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

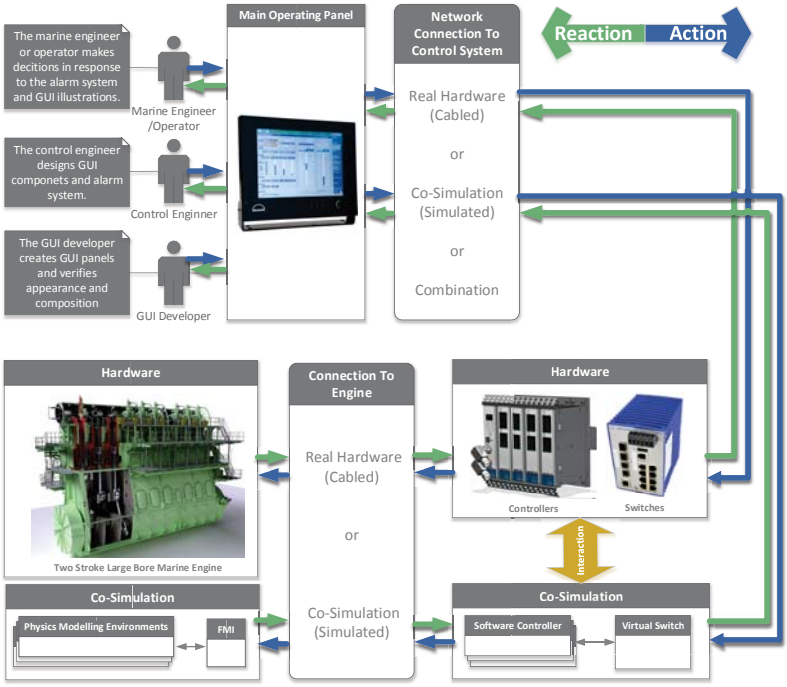

Figure 1: Co-simulation connection options.

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.

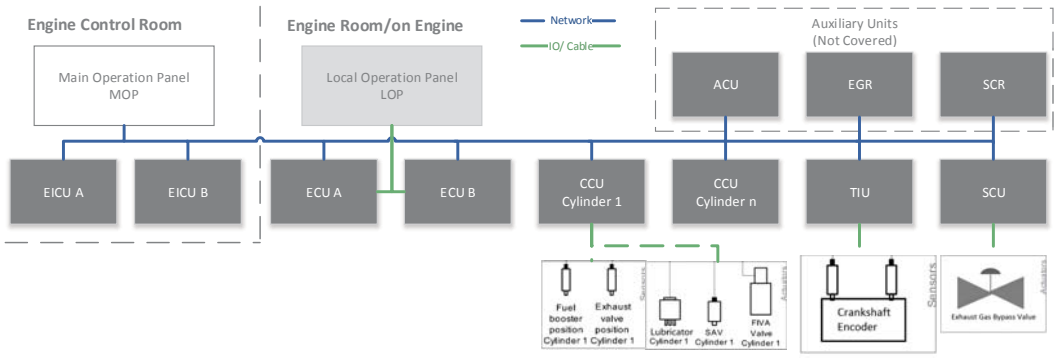

Figure 2: Engine control system communication diagram.

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.

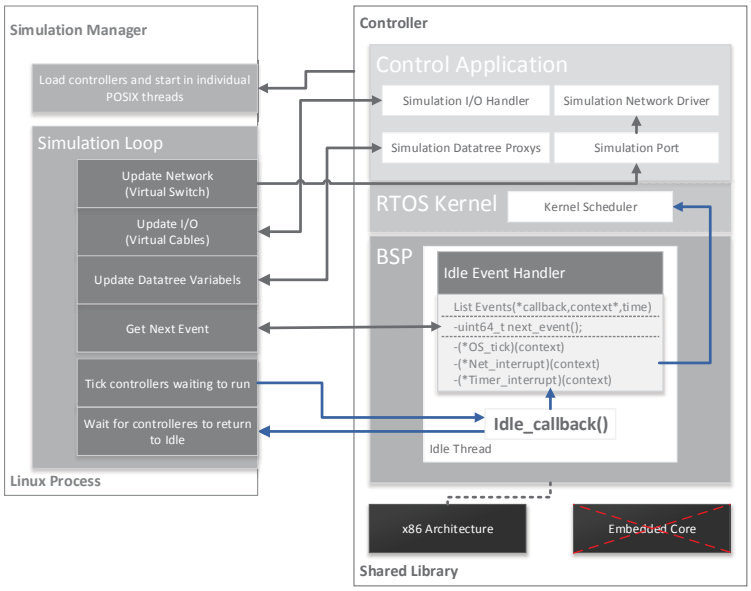

Figure 3: The software model of the embedded system and connection to the simulation manager.

