL SWITCH

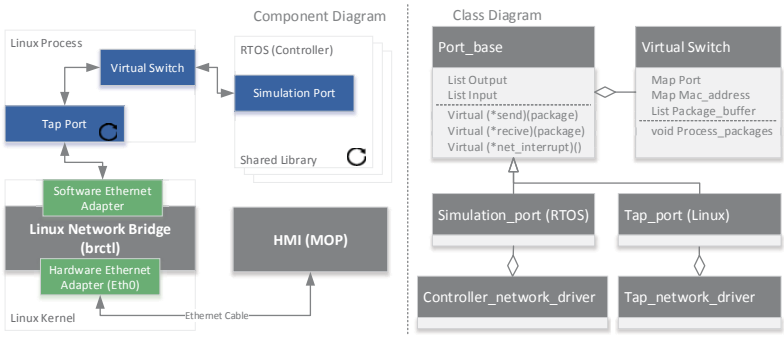

Figure 4: Illustration of the virtual switch connecting the co-simulation with the HMI.

The virtual switch is the component that enables the co-simulation to connect with the MOP. The purpose of the switch is to mimic a real switch. It runs in the simulation manager process and redistributes network packages to ports according to a mac-address table. The switch has two types of ports, a simulation port for the simulated controllers and a tap port for connected hardware. The components are illustrated in Figure 4.

4.1 Simulation Port

The simulation port instance contains input and output lists for network packages to and from the port. The port is instantiated on the controller and a pointer is given to the switch running in the simulation manager. This process is a Linux process. Data must be transferable between the manager and the shared library RTOS controller. This is achieved by implementing three port delegates, these connect the RTOS network driver and the switch described in Table 1.

Table 1: Port delegates

void(*send)(package)	A method called from the RTOS network driver that moves a network package from the controller to the output list of the port. The switch can then access this package from within the Linux process and place it in its own package buffer.
void(*net_interrupt)()	A method called from the switch that simulates a network interrupt on the controller. The interrupt is not allowed to execute, as it comes from a Linux context. The interrupt schedules a high priority task on the RTOS. The manager then allows the controller to run and the network package can be received within the correct context.

void(*receive)(&package)	A method called from the RTOS network driver that moves a network package from the input list of the port to the controller.

The switch is responsible for receiving packages from the port-output lists and moves them to the input lists of the correct port according to the destination mac-address of the package Ethernet-header. The input and output lists are protected by a semaphore to ensure that the lists are not accessed simultaneously.

4.2 Tap Port

The tap port is also an instance of the port class with the same components and functionality as the simulation port described in Table 1. It implements the delegates to link the tap network driver instead of the controller network driver. The *net_interrupt*() is allowed to execute directly, since the tap port is within the Linux context. The tap network driver connects the port to a tap interface, a software network adapter that only exists in the Linux kernel. The interface works as a regular network adapter, where the kernel exchanges full Ethernet frames from and to the network driver instead of a regular wire. A Linux software network bridge connects the tap interface with the physical Ethernet adapter that can be connected to the MOP by wire as seen in Figure 4. Multiple tap interfaces can be created and ports instantiated making it possible to connect the virtual switch to both the network of simulated controllers and the actual hardware.

Remark. *It is important to notice that connecting an Ethernet interface destroys the determinism of the co-simulation. This means that we can no longer ensure simulation reproducibility and that the system is no longer appropriate for regression tests, etc.*

5 FMI

The Functional Mockup Interface (FMI) for co-simulation provides a standardized way of doing co-simulation. In the FMI each subsystem model is solved independently with individual solvers and data exchange occurring in-between calculations at so-called discrete communication points. A subsystem that implements FMI is called a Functional Mockup Unit (FMU) and it is assembled as a zip-file containing all the necessary components required to utilize the FMU. This paper will not explain the FMI standard in further detail, see instead (Blockwitz et al. 2012, Pedersen et al. 2015). In (Pedersen et al. 2016) it is described how our simulation manager complies with the FMI standard.

The simulation presented in this paper contains two FMUs. Firstly, the engine control system co-simulation is wrapped as an FMU. It contains the simulation manager and 10 embedded controllers: EICU A/B, ECU A/B, TIU, CCU 1-4 and the SCU as presented in Figure 2. Secondly, the thermodynamic engine model presented in the following section is also an FMU. The engine model is developed in an internal C++ modeling tool that has been made compliant with the FMI standard.

6 THERMODYNAMIC ENGINE MODEL

The engine model is based on the MAN Diesel & Turbo 4T50ME-X test engine, a two-stroke compression-ignition engine. It has four cylinders of 50 cm bore and a stroke of 2.2 m, and it generates 7,080 kW at 123 rpm. The model is an air-path model focused on the mass flows and pressures through the system. The model has previously been published and validated in (Alegret et al. 2015) and is based on research done by (Wahlstrom and Eriksson 2011, Hansen et al. 2013). In (Alegret et al. 2015) the model also include an exhaust gas recirculation system (EGR). The EGR is not activated in the scenario presented here and will therefore not be covered.

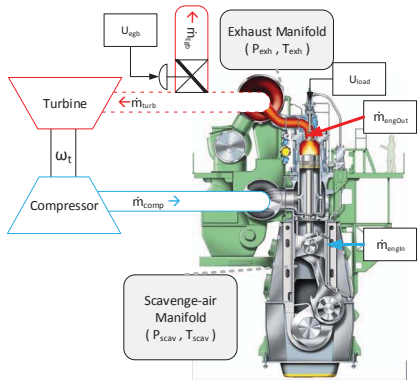

Figure 5: Engine air-path model.

The components with dominating dynamics are seen in Figure 5 and described below. The turbocharger consists of a turbine and compressor connected by a common shaft. The turbine is driven by the exhaust gas from the engine and the generated power is transferred through the shaft to the compressor. The purpose of the turbocharger is to balance the mass flow of air to the scavenge air manifold. The mass flows $\dot{m}_{comp}$ and $\dot{m}_{trub}$ are described in maps provided by the turbocharger producer. The maps show the mass flows as a function of pressure ratio and turbocharger velocity. The power delivered from the turbine and compressor is used to setup a state equation expressing the turbocharger velocity ω_t. Scavenge air and exhaust gas manifolds are modeled as control-volumes based on the ideal-gas law and conversion of mass with state equations describing the pressure P_{scav} and P_{exh}. The engine is modeled as flow through a restriction ($\dot{m}_{engIn}, \dot{m}_{engOut}$) with a cylinder temperature T_{exh} based on the Seiliger cycle. The Exhaust Gas Bypass (EGB) is the only component not presented in (Alegret et al. 2015). The EGB is redirecting the flow $\dot{m}_{egb}$ from the turbine inlet and is modeled as flow through a restriction.

$$\dot{m}_{egb} = U_{egb}\, A_{egb} \frac{p_{exh}}{\sqrt{R_e T_{exh}}} \sqrt{\frac{2\gamma_e}{\gamma_e - 1}\left[\frac{p_{egb}}{p_{exh}}^{\frac{2}{\gamma_e}} - \frac{p_{egb}}{p_{exh}}^{\frac{\gamma_e+1}{\gamma_e}}\right]}, \tag{1}$$

where A_{egb} is the maximum EGB-valve orifice and U_{egb} the valve position, γ_e is the ratio of specific heat, R_e is the exhaust-gas constant and P_{egb} is the back pressure from the subsequent system. The purpose of the EGB is both to control the turbocharger equivalent area and to increase the energy for downstream systems such as waste heat recovery.

The input to the model is EGB valve setpoint U_{egb} and engine load U_{load}, a non-dimensional power defined as a percentage of the maximal power available for the specific engine. The remaining components are assumed to have little impact on the air flow dynamics and are, therefore, disregarded. All cooling in the system is assumed ideal meaning that the temperature of the gas leaving and entering a manifold is assumed to have the exact same temperature. Heat transfer is also neglected, meaning that there is no temperature drop, e.g. from the cylinder to the exhaust gas receiver.

7 SIMULATION & RESULTS

The aim of this project and the presented simulation scenario are to engage the human operator and track the interaction with the simulation in a potentially hazardous situation.

The presented proof-of-concept scenario focuses on EGB control. When the EGB-valve is closed, the turbine receives the full power from the exhaust gas. This increases the angular velocity of the common shaft and, thereby, the compressor, causing the scavenge-air pressure (P_{scav}) to raise. This is opposite to when the

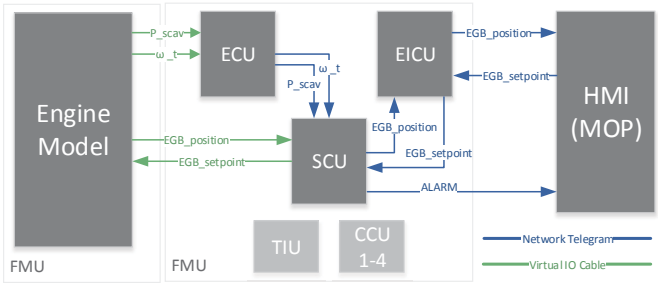

Figure 6: The two FMUs and the MOP are illustrated with relevant cable and network connections.

