Early Analysis of Cyber-Physical Systems using Co-simulation and Multi-level Modelling

Thomas Nägele
Radboud University
Nijmegen
The Netherlands
t.nagele@cs.ru.nl

Tim Broenink
University of Twente
Enschede
The Netherlands
t.g.broenink@utwente.nl

Jozef Hooman
Radboud University & ESI (TNO)
Nijmegen & Eindhoven
The Netherlands
hooman@cs.ru.nl

Jan Broenink
University of Twente
Enschede
The Netherlands
j.f.broenink@utwente.nl

***Abstract*—The multi-disciplinary nature of the design of cyber-physical systems makes it hard to gain insight in the system behaviour early in the design process. Our aim is to allow the designers to analyse the integration of system components as well as the behaviour of the complete system in an early stage. This is achieved by creating abstract component models and refining them throughout the design process. After every refinement cycle, the models can be co-simulated to analyse the behaviour of the system, supporting design decisions. The co-simulation is created based on existing standards such as HLA and FMI and uses a domain-specific language to construct a co-simulation automatically. This approach is illustrated using a case study which resembles a confidential industrial case.**

***Index Terms*—Cyber-physical systems, System design, System analysis, Co-simulation, Multi-level modelling, Concurrent development, HLA, FMI**

I. Introduction

The development of a cyber-physical system (CPS) is a complex multidisciplinary process. After the formulation of the requirements, a system architecture is defined with mechanical components, computing platforms, software components, etc. When integrated, these parts should together lead to a system that meets the requirements. Typically, however, the test and integration phase reveals many issues, leading to a lot of rework and project delays. To detect issues earlier in the development process, many methods for model-based control are proposed [11][9][8]. Our aim is to support an efficient and effective model-based method for CPSs that supports the industrial development process.

In this paper we focus on a method that uses co-simulation of models to analyse system behaviour in the early phases of development. Since multiple disciplines are involved, each one having its own modelling technology, the co-simulation of a number of different types of models has to be supported. The construction of such a co-simulation should be easy to allow fast design space exploration and industrial usage. Moreover, it should be possible to start with abstract models and gradually refine them into more detailed models, finally leading to a realisation in terms of hardware or code. The co-simulation should allow combinations of models at different levels of abstraction.

Our co-simulation technology is based on two standards. The Functional Mock-up Interface (FMI) [3] aims for interoperability between models from different tools. The High-Level Architecture (HLA) [1] can be used to synchronise multiple simulations. The manual construction of a co-simulation using these standards is far from trivial and time consuming. Also adaptations after changes, e.g. in interfaces of components, take a considerable amount of time. Since this can be a blocking factor for industrial applications, we provide a high-level language, automating the construction of a co-simulation to a large extent. It allows developers to design components in their own modelling tools while still being able to construct a virtual prototype easily, thus supporting the concurrent development of these components.

The development of models of the components is an iterative process. Every development cycle adds more details or fixes issues from the previous version of the model, working towards a realisation of the component. For instance, an abstract model may only contain information on the movements of a component while a more detailed model might also describe the motor that is responsible for the movement. When the model is mature enough, the model may be realised into actual hardware or software.

Our approach is illustrated by the design of a small CPS in our lab which reflects the characteristics of a confidential industrial case we are working on. We show that a virtual prototype of the system can already be constructed during the early phases in the development process. We start with rather abstract models of the components until enough information is available to create a co-simulation to analyse the design. Next, the models are refined, adding more detail and obtaining more accurate simulation results. This allows for the design of system-level features using co-simulation. Finally, the models are realised into a real CPS.

The remainder of this paper is structured as follows. Section II summarises related work. Section III explains the system that was designed, after which Section IV describes the design flow that was used as well as the interfaces between the models. Section V describes how the initial models for the design were created. Section VI provides the method that was used to create the co-simulation and Section VII describes how the design was refined using the co-simulation. We briefly explain the realisation of the system in Section VIII, round up with the results in Section IX and conclude in Section X.