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

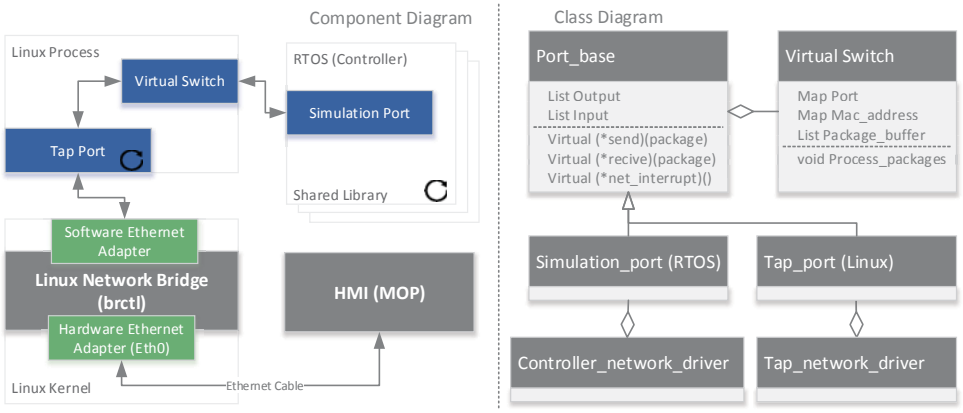

Figure 4: Illustration of the virtual switch connecting the co-simulation with the HMI.

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.

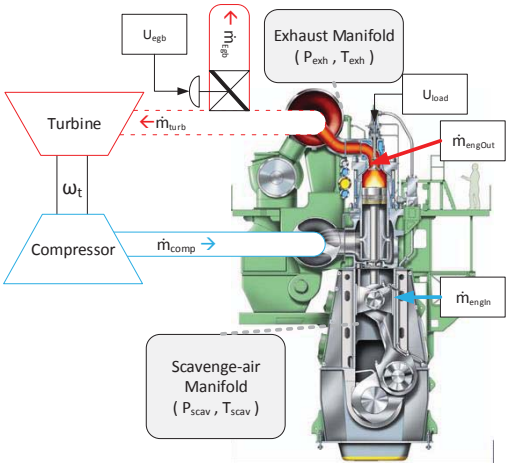

Figure 5: Engine air-path model.

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

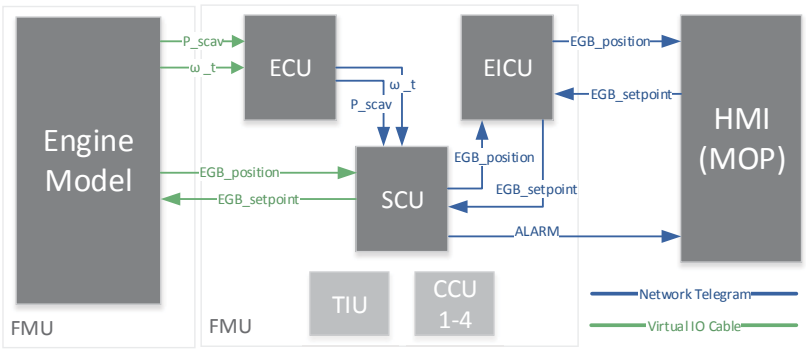

Figure 6: The two FMUs and the MOP are illustrated with relevant cable and network connections.

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.

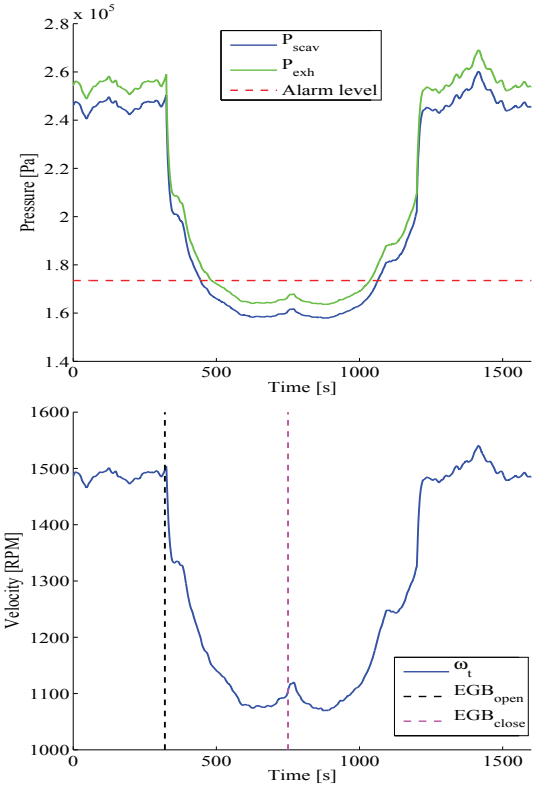

Figure 7: Plot of pressures and turbocharger velocity during the simulation scenario.

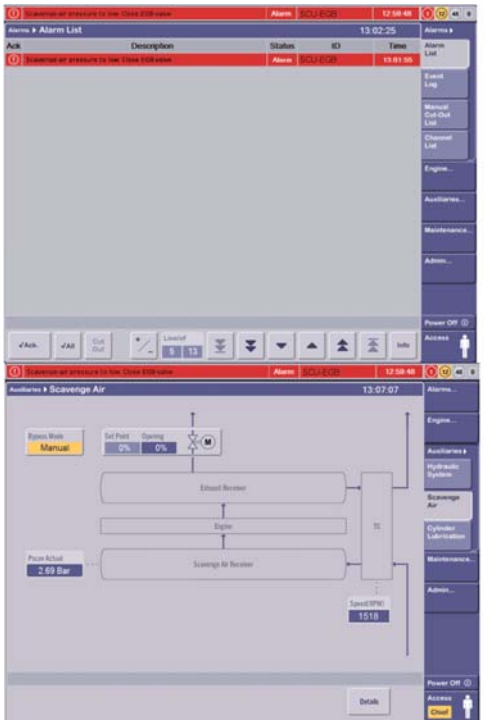

Figure 8: Top: Alarm system panel, Bottom: Scavenge Air panel.

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.