EGB-valve is opened, where the flow to the turbine decreases and P_{scav} drops. In certain situations both scenarios can be very undesirable. If the turbine velocity increases too much the turbine may be destroyed or even explode with extreme danger to the crew. If P_{scav} drops significantly the engine may suffocate and operation has to be stopped. These are of course extreme situations, where the safety-critical system has not been working. With working safety systems these scenarios would cause the alarm system to notify the operator and if no action is taken send a slow-down or shut-down command bypassing the operator. Slow-down and shut-down commands are still very undesirable measures that severely limits the maneuverability of the vessel. In this scenario the signals are sent between the units as seen in Figure 6. The scavenge air pressure P_{scav} and the turbocharger velocity ω_t will be provided by the engine model to the engine control unit through simulated analog cables. The control setpoint of the EGB valve $EGB_{setpoint}$ and the actual EGB-valve position $EGB_{position}$ are connected directly to the scavenge air control unit. Within the control system data is transferred by network telegrams between the controllers and the MOP. The tacho interface unit and the four cylinder control units shown in Figure 6 are required to simulate the system properly. They are connected to the MOP and running, however, they are not relevant for the presented scenario and will not be covered. The control signal representing engine load U_{load} is normally a command from the bridge, here it will be set at 40% load and provided internally in the engine model.

The simulation can be seen in Figure 7. The pressures start out stable at around 2.5×10^5 Pa, at 320 seconds the EGB-valve is opened by the engine model. The turbine velocity then drops dramatically and with that both P_{scav} and P_{exh}, when P_{scav} drops below the alarm limit of 1.75×10^5 Pa an alarm will be triggered on the SCU and displayed on the MOP. The operator will see an alarm appear on the top bar and the alarm list panel, which will look like the top figure of Figure 8. The simulation manager will track the alarm and record how and when the operator responds. The operator will have to navigate to the Scavenge Air panel as seen in the bottom figure of Figure 8. Here he has the option of either changing the EGB-valve setpoint to 0% or set the EGB-controller in automatic mode which will likewise close the valve. Both actions will cause an increased exhaust gas flow to the turbine and P_{scav} will return to a stable level. In Figure 7 we see how a MOP-command has ordered the valve to close at 750 seconds causing ω_t to increase and the pressure returning to a stable level. The effects of closing the valve are delayed due to the dynamics of the system.

This is a simple example showing how it is possible to create scenarios in a valid thermodynamic model that interacts with the complex control system connected to the HMI. Additional scenarios could be formulated and information regarding interacting tracked.

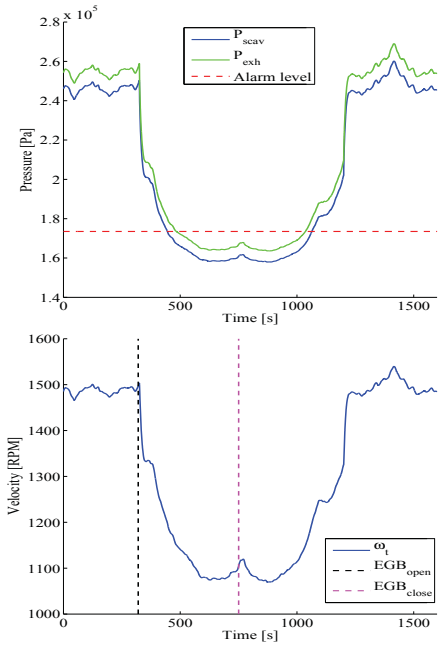

Figure 7: Plot of pressures and turbocharger velocity during the simulation scenario.

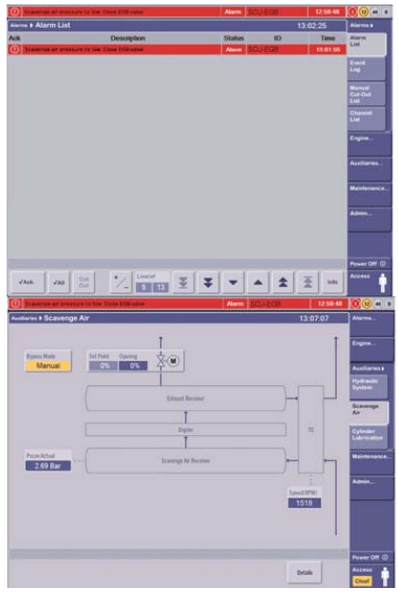

Figure 8: Top: Alarm system panel, Bottom: Scavenge Air panel.

8 DISCUSSION

In this section we will discuss the lessons learned from the experiment presented. To properly incorporate the human in the loop, we believe that investigation of human interaction has to be an integrated part of the CPS development. The traditional way of model based development, where human investigations are not taking into account before the HIL stage, could be optimized using the hybrid co-simulation presented here. With the hybrid co-simulation, human interaction can be investigated at any point of development.

From a survey of current research, it is clear that the multi-disciplinary nature of CPS development is the main challenge. We believe that the solution to this is standardized interfaces for connecting the tools each discipline prefer. The FMI standard plays a major part in this solution. Having a streamlined tool-chain optimal for every engineering discipline, and with a convenient transition between every stage of development, is very seldom and will often force companies to commit to a single tool provider. With the FMI standard, every tool and simulation environment would, in principle, be connectable.

The main advantage of the FMI standard is its diversity and flexibility. FMI is based on C-code making it platform independent and require nothing more then a C-compiler. Subsystem information is described in a simple manner in an XML model description, making the interconnection configuration between subsystems easy to manage. FMI provides an application interface, with a state machine, that needs to be implemented. This state machine ensure that each subsystem is simulated in a similar fashion and that execution and communication within the system is temporally correct. Even though you are forced to implement the application interface, it is only the function calls FMI require, how they are implemented is completely free. Implementing the standard to a custom simulation require some effort. This process is, however, well documented and exemplified in (QTronic , Widl et al. 2013), making the task manageable. This level of flexibility of cause come with some constrains. Especially, the way of exchanging data between subsystems

is limited. The only data types that can be exchanges are "Real","Integer","Boolean" and "String". There is no possibility to exchange e.g. arrays or any advanced data-objects, which can be inconvenient. In our case when adapting our SIL simulation environment, most of our data-types were fix-point types and similar. This required us to create a complete conversion layer between FMI data-types and internal data-types. Furthermore, the data-exchange is based on a Get/Set functionality, when the system and connection amount grow this become a significant execution overhead. In resent years FMI has become a widely accepted standard with multiple applications in automotive, energy systems, HVAC and more. The standard was initially developed for the automotive industry, an industry that MAN Diesel & Turbo share many similarities with including many of the same tools. 95 tools are currently supported the FMI standard, many of which departments in our company currently use and could be interfaced to in the future. Conclusively the FMI standard provided the flexibility we needed to co-simulate our custom environments and opens up for interconnection with a vast amount of tools already in our organization. The implementation of the standard required work, but once done, the interface has proven stable and shown more possibilities then first anticipated.

9 CONCLUSION

This paper presented a way of connecting a human machine interface with a software model of an embedded control system and thermodynamic models in a hybrid co-simulation. The real time operating system of the embedded target software was compiled to an x86 architecture to enable execution on a developer PC. The idle thread of the board support package was adapted to create a hook for the embedded system clock, making it possible to orchestrate a temporal execution across multiple controllers. An event scheduler was introduced in the idle thread simulating higher resolution events like network interrupts. The connection to the human machine interface is achieved by creating a virtual switch that connects the network ports of the software controllers with a software Ethernet interface. This interface is connected to a Linux Ethernet bridge communicating with the HMI. Engine dynamics are simulated in a separate tool with its own solver, which is made possible by the FMI co-simulation standard. A scenario requiring user action was formulated in the advanced thermodynamic model and propagated through the control system software to an operator, who interacted with the human machine interface.

Human error and misuse are difficult to guard against but an understanding of the cognitive assessment of an operator can be very beneficial. A hybrid co-simulation environment as the one presented can provide data otherwise hard to obtain, when building systems that take human interaction into account. Another way of guarding a system against human error is proper training. At the MAN PrimeServ Academy Copenhagen, marine engineers are educated in using the system. However, testing resources are very limited due to the immense system cost. With the hybrid co-simulation environment the cost would be reduced to the cost of the PC used by the students. The environment even provides much more sophisticated thermodynamic models than the HIL models currently used in the academy. Furthermore, partners around the world educating marine engineers could benefit from the environment as an education tool, which in return would benefit MAN Diesel & Turbo with better operated engines.

REFERENCES

Alegret, G., X. Llamas, M. Vejlgaard-Laursen, and L. Eriksson. 2015. “Modeling of a large marine two-stroke diesel engine with cylinder bypass valve and EGR system”. *IFAC Proceedings Volumes (IFAC-PapersOnline)* vol. 48 (16), pp. 273–278.

Awais, M. U., P. Palensky, W. Mueller, E. Widl, and A. Elsheikh. 2013, 11. “Distributed hybrid simulation using the HLA and the Functional Mock-up Interface”. In *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*, pp. 7564–7569, IEEE.

Blockwitz, T., M. Otter, J. Akesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel. 2012, 11. “Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models”. In *8th International Modelica Conference 2011*, pp. 173–184.

Eker, J., J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. 2003. "Taming heterogeneity - The ptolemy approach". *Proceedings of the IEEE* vol. 91 (1), pp. 127–143.

Gopalakrishna, A. K., T. Ozcelebi, J. J. Lukkien, and A. Liotta. 2017, 2. "Relevance in cyber-physical systems with humans in the loop". *Concurrency and Computation: Practice and Experience* vol. 29 (3), pp. e3827.