II. Related work

For the creation of co-simulations based on the FMI standard, approaches have already been developed. The INTO-CPS [10] project provides a tool chain to construct co-simulations from sets of FMI compliant models. The INTO-CPS tool chain, however, relies on a specific set of tools and requires quite some meta-modelling in order to create a co-simulation. The use of an HLA implementation as master algorithm for FMI has been proposed in [2]. [19] describes a method to create an HLA wrapper for FMI. Also, [14] demonstrates an approach to automatically integrate FMUs into one HLA co-simulation. However, this approach also requires quite some meta-models to be created. These meta-models would need to be updated when the models are changed, which we intend to do frequently.

For the design of CPSs, tools and methodologies exist that tackle the challenges inherent in these systems [7]. [8] proposes a model based approach to deal with these challenges. A heterogeneous simulator, the Ptolemy framework, was proposed in [6]. The DESTECS project [16] deals with co-modelling, co-simulation and co-designing supervisory control models [4]. The TERRA tooling [12], provides for co-simulating real-time control models using FMI and CPS [5]. [18] proposes a method to generate simulatable models from multi-domain functional descriptions of systems.

III. Case study: SliderSetup

A system which we call the SliderSetup serves as case study throughout this paper to illustrate our methodology The SliderSetup consists of two independent axes, each having a slider – encircled in blue in Figure 1 – that can be moved along its axis. One axis is located at the bottom while the other is positioned on top. A thread coil can be attached on each of these sliders. The goal of the SliderSetup is to unwind the threads from one of the coils onto the other coil by letting the sliders orbit around each other without colliding. From this behaviour, the following requirements are specified.

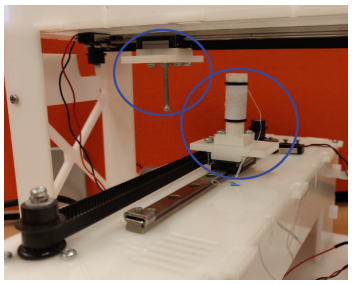

Figure 1. The SliderSetup with thread coils attached to the sliders.

1) The system is capable of (un)winding the thread coil at a minimum speed of 2 rotations per second.
2) The components of the system may not collide.

The SliderSetup consists of six components, which are displayed in the component architecture in Figure 2. Each slider is moved by a motor which is controlled by its own controller. These controllers are controlled by a supervisory controller. The supervisory controller receives its input from management software that is used by the user to control the behaviour of the system, such as moving the sliders to specified positions.

For the development of the SliderSetup, the collaboration of different disciplines is required. Both the management software and supervisory controller are developed by the software engineers, while the controllers and physical sliders are developed by the mechatronic engineers.

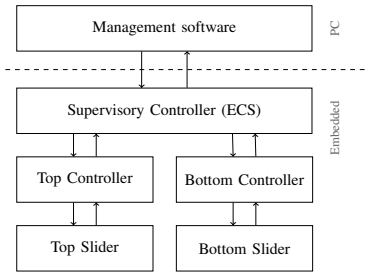

Figure 2. Overview of the component architecture of the SliderSetup.

IV. System design

For the development of the SliderSetup, the design method outlined in [4] is used. Figure 3 shows an overview of this design flow. When using this design flow, three parts are required: the plant dynamics, the control laws and the embedded control software (ECS). These three parts correspond with the components displayed in Figure 2.

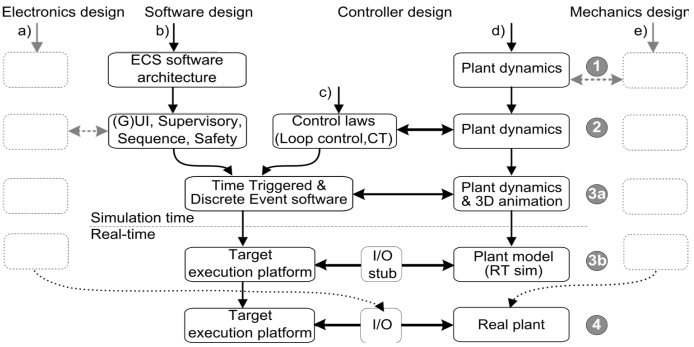

Figure 3. An overview of the design flow used for the SliderSetup [4], showing the different stages of the models and implementations.