Hansen, J. M., C.-G. Zander, N. Pedersen, M. Blanke, and M. Vejlgaard-Laursen. 2013. "Modelling for Control of Exhaust Gas Recirculation on Large Diesel Engines". *IFAC Proceedings Volumes* vol. 46 (33), pp. 380–385.

Larsen, P. G., J. Fitzgerald, J. Woodcock, P. Fritzson, J. Brauer, C. Kleijn, T. Lecomte, M. Pfeil, O. Green, S. Basagiannis, and A. Sadovykh. 2016, 4. "Integrated tool chain for model-based design of Cyber-Physical Systems: The INTO-CPS project". In *2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)*, pp. 1–6, IEEE.

Lieber, R., and D. Fass. 2011. "Human Systems Integration Design: Which Generalized Rationale?". In *Human Centered Design: Second International Conference, HCD 2011, Held as Part of HCI International 2011, Orlando, FL, USA, July 9-14, 2011. Proceedings*, edited by M. Kurosu, pp. 101–109. Berlin, Heidelberg, Springer Berlin Heidelberg.

Nunes, D. S., P. Zhang, and J. Sá Silva. 2015. "A Survey on Human-in-the-Loop Applications Towards an Internet of All". *IEEE COMMUNICATION SURVEYS & TUTORIALS* vol. 17 (2).

Pedersen, N., T. Bojsen, J. Madsen, and M. Vejlgaard-Laursen. 2016. "FMI for Co-Simulation of Embedded Control Software". In *The First Japanese Modelica Conferences, May 23-24, Tokyo, Japan*, Number 124, pp. 70–77. MAN Diesel & Turbo, Copenhagen, Denmark, Linköping University Electronic Press, Linköpings universitet.

Pedersen, N., J. Madsen, and M. Vejlgaard-Laursen. 2015. "Co-Simulation of Distributed Engine Control System and Network Model using FMI and SCNSL". *10th IFAC Conference on Manoeuvring and Control of Marine Craft MCMC 2015* vol. 48 (16), pp. 261–266.

QTronic. "FMU SDK".

Sixto, V., P. Lopez, F. Sanchez, S. Jones, E. Kural, A. F. Parrilla, and F. Le Rhun. 2015. "Advanced co-simulation HMI environment for fully Electric Vehicles". In *2014 IEEE International Electric Vehicle Conference, IEVC 2014*.

Wahlstrom, J., and L. Eriksson. 2011. "Modelling diesel engines with a variable-geometry turbocharger and exhaust gas recirculation by optimization of model parameters for capturing non-linear system dynamics". *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering* vol. 225 (7), pp. 960–986.

Widl, E., W. Muller, A. Elsheikh, M. Hortenhuber, and P. Palensky. 2013. "The FMI++ library: A high-level utility package for FMI for model exchange". *2013 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems, MSCPES 2013*.

Zeller, M., G. Weiss, D. Eilers, and R. Knorr. 2010. "Co-simulation of self-adaptive automotive embedded systems". *Proceedings - IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, EUC 2010*, pp. 73–80.

Zhang, Z., E. Eyisi, X. Koutsoukos, J. Porter, G. Karsai, and J. Sztipanovits. 2013. "Co-simulation framework for design of time-triggered cyber physical systems". *2013 ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2013*, pp. 119–128.

AUTHOR BIOGRAPHIES

NICOLAI PEDERSEN is an industrial Ph.D. student at the Department of Applied Mathematics and Computer Science at the Technical University of Denmark. His Ph.D. is in collaboration with MAN Diesel & Turbo in the department of Basic Software Platform. He holds a M.Sc. in Electrical Engineering and his research interests lie in embedded systems and optimization of development process through co-simulation.

TOM BOJSEN is a software engineer at MAN Diesel & Turbo. He holds a Bachelor in Electronic Engineering form the Technical University of Denmark. Tom has more than 20 years of experience in embedded software with special interest in network engineering.

JAN MADSEN is Full Professor in Computer-Based Systems at Department of Applied Mathematics and Computer Science (DTU Compute), Technical University of Denmark (DTU). His research interests include methods and tools for systems engineering of computing systems. Present research covers embedded systems, wireless sensor networks (Internet-of-Things), microfluidic biochips (Lab-on-Chip) and synthetic biology.

Chapter 11

Paper D: Distributed Co-Simulation of Embedded Control Software with Exhaust Gas Recirculation Water Handling System using INTO-CPS

Declaration of co-authorship

According to the Ministerial Order on Doctoral Degrees, Section 5(4), if a dissertation or parts of it are based on previously published papers which are the result of collaboration, a declaration signed by each of the joint authors including the author of the dissertation, and setting out the amount and character of the author's contribution to the work, must be submitted with the dissertation.
Co-authorship is defined according to the Vancouver guidelines, see http://www.icmje.org/index.html

Regarding the following publication:
Nicolai Pedersen, Kenneth Lausdahl, Enrique Vidal Sanchez, Peter Gorm Larsen, Jan Madsen (2017).
Distributed Co-Simulation of Embedded Control Software with Exhaust Gas Recirculation Water Handling System using INTO-CPS

which will be included in the following dissertation:
Co-Simulation of Cyber-Physical System with Distributed Embedded Control

The degree of Nicolai Pedersen's contribution to the publication based on the following scale:

A. has contributed to the collaboration (0-33%).
B. has contributed substantially (34-66%).
C. has to a high degree carried out the work independently (67-100%).

Declaration in each element.	A,B, or C
1. Formulating the scientific idea based on theoretical assumptions to be clarified, including formulation of the question to be answered through analytical work and research plans.	C
2. Planning of experiments and analyses, design of the experimental methods in a way that the questions asked under point 1 can be expected to be answered.	B
3. Involvement in analytical work with respect to the concrete experimental studies and investigations.	B
4. Presentation, interpretation and discussion of the results.	C

Co-authors signature:			
Date	Name	Title	Signature
21/9-2017	Kenneth Lausdahl	PostDoc	
29/9-2017	Enrique Vidal Sanchez	Ph.D EE	EVidal
21/9-2017	Peter Gorm Larsen	Professor	
29/9-2017	Jan Madsen	Professor	

Dissertation author's signature

Date: 29/9-2017 Signature:

Distributed Co-Simulation of Embedded Control Software with Exhaust Gas Recirculation Water Handling System using INTO-CPS

Nicolai Pedersen[1,3], Kenneth Lausdahl[2], Enrique Vidal Sanchez[1],
Peter Gorm Larsen[2] and Jan Madsen[3]

[1]*MAN Diesel & Turbo, Teglholmsgade 41, 2450 Kbenhavn SV, Denmark*

[2]*Department of Engineering, Aarhus University, Finlandsgade 22, Aarhus N, Denmark*

[3]*Embedded Systems Engineering, Technical University of Denmark, Anker Engelunds Vej 1, Kgs. Lyngby, Denmark*

{*nicp, jama*}*@dtu.dk,* {*lausdahl, pgl*}*@eng.au.dk,* {*nicolai.pedersen, enrique.sanchez*}*@man.eu*

Keywords: INTO-CPS, Cyber-Physical-Systems, Co-Simulation, Parallel Simulation, Distributed Simulation, Embedded Control System, Exhaust Gas Recirculation.

Abstract: Engineering complex Cyber-Physical Systems, such as emission reduction control systems for large two-stroke engines, require advanced modelling of both the cyber and physical aspects. Different tools are specialised for each of these domains and a combination of tools validating different properties is often desirable. However, it is non-trivial to be able to combine such different models of different constituent elements. In order to reduce the need for expensive tests on the real system it is advantageous to be able to combine such heterogeneous models in a joint co-simulation in order to reduce the overall costs of validation. This paper demonstrates how this can be achieved for a commercial system developed by MAN Diesel & Turbo using a newly developed tool chain based on the Functional Mock-up Interface standard for co-simulation supporting different operating systems. The generality of the suggested approach also enables future scenarios incorporating constituent models supplied by sub-suppliers while protecting their Intellectual Property.

1 INTRODUCTION

With increased complexity in physical dynamics, control and communication, the development of Cyber-Physical Systems (CPS) require more advanced modelling and specialized tools. For differential-equation based continuous models, multiple tools are available (MathWorks, 2011; SYSTÈMES, 2017; Kleijn, 2006), each with their specific specialization and validity. Discrete event models are often developed within companies own software frameworks, or created in one of the many tools available. The interconnection between the physical and cyber parts of CPS is becoming more dependent and dynamical influences have to be considered. The main challenge connecting these models comes from the fundamental differences in the underlying mathematical frameworks, their simulation tools and how they are developed. In this regard one typically distinguishes between Discrete Event (DE) models based on discrete mathematics and Continuous-Time (CT) models that are based on differential equations. Many initiatives for connection tools in a so called co-simulation have been published (Fitzgerald et al., 2014; ITEA Office Association, 2015). However, connecting the specific tools making up the holistic simulation is often not the only issue. Deviations in development platforms and performance is as often the issue. A solution for this is a distributed co-simulation, where models can be executed, not only in the tool where they were developed, but also on the correct platform. Furthermore, a distribution of the simulation makes it possible to increase performance by utilizing additional hardware, given that the models are prepared for it.

At MAN Diesel & Turbo (MDT) the conventional approach for developing two-stroke combustion engines with a distributed embedded control system is being challenged. In particular for diesel engines pollution is a key element that it is desirable to reduce from a competitive perspective. New emission legislation focuses on the reduction of especially NO_x emission. Widely known emission reduction technologies for reducing NO_x are selective catalytic reduction and Exhaust Gas Recirculation (EGR), both being developed at MDT (MAN Diesel & Turbo, 2016). These systems require advanced algorithms to control the complexity of the physical dynamics of large engines. Historically, in the same way as

Pedersen, N., Lausdahl, K., Sanchez, E., Larsen, P. and Madsen, J.
Distributed Co-Simulation of Embedded Control Software with Exhaust Gas Recirculation Water Handling System using INTO-CPS.
In *Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2017)*, pages 73-82
ISBN: 978-989-758-265-3
Copyright © 2017 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

many other large organisations, MDT is divided into different departments with different responsibilities. In the control department at MDT, control algorithms are created directly in the target software framework with the possibility of performing Software In the Loop (SIL) simulation during development. Models of the physical behaviour are created in other departments of MDT using the tools most suitable for the specific constituent system. For control system development, the physical dynamics models are implemented in an internally developed tool for CT simulation called Dynamic Simulation Environment (DSE) which is part of the software framework. The primary focus in DSE is SIL/Hardware In the Loop (HIL), and the physics models implemented here are often an abstraction of high-fidelity models. Historically it has been challenging inside MDT to enable heterogeneous collaborations between the different teams producing models in different departments.

The software framework and DSE are based on C++ and run on a 32-bit Linux platform while the physical modelling tools often require Windows. In this paper the current simulation process at MDT is compared with an alternative using co-simulation utilizing the Functional Mock-up Interface (FMI) standard and the Co-simulation Orchestration Engine (COE) from the Integrated Tool Chain for Model-based Design of Cyber-Physical Systems (INTO-CPS) project. The aim with the approach suggested in this paper is to reduce redundancy in the development process and reuse models from different departments. One of the main challenges is to enable co-simulation across different hardware architectures and Operating System (OS) platforms due to constraints from software frameworks, physical simulation tools and version compatibility.

In section 2 the overall system is presented. section 3 describes the previous simulation of a specific subsystem for EGR, and section 4 describes the approach taken to enable co-simulation. Afterwards, section 5 describes the co-simulation results for the EGR system. Finally, the paper concludes with section 6.

2 EXHAUST GAS RECIRCULATION WATER HANDLING SYSTEM

The EGR system presented in this paper recirculates exhaust gas to the intake manifold thereby reducing environmental impact while maintaining efficient combustion. The unclean exhaust gas is potentially

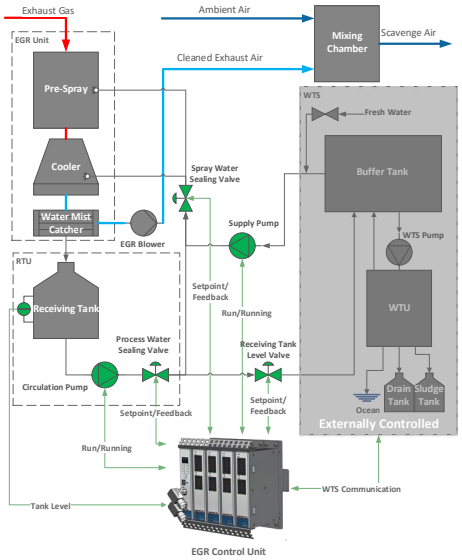

Figure 1: Water Handling System Setup.

damaging to the engine and has to be cleaned before return, which is the purpose of the Water Handling System (WHS). The system is shown in Figure 1 where the exhaust gas is drawn into the *EGR Unit* using an EGR blower, it is then sprayed with water and cooled so that a Water Mist Catcher (WMC) can collect the damaging particles. Before the gas is returned, the water is collected in the WMC and led to a receiving tank. The water level in this tank is one of the important variables that the WHS controls, as discussed in section 3.4. The water is pumped from the receiving tank to an external constituent system (Water Treatment System (WTS)) for processing where the water is either cleaned and pumped back to the EGR Unit, pumped overboard or stored for treatment at a harbour.

At the chemical level, EGR is based on exchange of the in-cylinder oxygen (O_2) with carbon dioxide (CO_2) from the exhaust gas, which is re-circulated into the scavenged air. The exchange of O_2 with CO_2 leads to a decrease of combustion speed, resulting in lower peak temperatures during combustion. Furthermore the exchange of O_2 with CO_2 results in a higher in-cylinder heat capacity of the gas which also lowers the combustion temperature. Lower combustion temperatures and especially lower peak temperatures result in lower formation of thermal NO_x during the combustion process. The recirculated exhaust gas is hotter and not as clean as the residual ambient scavenge-air. To prevent Sulphur (SO_2) and other par-

ticles from damaging the engine, cleaning and cooling of the recirculated exhaust gas is required. A WHS provides the water used for cleaning the exhaust gas in the EGR unit. To control the flow of exhaust gas to the mixing chamber, an EGR blower is installed. Water from the EGR unit is drained to the Receiving Tank Unit (RTU) and recirculated to the EGR unit. Part of the recirculated water is led to the WTS to be cleaned and returned to the EGR unit. The surplus of water originating from the combustion process is drained from the WTS as bleed-off water and discharged to the sea. The residuals from the cleaning process are discharged to the sludge tank. Depending on engine load and ambient conditions the combustion process will accumulate water in the system, which must be discharged as bleed-off water. If discharged to the sea, the bleed-off water must meet the quality criteria required by International Maritime Organization (IMO)[1], presently defined in the 2015 Guidelines for Exhaust Gas Cleaning Systems, MEPC 259 (68). Bleed-off water, which does not meet the discharge criteria or cannot be discharged to sea due to local restrictions, is drained to a drain tank for delivery at port.

Vessels operating within an emission control area have to comply with the Tier III emission requirements (IMO, 2015). This is achieved by activating EGR, at which point the water handling system is required to run. The control system for the WHS is divided into two parts, the EGR control which is part of the distributed engine control system and the WTS control which is delegated to the producer of the auxiliary system. The engine control system consists of several multi-purpose controllers. Each controller is composed of a power module, multiple I/O chassis and an Field-Programmable Gate Array (FPGA). All controllers on the engine are identical but the software running on the FPGA determines the specific control objective. The controller controlling the WHS is called the EGR Control Unit (EGRCU) and is seen in Figure 1, with the connections relevant for this simulation. This paper focuses on the control of the WHS. The remaining control of the EGR system will not be covered.

The WHS is controlled and monitored by the EGR control, so that water can be provided to clean and cool the exhaust gas. There are two main water loops that can be distinguished. The recirculation loop where the water from the EGR unit is sent to the RTU and back again by the 'Circulation pump' via the 'Process Water Sealing Valve' and 'Spray Water Sealing Valve'. The other loop is where part of the water from the recirculation loop is sent via the 'Receiving Tank Level Valve' to the externally controlled system, the WTS. The water from the WTS is sent back to the recirculation loop with the 'Supply Pump'. The WTS receives the processed water from the RTU and is collected in the buffer tank. A separate system in the WTS treats the water of the buffer tank. Any excess of water is either sent to the sludge/drain tank or, if the water quality parameters are met, the water can be sent overboard.

The objective of the control loop discussed in this paper is to maintain the water level of the 'Receiving Tank' within specified limits. During start up and shutdown of the WHS the actuation timing of the components has a direct impact on the water level. During running mode, the water level is controlled by the 'Receiving Tank Level Valve' and compensates for deviations in the water flow due to e.g. engine load, exhaust gas and scavenge air pressure changes.

[1] http://www.imo.org.

3 WHS SIMULATION

This section describes the development process of the primary WHS control strategy. The approach and tools used for the first edition of the control system are described and the solution is evaluated.

3.1 Software Application Framework

Application development at MDT is carried out in a comprehensive in-house C++ software application framework. The framework is developed to enable development of DE control models. The main advantage of the software application framework is the possibility of cross compiling the same application to both SIL, HIL and target platform, see Figure 2. SIL simulation of target code is made possible by compiling the Board Support Package (BSP) and the Real Time Operating System (RTOS) to an x86 platform. With the SIL simulation, engineers are able to test their application on their own PC. When moving to HIL or target, the same application code, the BSP and the RTOS, are simply cross-compiled to the embedded core of the controller. The primary focus of the framework is control development, where algorithms are directly implemented in C++ with a vast amount of reusable components and macros available, aiding engineers. For CT models, an extension to the framework can be utilized, called DSE. DSE includes a kernel for execution, an ODE solver and a model library of physical components. Models created in DSE are executable on both PC (SIL) and the HIL platform, given that the abstraction of the models allow for real-time execution.

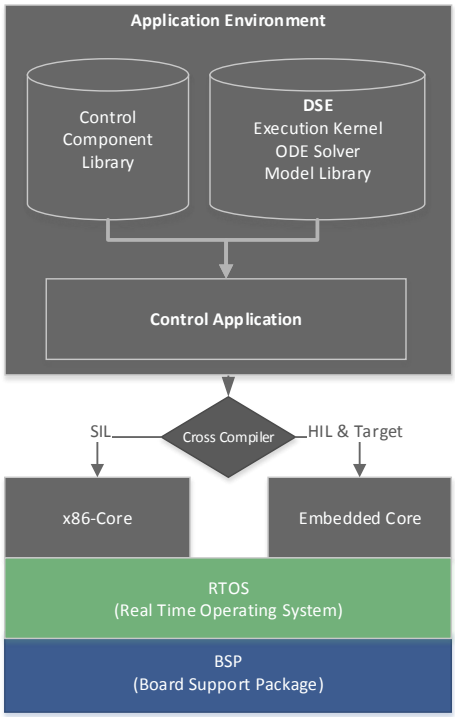

Figure 2: Software Application Framework.