To be able to construct a co-simulation of these three parts, the interfaces between them should be defined. Deciding on the interfaces requires information on the actuators and sensors that are present and the information that needs to be shared between component models. Although the exact implementations of these components of the system are determined by the system requirements and the available options, the interfaces between them can be decided on very early in the design process. This section briefly describes the design decisions made and the resulting interfaces and implementations for the three parts mentioned above.

A. Plant dynamics

Based on the speed requirements for SliderSetup, a belt was chosen to actuate the slider. This belt is driven by an electromotor, which is selected based on two factors. First, the velocity requirement of the slider requires a certain amount of torque. Secondly the motor should be easily controlled using available motor drivers. The selected motor and driver combination should be driven by a voltage and can be enabled and disabled. The sensor on the slider senses the position of the slider in a way that yet has to be determined. This results in the plant model having two inputs: a voltage and an enable

signal; and a single output: a sensor value representing the position. These are all connected to the controller.

B. Control laws

The control law of the system is largely based on the plant dynamics. Depending on the application, a certain control mode needs to be implemented. For the SliderSetup the decision was made to implement three different control modes:

- A fast control mode using linear control.
- A controlled move using a motion profile.
- A constant velocity mode.

These three modes require different inputs from the supervisory controller, which are converged into one set of attributes: a mode selector, a set-point input – either velocity or position – and a duration for the controlled move. Furthermore the controller outputs an estimated position of the slider to the supervisory controller.

C. Embedded control software

The initial design of the ECS is purely based on the inputs and outputs of the controller to which it is connected. It is designed to control two axes and supports default sequences of actions that should be executed, such as setting values for the setpoints and modes of the axes. The management software is ignored at this stage, since it is not part of the core functionality of the SliderSetup itself. Only when this functionality is sufficiently developed, this software is added.

V. Modelling

For stages one and two of the design flow as outlined in Section IV, three models are required: the plant dynamics, the control laws and the embedded control software (ECS). These models correspond to the sliders, controllers and supervisory controller in Figure 2 respectively. Each of these models starts as an abstract model and is gradually refined to improve the accuracy of the model. This section briefly describes the component models that are developed.

A. Plant dynamics

The model of the plant consists of two continuous-time models that are developed in 20-sim[1]. The models for both slider axes are functionally identical. They describe the behaviour of the motor, the driving electronics, the transmission to linear motion and the sensor. The plant dynamics also includes a 3D kinematics model, which can be used for collision detection. The plant dynamics model should eventually correspond to the slider axis that is realised. Each instance of this model will be connected to the corresponding controller.

B. Control laws

The controller model is a discrete-time model of software that is also developed in 20-sim. The model contains the control laws to control the axis as well as the processing required for the sensors. This model is eventually realised as a piece of software. The controller is connected with both the supervisory controller and the motor.

[1] http://www.20sim.com/

C. Embedded control software

The model of the ECS is a discrete-time model of the software that coordinates both controllers and provides a connection for the management software. It is developed using the Parallel Object-Oriented Specification Language (POOSL) [17], which is a discrete-time modelling language that is suitable for modelling software architectures. The supervisory controller controls the controllers and provides an interface for user control and status monitoring. The model should be realised into a software component that runs on an embedded board in the system.

VI. Co-simulation

This section briefly outlines the standards and techniques that are used for the construction of co-simulations.

A. Functional Mock-up Interface

Our methodology is primarily based on the use of the Functional Mock-up Interface [3] as default model format. The FMI is a standard interface that can be used for model exchange and model simulation by other tools. Consequently, the FMI standard is very suitable for co-simulation, although it still requires some master algorithm to synchronise time and attributes between the models participating in the co-simulation. The standard is widely supported by modelling tools such as 20-sim, Modelica[2] and Simulink[3].

B. High-Level Architecture

The High-Level Architecture [1] is a standard for co-simulation of models from different tools. It describes a software interface that participating models should comply to as well as a set of rules that ensure proper simulation behaviour. An implementation of HLA includes a Run-Time Infrastructure (RTI) that coordinates the co-simulation. Within HLA, a co-simulation is called a *federation* while a single simulation is called a *federate*. Every co-simulation – including models and their attributes – is described in a configuration XML. Different implementations of HLA are available, both commercial and open source.