When challenged with a new application, control engineers at MDT often start studying the physical dynamical challenges of the system in MATLAB/Simulink. When a sufficient understanding of the system is achieved, the control strategy is formulated in the software application framework and tested against a DSE model implementation of the MATLAB model. DSE is designed with HIL execution in mind, and while it can simulate complex CT systems, the models implemented are often at a lower abstraction level than e.g. the MATLAB models.

3.2 WHS Model

The WHS model is divided into a control algorithm created in the software application framework and a model of the physical components in DSE. The control algorithm is created as a component in the controller EGRCU along side the additional components that comprise the entire engine control. The control model consists of a Proportional Integral (PI)-controller regulating the 'Receiving Tank Level Valve' set-point from the 'Process Water Receiving Tank' level sensor feedback. Besides controlling the receiving tank level, the control algorithm also has to ensure that transitions between states in the system is possible, according to signals from engine operators and the WTS control system. This control strategy is formulated in a state-machine running in the EGRCU. The DSE model describes the pressures and flow of all the components illustrated in Figure 1. The model resembles the preliminary model developed in MATLAB but without details such as pressure build-up in piping, water accumulation in components and pressure loss over valves. The main purpose of the DSE is to test the control strategy, ensuring that all state transitions are possible and that the regulator works correctly. To prepare for HIL simulation, the DSE model is placed in a separate controller called the Engine Simulation Unit (ESU), shown in Figure 3. The ESU controller is installed in the HIL platform as a representation of the real engine. Data exchange between the EGRCU and ESU imitate the actual communication with the real engine through analog and digital IOs. In SIL, a software implementation of virtual IOs and network simulate the communication.

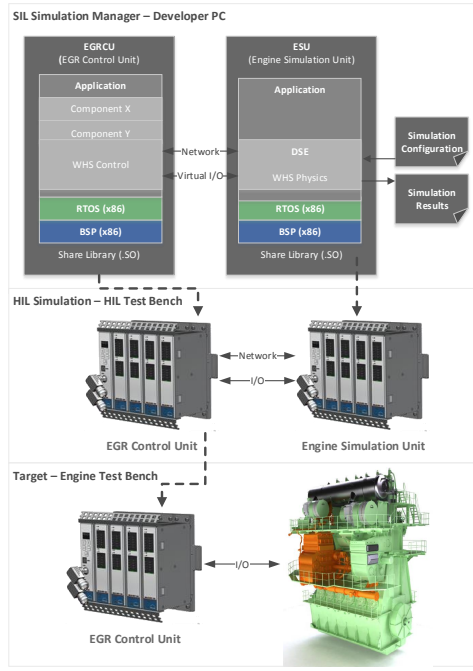

Figure 3: SIL WHS Simulation setup.

3.3 Simulation and Verification Process

SIL simulation is achieved by compiling controllers to an x86 platform and into shared libraries. The

shared libraries are executed by a simulation manager ensuring temporal execution and correct data exchange between controllers. How the embedded control software has been adapted to enable deterministic simulation has been described in (Pedersen et al., 2016) and will not be further explained in this paper. A simulation scenario is provided to the DSE model through a simulation configuration file and the results are delivered in a simulation results file.

When the system has been properly tested in the SIL environment, the models are moved to the HIL platform by cross-compiling to the embedded system. On the HIL test bench additional tests of computation overhead, communication and additional temporal issues are performed.

For final testing, an engine test bench is physically available at the MDT research center in Copenhagen. Only a single test bench is available due to the immense cost and sheer size of the engine. Available time-slots on the test engine are very limited and extremely costly, due to the fuel consumption and amount of operators required, so the proper modelling and testing of the previous steps is desirable.

3.4 Simulation Evaluation

The SIL simulation was used to develop a functioning PI controller that regulates the process water tank level and a state machine for actuating valves and pumps according to a number of states for starting and stopping the WHS system. The system was tested on the HIL test bench, ensuring that the systems worked properly on the controller hardware. Finally, a test session was performed on the engine test bench. This test showed that the PI controller worked as intended, however, an unsuspected situation occurred when stopping the WHS system.

Figure 4 shows the results of running the initial control strategy on the real system. After 100 seconds the EGR control system ordered the WHS system to prepare for EGR operation. Then after 50 seconds the state-machine was finished starting different pumps and opening of valves. At this point the Process Water Receiving Tank (PWRT) control is fully engaged. The bottom figure in 4 shows how the Receiving Tank Level Valve (RTLV) is regulated to redirect water from the process circuit to the WTS for cleaning and stabilizing the PWRT level. The top figure in Figure 4 shows the water level in the PWRT and how it became stable after a transient period, proving that the PWRT control worked correctly during WHS operation. Vessels are not always required to use EGR, so shut-down of the system should be possible during engine operation. After 600 seconds a command from the engine

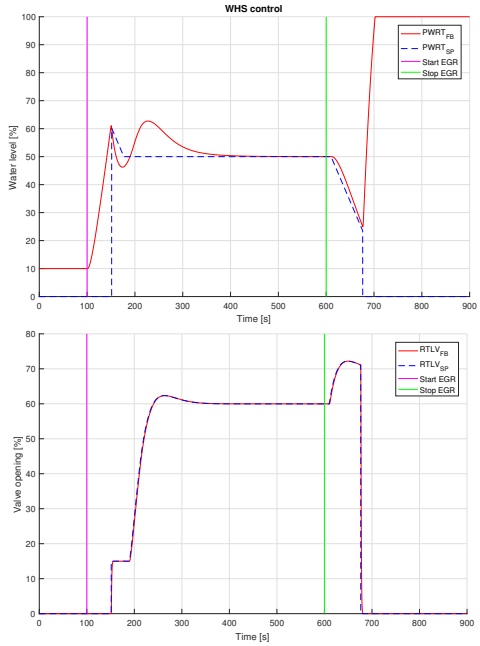

Figure 4: WHS Control results.

operator was ordered, from an operating panel, for the EGR system to shutdown and WHS to stop operation. The state machine started emptying the tank to reach a stable offline level around 20-25% in the tank. At 676 seconds a behaviour not seen in either the SIL or HIL simulation was observed. When the WHS system started, water in the WMC started to accumulate gradually. At a point in time an equilibrium was achieved due to increased water pressure resulting in a consistent flow through the WMC (without increased water accumulation in the WMC as a consequence). During shutdown, when the desired water level in the PWRT was achieved and RTLV control stopped, the accumulated water in the WMC started to flow to the PWRT tank. As seen in Figure 4, the amount of residual water in the WMC is so large that it overfills the PWRT.

From the engine test bench it was discovered that the controller actuating the RTLV was working properly, but the state-machine was not properly handling the emptying of the WMC. Engine tests are very costly and MDT would like to investigate if a more efficient development process can be achieved. In the DSE model used for development of the state-machine, the accumulation of water in the WMC had not been modelled. To improve the control strategy of the WHS, a higher-fidelity model should be used. Instead of simply extending the DSE model to include a

more detailed WMC model, a co-simulation solution was chosen (Gomes et al., 2017). The co-simulation should not only include a detailed WMC model but be so generic that changes to the system layout and more advanced models of components can be easily implemented. The argument for the choice of co-simulation is given in the following section.

4 TARGETING CO-SIMULATION

The software application framework and DSE is central for development because they are designed for the target platform of the final system, and directly enable validation through both SIL and HIL. Keeping this in mind it is rational to keep the control systems in the framework. However, there are a number of options for enhancing physics modelling that would be beneficial:

- Porting the controller software to a notation that can be used in the MATLAB environment, where it is easier to express the physical model. This would however, just shift the issue to the controller, that then needs to be ported back to the software application framework.
- Enhancing the physical model in DSE, while the standard approach, it is more time consuming than using a dedicated modelling tool like MATLAB, but it enables faster simulation speeds.
- Use a generic solution that enables co-simulation between the control system expressed in the software application framework and a physical modelling tool like MATLAB. This will not require any changes to software development at MDT, but would enable physical models to be created using the desired modelling tool. It would potentially run slower than a complete model expressed in DSE but would be more flexible. This solution would make the representation of the physical dynamics more detailed in SIL simulation. The co-simulation model would not be able to run on the HIL platform, however. The purpose of the HIL test is not to test functionality already verified in the SIL simulation, but to ensure computational overhead and investigate temporal aspects.

The latter approach was chosen because it is generic and it allows well known modelling tools in the physical domain to be used. To interface between models, the FMI is used, which provides a standardised model interface. The last constraint on the co-simulation is that it needs to be performed across architectures and platforms. The software application framework is required to run as a Linux 32bit process. The reason for this is, as previously mentioned, because the framework is developed to build directly to the embedded system which is a 32bit architecture. It is also a requirement that the physical modelling environment be a Windows 64bit application. The control developers working in e.g. MATLAB do so in Windows 64bit and management-wise, introducing co-simulation to the current tool-chain would be preferable. Another reason for the choice of deviation in platform is the lack of 32bit support for MATLAB on Linux.

It must be possible to run the simulation using Linux 32bit for the software application framework and Windows 64bit for MATLAB. Therefore a solution is to use the free FMI COE from the INTO-CPS research project since it supports both. However, the co-simulation cannot span architectures or platforms. Therefore an extension is presented in section 5.1 that enables co-simulation in a distributed setting, spanning both architectures and platforms.

4.1 Functional Mock-up Interface

FMI is a tool independent standard developed within the MODELISAR project (ITEA Office Association, 2015). It supports both model exchange and co-simulation and exists as Version 1, released in 2010 and Version 2, released in 2014. It was developed to improve exchange of simulation models between suppliers and Original Equipment Manufacturers (OEM). The standard describes how simulation units are to be exchanged as ZIP archives called a *Functional Mock-up Unit (FMU)* and how the model interface is described in an XML file named *modelDescription.xml*. The functional interface of the model is described as a number of C functions that must be exported by the library that implements the model inside the FMU. Since the FMU only contains a binary implementation of the model it offers some level of intellectual property protection. The focus of this work is on co-simulation, where each FMU is capable of participating in a co-simulation without the need of an external solver, i.e. each FMU includes the required solvers needed for simulation.

4.2 The INTO-CPS Tool Chain

While individual tools and formalisms for the development of controllers, including simulation, testing and code generation, are very mature, the design workflow is only partially integrated. The Horizon 2020 project INTO-CPS (Fitzgerald et al., 2015; Fitzgerald et al., 2016) aims at closing this gap, by

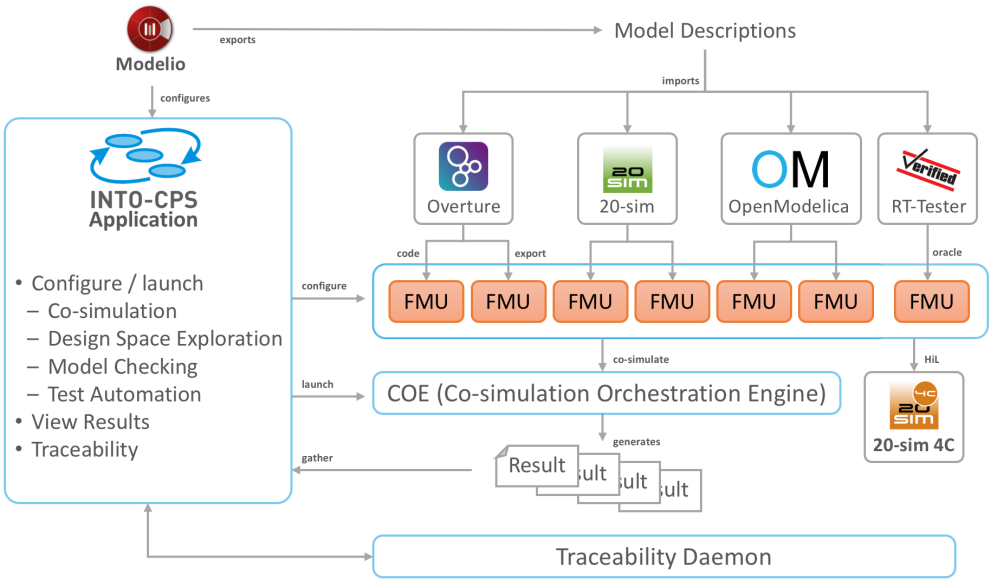

Figure 5: The INTO-CPS Tool Chain.

creating an Integrated Tool-chain for the model-based design of Cyber-Physical Systems (Larsen et al., 2016). The chain of tools are connected as illustrated in Figure 5, moving all the way from requirements to final realisations (Bandur et al., 2016). One of the core tools of this chain is a newly developed COE, which is a fully FMI 2.0 co-simulation compliant Master supporting both fixed and variable step size simulations. It was decided to use FMI as the interface for the different simulation and testing tools, since it is a mature standard[2] created in the MODELISAR project (ITEA Office Association, 2015) with an active community.

The COE is developed in a combination of Java and Scala, which makes it multi-platform and provides the simulation service through HTTP. Currently, two methods for time-stepping are implemented; one for fixed time steps, and one for variable time steps. The COE is capable of switching on stability checking as well as using parallelism (Thule and Larsen, 2016). In addition to the baseline tools incorporated inside the tool chain, a number of other modelling and simulation tools have been tested with the COE. This includes both commercial tools such as Dymola, Modelon, SimulationX and Unity as well as additional open source tools such as 4Diac. While the COE is multi-platform it does not directly support mixed-architecture (combinations of 32bit and 64bit architectures) or mixed-platform (combinations of e.g. Windows and Linux) simulations as required for the WHS system as discussed next in section 5.

[2]http://fmi-standard.org

5 WHS CO-SIMULATION

To co-simulate the WHS from section 3 using FMI, it is required that both constituent models must support FMI, and that a suitable orchestration engine that supports FMI and the required platform and architecture combination is available. Since no such simulator is available an extension to the COE is described in section 5.1. To enable FMI for the constituent models, an extension was developed for the MDT software application framework which has been published in (Pedersen et al., 2016; Pedersen et al., 2015). The model of the WHS is exported from MATLAB to an FMU using the Modelon FMI Toolbox for MATLAB/Simulink (Modelon, 2015). The complete co-simulation model is shown in section 5.2, and evaluated in section 5.3.

5.1 Distributed COE Extension

To enable multi-architecture co-simulation, the challenge of mixing 32bit and 64bit code needs to be addressed. Essentially, two processes with inter-process

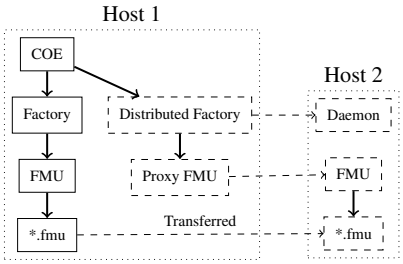

Figure 6: Distributed Extension Overview.

communication are required by the host system to realize this, where one of them acts as the simulation master. A similar challenge arises when different platforms need to interact in a co-simulation.

An extension to the COE was developed that is capable of both simulating across architectures and platforms. The solution chosen was to utilize an extension point in the COE that allows a custom factory to be used for FMU instantiation. An overview of the extension is realized and shown in in Figure 6. The COE uses the distributed factory to instantiate FMUs that require execution with a different host configuration, either architecture or platform deviation.

The extension is realized using Java-Remote Method Invocation (JAVA-RMI) to provide cross-platform communication (JavaRMI, 2004). It consists of a distribution factory and an FMU proxy that is plugged into the COE. It uses a daemon that must run on the remote host to provide a service that enables the COE to remotely load and control FMUs. The COE configuration is also extended to specify which remote daemon a specific FMU should be executed by. When a co-simulation is started the COE will communicate with the specified remote daemons to configure the co-simulation by first pushing FMUs to the remote daemons that then in turn load and setup a communication channel for the loaded FMUs. These will then be connected to the FMU proxy in the COE, which is responsible for handling remote communication.

5.2 Co-Simulation Setup

The co-simulation setup is illustrated in Figure 7. The master COE is running on the Windows host, and the COE-deamon on the Linux host. A JSON configuration file describes the co-simulation setup to the COE. The configuration file tells the COE where the FMU-archives are located and on which host-ip they should be executed. The configuration file also contains information about connections between the inputs and outputs of the FMUs, parameters and simulation al-

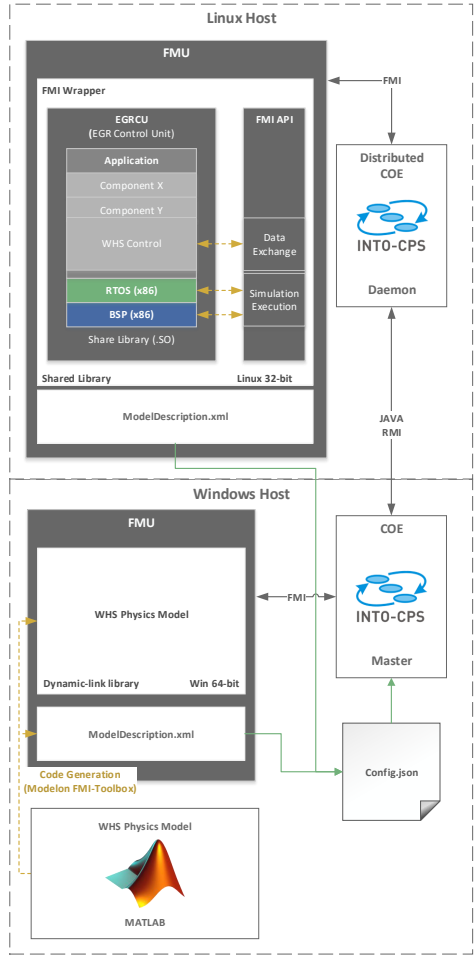

Figure 7: Co-Simulation Configuration.

gorithm: variable/fixed time step.

The WHS MATLAB model is code generated into an FMU using the Modelon FMI toolbox for MATLAB/simulink. The toolbox compiles the MATLAB model to a 64-bit DLL including the FMI-API and auto-generates the model description XML defining the interface to the FMU. The control system FMU has been created by wrapping the FMI Application Programming Interface (API) around the SIL simulation and compiling it to a Linux 32-bit shared library. The simulation can access the RTOS for scheduling and a hook to the clock in the BSP, all described in (Pedersen et al., 2016). Accessing the variables of the WHS control is done through a proxy interface that provides pointers to internal variables to be manipu-

lated. Furthermore, the proxy interface introduces a conversion layer between internal types such as FIX-POINT16 and FMI-types. The SIL simulation only includes the EGRCU controller, which contains the WHS control. The ESU controller, containing the DSE models, has been replaced with the MATLAB model in the co-simulation.