C. CoHLA

Our methodology entails many small changes to the component models during the development process. Since HLA requires wrappers for the models to communicate with the RTI, these wrappers are changed frequently. Also, the configuration XML has to be adapted upon every change in a model.

Configuring HLA (CoHLA) is an open source[4] Domain Specific Language (DSL) that was introduced in [13]. CoHLA allows the user to easily specify a co-simulation of different models. From such a co-simulation specification, the previously mentioned wrappers and configuration files are generated, enabling the automatic construction of a co-simulation. CoHLA allows the user to specify federate classes for the

[2] https://www.openmodelica.org/

[3] https://www.mathworks.com/products/simulink.html

[4] https://github.com/phpnerd/cohla

models in terms of input and output attributes. Listing 1 displays an example of such a federate class specification.

```
FederateClass Axis {
  Type FMU
  Attributes {
    Input Boolean enable
    Input Real voltage
    Output Real encoder
    Output Real position
  }
  DefaultModel "SliderAxis.fmu"
}
```

Listing 1. Federate class specification in CoHLA.

Listing 2 shows a small example of a co-simulation definition in CoHLA. A co-simulation is defined as a set of model instances of classes that have been specified previously, together with a set of connections that specify how the attributes from these models are connected to each other. For convenience, the displayed definition is only a subset of the instances in the SliderSetup.

```
Federation SliderSetup {
  Instances {
    bAxis : Axis
    bController : Controller
    svc : SuperVisoryController
  }
  Connections {
    { bController.encoder <- bAxis.encoder }
    { bAxis.voltage <- bController.voltage }
    { svc.b_pos <- bController.encoder }
    { bAxis.enable <- svc.bottomEnable }
    { bController.mode <- svc.b_mode }
    { bController.setpoint <- svc.b_setpoint }
  }
}
```

Listing 2. Co-simulation definition in CoHLA.

From such specifications, all the necessary code for an HLA co-simulation is generated. The generated code includes a wrapper for each of the models, configuration files and a runscript to easily start the co-simulation. Currently, there is support for the open source RTI called OpenRTI[5], POOSL models and models compliant with the FMI standard. CoHLA also adds support for logging, design space exploration, fault injection and collision detection using 3D models.

Since CoHLA makes the construction of a co-simulation fast and easy, it stimulates the use of co-simulations at all levels of abstraction. This provides feedback on design decisions and early detection of integration issues.

D. CoHLA co-simulation of SliderSetup

To create a co-simulation of the SliderSetup using CoHLA, we have specified federate classes for all component models described in Section IV. These federate class specifications are similar to the specification displayed in Listing 1. From these class specifications, a co-simulation definition is created that is similar to Listing 2. The co-simulation consists of two axis simulations, two controller simulations and one supervisory controller. We also added a logger to be able to review the simulation execution afterwards and a collision detector federate to verify that the axes do not collide with each other.

[5]https://sourceforge.net/projects/openrti/

This co-simulation is executed using our initial set of abstract models. Based on the results of the co-simulation, we move on to the next step – refining the models – which is described in Section VII.

VII. System refinement

With the co-simulation it is possible to simulate the complete system using the abstract models. This system-level simulation can analyse behaviour of the system and can be used for designing system-level features. Given that these features are implemented in multiple components of the system, they require a simulation of the complete system to analyse their functionality. The benefits of being able to simulate in this early stage of the system development are illustrated in this section.

A. System level analysis

The models described in Section V correspond to the second stage of the design flow illustrated in Figure 3. Traditional co-simulation methods would require a merge of all software models into a single model or an implementation of the control software, which corresponds to the third stage of the design flow. Our approach allows the co-simulation of the separate software and controller models from the current stage. Thus, the interaction between the different parts of the system can be analysed in an earlier stage of the development.

The early detection of issues was also observed in the SliderSetup. As mentioned, the motion controller has multiple modes of movement: a fast travel, a controlled travel, and a speed mode. During early system-level analysis, the system would sometimes not respond to a controlled move. Simulation showed that when the controller changed mode and setpoint at the same time, or the setpoint before the mode change, that the mode change would not be processed correctly, resulting in a very slow movement, or no movement at all.

Due to the early detection, the supervisory control could be changed to always change the mode before changing the setpoint. If this error was only detected later in the design process we expect it to take more effort to change the control structure to guarantee this timing.

B. Feature co-design

Adding more detail or new features to the system may require changes in multiple sub-systems. When these changes are implemented using the co-modelling approach as posed in [15], the updated components can be co-simulated to validate the new feature. This allows for a shorter feedback loop on the consequences of the change.

An example of this process is the design of the sensor system for the SliderSetup. The sensor available on the motor of the setup is an absolute rotation sensor. However, as the motor turns multiple times during the slider movement, the position of the slider carriage is not exactly known. Thus it was decided to implement a limit switch on one side of each axis. Consequently, when the carriage hits the limit switch, its position is known.

To implement this, all sub-models required changing. The switch itself was added to the plant model, as well as the aliasing/modulus behaviour of the motor position sensor. This also scaled the signal of the position to be in rotations instead of in distances. In the controller, the new sensor value needed to be interpreted correctly. This included scaling the value, "unwrapping" the motor aliasing, and the processing of the limit switch. It also required an extra connection between the controller and the plant model. In the supervisory controller, initialisation behaviour needed to be implemented. An axis should be initialised before it is used, by moving to the limit switch slowly until it is touched.

After adapting these changes in the models, the updated system could be analysed using the co-simulation. This allows the initialisation procedure to be tested, thus validating the new homing behaviour. Furthermore this validation showed a flaw in the implementation of the control laws, as the correction of the position due to the limit switch was interpreted as a large change in position, and thus in velocity. This change resulted in an (in)appropriate correction by the controller. While this was not a problem in the simulation, this could have been damaging when testing on the real hardware. In this case the flaw was detected quickly and could be fixed appropriately.

VIII. Realisation

From the last set of models that is co-simulated, the step towards realisation is small, since these models already contain a lot of details. The physical SliderSetup is built using rapid prototyping techniques based on the 3D models, which have also been used for the collision detection. For the motors and driver electronics a Raspberry Pi 3 is used in combination with a driver board that is capable of driving both motors and interfacing with the sensors. IO drivers for this board are already implemented in a Yocto Project[6] image for the Raspberry Pi 3. It also functions as the embedded computer. The other component models are implemented together as one software component that runs on the embedded board.

The embedded software consists of two parts, each matching one of the models. Since 20-sim is able to generate C++ code from a model, this will be used to generate the code for the control loop. The remainder of the software consists of a JSON socket interface that interacts with the control loop together with the hardware interface. This software must be developed manually as there is no direct model to code transformation available.

This is, however, not difficult, as the POOSL model already shows the structure of the software. The logic itself is basically a one-to-one translation from POOSL to C++, which is quite simple and might even be automated in the future. Implementation of the software can be split up into three steps.

1) Implementation of the interface to the management software. This can be tested without requiring the embedded board.
2) Implementation of the control loops. This requires the software to be tested on the Raspberry Pi instead of testing it on the development system.
3) Implementation of the interface to the hardware components. This step requires all hardware to be in place to properly test all software functionality.

After assembly of the system and installation of software, the system was up and running. The resulting system is shown in Figure 1.

IX. Results

The realised system functions as expected and can be controlled using the management software that is connected to the embedded board. The system initialises correctly and moves with sufficient velocity, meeting the first requirement. The initialisation procedure is chosen to minimise the chance of having the two axes collide. This behaviour was verified using the collision simulator, meeting the second requirement.

As a result of the design flow that was used, a set of models with different levels of detail has been developed. An overview of these different iterations of models is shown in Table I, showing the stages of the design flow, the features and details added, and how the models have been simulated (individually or as co-simulation).

Stage	Added features or details	Simulatability
1a	Kinematics of sliders	Individual
1b	Transmission	Individual
2a	Motor dynamics, control law	Co-simulation
2b	Sensors, initialisation	Co-simulation
3	Code implementation, IO drivers	Runnable
4	Hardware implementation	Physical setup

Table I
AN OVERVIEW OF THE DIFFERENT STAGES IN THE DESIGN FLOW.