The simulation is initiated through the COE and results delivered in Comma-Separated Values (CSV) format on the Windows host.

5.3 Co-Simulation Evaluation

Figure 8 shows the simulation results with the proposed Co-Simulation setup, where the DSE physical model has been replaced with the more detailed MATLAB model. Being able to anticipate the behaviour of the accumulated water in the Water Mist Catcher, it is now possible to address it and to modify the state machine accordingly to control the components in a more appropriate way during WHS shutdown. The first 600 seconds show the same response as in Figure 1. However, the new state machine now ensures

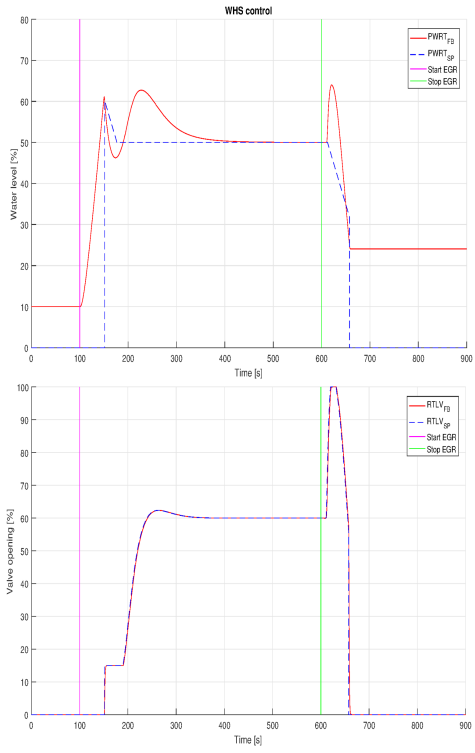

Figure 8: WHS Control results.

that the WMC is drained before shutting down RTLV control. This prevents the water level in the PWRT from overflowing, but instead stabilize at a desired level of approximately 20-25%.

6 CONCLUDING REMARKS

This paper shows how the control development process at MAN Diesel & Turbo could benefit from introducing co-simulation. The conventional approach, where control algorithms and strategy are formulated using simplified models of the physical dynamics, is not always able to properly represent the complexity of the system. Importantly, with this approach, defects are typically not found before moving to the expensive engine test bench. With the co-simulation approach proposed in this paper, higher-fidelity constituent models of physical dynamics, formulated in dedicated tools, can be simulated together with a SIL simulation of the control software, at an earlier stage of development. In the example presented, the accumulation of water in a water mist catcher was neglected in the initial model, essentially resulting in a water tank overflow during shutdown. With the co-simulation, a more detailed model, formulated in MATLAB, could be used for developing a working control strategy. Had the co-simulation been used for initial control development, the issues seen on the test engine would likely have been discovered at an earlier stage, saving money and time.

The main challenge enabling co-simulation at MDT was the deviation in both OS platform and hardware architecture of the simulation tools used. The SIL simulation of the control software is constrained to a 32-bit Linux platform and the MATLAB environment was required to run on a 64-bit Windows platform due to change management concerns. In cooperation with the INTO-CPS-project, the INTO-CPS Co-simulation Orchestration Engine for executing co-simulations complying with the Functional Mock-up Interface standard was adapted to enable distributed co-simulation. With the distributed COE it was possible to conduct the co-simulation despite the platform and architecture deviation.

Besides the promising results shown in this paper, additional benefits form the INTO-CPS co-simulation tool chain are anticipated. The extensions developed to the MDT frameworks and development processes enable not only co-simulation of the EGR system, but also of any other system to be developed in the future by MDT, with minimal effort. In the future it will be explored when it makes sense to also make use of more capabilities from the INTO-CPS tool chain.

In the EGR Water Handling System presented here, a subsystem called Water Treatment System is delivered by an MDT OEM and neither modelled nor controlled by MDT. One of the main advantages of the Functional Mock-up Interface standard used by INTO-CPS is that models are exchanged on a binary level offering protection of intellectual property. One of the future ambitions is to be able to share models with OEMs so systems like the WHS and WTS can be simulated together, improving both companies products. Part of the high-fidelity models developed at MDT are very complex and require time to simulate, especially if co-simulated with several other models. One of the additional advantages of the distributed co-simulation is that the simulation process can be parallelized and perhaps distributed to centralized high-performance hardware. This could potentially speed up simulation execution times and enable more advanced system investigations, previously deemed too time consuming. Initial work on using the COE in a cloud setting has already been initiated, in particular in relation to design space exploration in situations where there is large room for different alternative solutions.

ACKNOWLEDGEMENTS

The work presented here is partially supported by the INTOCPS project funded by the European Commission's Horizon 2020 programme under grant agreement number 664047. In addition we would like to thank Victor Bandur for inout on an earlier draft of this article.

REFERENCES

Bandur, V., Larsen, P. G., Lausdahl, K., Thule, C., Terkelsen, A. F., Gamble, C., Pop, A., Brosse, E., Brauer, J., Lapschies, F., Groothuis, M., Kleijn, C., and Couto, L. D. (2016). INTO-CPS Tool Chain User Manual. Technical report, INTO-CPS Deliverable, D4.2a.

Fitzgerald, J., Gamble, C., Larsen, P. G., Pierce, K., and Woodcock, J. (2015). Cyber-Physical Systems design: Formal Foundations, Methods and Integrated Tool Chains. In *FormaliSE: FME Workshop on Formal Methods in Software Engineering*, Florence, Italy. ICSE 2015.

Fitzgerald, J., Gamble, C., Payne, R., Larsen, P. G., Basagiannis, S., and Mady, A. E.-D. (2016). Collaborative Model-based Systems Engineering for Cyber-Physical Systems – a Case Study in Building Automation. In *INCOSE 2016*, Edinburgh, Scotland.

Fitzgerald, J., Larsen, P. G., and Verhoef, M., editors (2014). *Collaborative Design for Embedded Systems – Co-modelling and Co-simulation*. Springer.

Gomes, C., Thule, C., Broman, D., Larsen, P. G., and Vangheluwe, H. (2017). Co-simulation: State of the art. Technical report.

IMO, I. M. O. (2015). MARPOL ANNEX VI and NTC 2008 with Guidelines for Implementation - Supplement. Technical Report September 2015.

ITEA Office Association (2015). Itea 3 project 07006 modelisar. https://itea3.org/project/modelisar.html. (Visited on 12/06/2015).

JavaRMI (2004). Java remotemethodinvocation specification 1.5.0. http://java.sun.com/j2se/1.5/pdf/rmi-spec-1.5.0.pdf.

Kleijn, C. (2006). Modelling and Simulation of Fluid Power Systems with 20-sim. *Intl. Journal of Fluid Power*, 7(3).

Larsen, P. G., Fitzgerald, J., Woodcock, J., Fritzson, P., Brauer, J., Kleijn, C., Lecomte, T., Pfeil, M., Green, O., Basagiannis, S., and Sadovykh, A. (2016). Integrated Tool Chain for Model-based Design of Cyber-Physical Systems: The INTO-CPS Project. In *CPS Data Workshop*, Vienna, Austria.

MAN Diesel & Turbo (2016). Emission Project Guide, MAN BW Two-stroke Marine Engines. Technical report, MAN Diesel & Turbo.

MathWorks (2011). http://www.mathworks.com. Matlab official website.

Modelon (2015). http://www.modelon.com/products/fmi-tools/fmi-toolbox-for-matlabsimulink/. Modelon FMI Toolbox for MATLAB/Simulink official website.

Pedersen, N., Bojsen, T., Madsen, J., and Vejlgaard-Laursen, M. (2016). FMI for Co-Simulation of Embedded Control Software. In *Linköping Electronic Conference Proceedings*, number 124, pages 70–77.

Pedersen, N., Madsen, J., and Vejlgaard-Laursen, M. (2015). Co-Simulation of Distributed Engine Control System and Network Model using FMI and SCNSL. *10th IFAC Conference on Manoeuvring and Control of Marine Craft MCMC 2015*, 48(16):261–266.

SYSTÈMES, D. (2017). https://www.3ds.com/products-services/catia/products/dymola. 3ds official website.

Thule, C. and Larsen, P. G. (2016). Investigating concurrency in the co-simulation orchestration engine for into-cps. In Alexander S. Kamkin, A. K. P. and Terekhov, A. N., editors, *Preliminary Proceedings of the 10th Anniversary Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2016), Krasnovidovo, Russia, May 30-June 1, 2016)*, pages 223–228. ISP RAS.

Bibliography

Abel, A., Blochwitz, T., Eichberger, A., Hamann, P., and Rein, U. (2012). Functional mock-up interface in mechatronic gearshift simulation for commercial vehicles. *9th International MODELICA Conference*, pages 775–780.

Alegret, G., Llamas, X., Vejlgaard-Laursen, M., and Eriksson, L. (2015). Modeling of a large marine two-stroke diesel engine with cylinder bypass valve and EGR system. *IFAC Proceedings Volumes (IFAC-PapersOnline)*, 48(16):273–278.

Awais, M. U., Palensky, P., Mueller, W., Widl, E., and Elsheikh, A. (2013). Distributed hybrid simulation using the HLA and the Functional Mock-up Interface. *IECON Proceedings (Industrial Electronics Conference)*, pages 7564–7569.

Bandur, V., Larsen, P. G., Lausdahl, K., Thule, C., Terkelsen, A. F., Gamble, C., Pop, A., Brosse, E., Brauer, J., Lapschies, F., Groothuis, M., Kleijn, C., and Couto, L. D. (2016). {INTO-CPS} Tool Chain User Manual. Technical report, {INTO-CPS} Deliverable, D4.2a.

Bian, D., Kuzlu, M., Pipattanasomporn, M., Rahman, S., and Wu, Y. (2015). Real-time co-simulation platform using OPAL-RT and OPNET for analyzing smart grid performance. *IEEE Power and Energy Society General Meeting*, 2015-Septe:1–5.

Blochwitz, T., Otter, M., Åkesson, J., Arnold, M., Clauss, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., others, Blockwitz, T., Otter, M., Akesson, J., Arnold, M., Clauss, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., Olsson, H., and Viel, A. (2012). Functional mockup interface 2.0: The standard for tool independent exchange

of simulation models. In *9th International Modelica Conference*, pages 173–184.

Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Clauß, C., Elmqvist, H., Junghanns, a., Mauss, J., Monteiro, M., Neidhold, T., Neumerkel, D., Olsson, H., Peetz, J. V., and Wolf, S. (2009). The Functional Mockup Interface for Tool independent Exchange of Simulation Models. *8th International Modelica Conference 2011*, pages 173–184.

Brezina, T., Hadas, Z., and Vetiska, J. (2011). Using of Co-simulation ADAMS-SIMULINK for development of mechatronic systems. *14th International Conference Mechatronika*, pages 59–64.

Broman, D., Brooks, C., Greenberg, L., Lee, E. A., Masin, M., Tripakis, S., and Wetter, M. (2013). Determinate composition of FMUs for co-simulation. *2013 Proceedings of the International Conference on Embedded Software, EMSOFT 2013*, pages 1–12.

Brooks, C. X., Lee, E. A., and Tripakis, S. (2010). Exploring models of computation with ptolemy II. *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis - CODES/ISSS '10*, page 331.

Dahmann, J. (1997). High Level Architecture for simulation. *Proceedings First International Workshop on Distributed Interactive Simulation and Real Time Applications*, pages 9–14.

Design Automation, S. C. (2012). *IEEE Std 1666-2011, IEEE Standard for Standard SystemC® Language Reference Manual*, volume 2011.

Dols, W. S., Emmerich, S. J., and Polidoro, B. J. (2016). Coupling the multizone airflow and contaminant transport software CONTAM with EnergyPlus using co-simulation. *Building Simulation*, 9(4):469–479.

Eker, J., Janneck, J. W., Lee, E. A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., and Xiong, Y. (2003). Taming heterogeneity - The ptolemy approach. *Proceedings of the IEEE*, 91(1):127–143.

Elsheikh, A., Awais, M. U., Widl, E., and Palensky, P. (2013). Modelica-enabled rapid prototyping of cyber-physical energy systems via the functional mockup interface. *2013 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems, MSCPES 2013*, pages 1–6.

Fitzgerald, J., Gamble, C., Larsen, P. G., Pierce, K., and Woodcock, J. (2015). Cyber-Physical Systems design: Formal Foundations, Methods and Integrated Tool Chains. In *FormaliSE: FME Workshop on Formal Methods in Software Engineering*, Florence, Italy. ICSE 2015.

Fitzgerald, J., Gamble, C., Payne, R., Larsen, P. G., Basagiannis, S., and Mady, A. E.-D. (2016). Collaborative Model-based Systems Engineering for Cyber-Physical Systems – a Case Study in Building Automation. In *INCOSE 2016*, Edinburgh, Scotland.

Fitzgerald, J. and Gorm, P. (2014). *Collaborative Design for Embedded Systems*.

Fummi, F., Quaglia, D., and Stefanni, F. (2008). A SystemC-based framework for modeling and simulation of networked embedded systems. *2008 Forum on Specification, Verification and Design Languages*, pages 49–54.

Gomes, C., Thule, C., Broman, D., Larsen, P. G., and Vangheluwe, H. (2017). Co-simulation: State of the art. Technical report.

Hafner, I., Heinzl, B., and Rössler, M. (2013). An Investigation on Loose Coupling Co-Simulation with the BCVTB. *Simulation Notes Europe SNE*, 23(1):45–50.

Hansen, J. M., Zander, C.-G., Pedersen, N., Blanke, M., and Vejlgaard-Laursen, M. (2013). Modelling for Control of Exhaust Gas Recirculation on Large Diesel Engines. *IFAC Proceedings Volumes*, 46(33):380–385.

ITEA Office Association (2015). ITEA 3 Project 07006 MODELISAR. \url{https://itea3.org/project/modelisar.html}.

Larsen, P. G., Fitzgerald, J., Woodcock, J., Fritzson, P., Brauer, J. J., Kleijn, C., Lecomte, T., Pfeil, M., Green, O., Basagiannis, S., and Sadovykh, A. (2016). Integrated Tool Chain for Model-based Design of Cyber-Physical Systems: The INTO-CPS Project. In *CPS Data Workshop*, pages 1–6, Vienna, Austria. IEEE.

Lee, E. A. (2008). Cyber Physical Systems: Design Challenges. *11th IEEE Int. Symp. on Object and Component-Oriented Real-Time Distributed Computing*, pages 363–369.

Li, S. and He, L. (2011). Co-simulation study of vehicle ESP system based on ADAMS and MATLAB. *Journal of Software*, 6(5):866–872.

Lin, H., Sambamoorthy, S., Shukla, S., Thorp, J., and Mili, L. (2011). Power system and communication network co-simulation for smart grid applications. *IEEE PES Innovative Smart Grid Technologies Conference Europe, ISGT Europe*, pages 1–6.

Liu, J., Liu, X., and Lee, E. A. (2001). Modeling Distributed Hybrid Systems in Ptolemy I1. *Proceedings of the 2001 American Control Conference*, 6:4984–4985.

Mews, M., Svacina, J., and Weißleder, S. (2012). From AUTOSAR models to co-simulation for MiL-testing in the automotive domain. *Proceedings - IEEE 5th International Conference on Software Testing, Verification and Validation, ICST 2012*, pages 519–528.

Nouidui, T. S., Wetter, M., and Zuo, W. (2014). Functional mock-up unit for co-simulation import in EnergyPlus. *Journal of Building Performance Simulation*, 7(3):192–202.

Oracle Corporation (2004). Java Remote Method Invocation Specification 1.5.0. http://java.sun.com/j2se/1.5/pdf/rmi-spec-1.5.0.pdf.

Organization., I. M. (2013). *MARPOL : Annex VI and NTC 2008 with guidelines for implementation*. International Maritime Organization London, third edit edition.

Paper, C., Durak, U., and Aerospace, G. (2015). MOKA : An Object-Oriented Framework for FMI MOKA : An Object-Oriented Framework for FMI. *47th Summer Computer Simulation Conference 2015*, (July).

Pedersen, N., Bojsen, T., and Madsen, J. (2017a). CO-SIMULATION OF CYBER PHYSICAL SYSTEMS WITH HMI FOR HUMAN IN THE LOOP INVESTIGATIONS. *Proceedings of the Symposium on Theory of Modeling & Simulation*, pages 1:1–1:12.

Pedersen, N., Bojsen, T., Madsen, J., and Vejlgaard-Laursen, M. (2016). FMI for Co-Simulation of Embedded Control Software. In *Linköping Electronic Conference Proceedings*, number 124, pages 70–77. MAN Diesel & Turbo, Copenhagen, Denmark, Linköping University Electronic Press, Linköpings universitet.

Pedersen, N., Lausdahl, K. G., Sanchez, E. V., Larsen, P. G., Madsen, J., Vidal, E. S., Lausdahl, K. G., Madsen, J., Pedersen, N., and Larsen, P. G. (2017b). Distributed Co-Simulation of Embedded Control Software with Exhaust Gas Recirculation Water Handling System using INTO-CPS. In *Simultech 2017*, pages 73–82.

Pedersen, N., Madsen, J., and Vejlgaard-Laursen, M. (2015). Co-Simulation of Distributed Engine Control System and Network Model using FMI and SCNSL. *IFAC-PapersOnLine*, 48(16):261–266.

Stoermer, C. and Tibba, G. (2014). Powertrain Co-Simulation using AUTOSAR and the Functional Mockup Interface standard. *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference - DAC '14*, (March):1.

Thule, C. and Larsen, P. G. (2016). Investigating Concurrency in the Co-Simulation Orchestration Engine for INTO-CPS. In Alexander S. Kamkin, A. K. P. and Terekhov, A. N., editors, *Preliminary Proceedings of the 10th Anniversary Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2016), Krasnovidovo, Russia, May 30-June 1, 2016)*, pages 223–228. ISP RAS.

Vanfretti, L., Bogodorova, T., and Baudette, M. (2014). Power system model identification exploiting the Modelica language and FMI technologies. *2014 IEEE International Conference on Intelligent Energy and Power Systems, IEPS 2014 - Conference Proceedings*, pages 127–132.

Wahlstrom, J. and Eriksson, L. (2011). Modelling diesel engines with a variable-geometry turbocharger and exhaust gas recirculation by optimization of model parameters for capturing non-linear system dynamics. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 225(7):960–986.

Widl, E., Muller, W., Elsheikh, A., Hortenhuber, M., and Palensky, P. (2013). The FMI++ library: A high-level utility package for FMI for model exchange. *2013 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems, MSCPES 2013*.