The results of the simulations of the stages have been compared to each other and to the real motion of the system. This comparison is presented in Figure 4, showing the response of the SliderSetup based on a step input of 5 cm. The ideal motion is based on the velocity required to meet the first requirement. It is assumed that the system accelerates to and decelerates from this velocity instantly. When the motor dynamics have been added (stage 2a) the system responds slower then expected. After the sensor processing is added to the system (stage 2b), the controller of the system is tuned to reach the fast motion again.

Although the physical SliderSetup has a slight overshoot, it responds to the setpoint change sufficiently fast. The difference between stage 2b and the physical setup is due to a parameter mismatch between the simulated plant dynamics and the real plant. To reduce this mismatch, the physical plant could be measured to identify its parameter values, which could be used to improve the simulation accuracy. Since we only intend to illustrate the method, this step is left out of the example.

The realisation of the software is rather straightforward. The software structure is already defined in the POOSL model and after each of the three steps specified in Section VIII a working piece of software could be tested. Throughout

[6] https://www.yoctoproject.org/

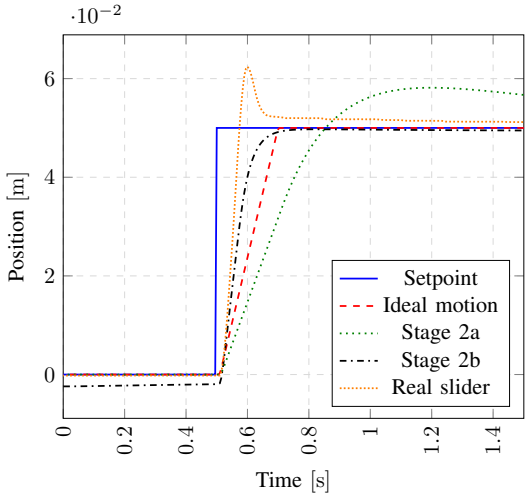

Figure 4. The step response of the simulations and physical SliderSetup to a fast move command to move the slider from 0 to 5 cm.

the implementation, testing of the software requires more hardware components to be connected, which fits in the development process. Because of the hardware that is required, the simulatability of the software is then noted as 'Runnable' in Table I. Due to the fact that the implemented software has the same connections and functionality as the model of the software, the realised system behaves as could be expected from the co-simulations that were executed during the design process.

X. Conclusion

In this paper we have shown how co-simulation and multi-level component modelling support the design of a cyber-physical system. Being able to simulate the models on system-level in an early stage of the development provides the opportunity to detect potential integration issues in both the field of hardware and software. Consequently, it requires less effort to address these issues compared to when they are detected in later stages. This would be the case when more traditional co-simulations approaches are used.

We illustrated this by developing a CPS using this approach. Here, we built a virtual prototype in an early stage of development, which allowed us to tackle some integration issues early. The approach also facilitates the concurrent development of component models. Once an interface between the components has been defined, our approach makes it easy to simulate them together, allowing the models to be developed rather independently from each other.

After having realised the system, we found that the behaviour as simulated by the models was accurate enough to make design decisions during the development process.

For future work it is interesting to improve the way the software model is implemented. Although some parts of the software were already generated, some implementation work was done manually, that might have been synthesised by using code generation. The system level behaviour was now analysed using manual evaluation of the simulation results. Future research could determine methods to automatically analyse and test system behaviour.

References

[1] IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Framework and Rules. *IEEE Std 1516-2010*, pages 1–38, Aug 2010.

[2] M. U. Awais, P. Palensky, A. Elsheikh, E. Widl, and M. Stifter. The high level architecture RTI as a master to the Functional Mock-up Interface components. In *2013 International Conference on Computing, Networking and Communications (ICNC)*, pages 315–320. IEEE, 2013.

[3] T. Blochwitz, M. Otter, et al. The Functional Mockup Interface for tool independent exchange of simulation models. In *8th Modelica Conference*, pages 105–114, 2011.

[4] J. F. Broenink and Y. Ni. Model-driven robot-software design using integrated models and co-simulation. In *Embedded Computer Systems (SAMOS), 2012 International Conference on*, pages 339–344. IEEE, 2012.

[5] J. F. Broenink, P. J. D. Vos, Z. Lu, and M. M. Bezemer. A co-design approach for embedded control software of cyber-physical systems. In *2016 11th System of Systems Engineering Conference (SoSE)*, pages 1–5, June 2016.

[6] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. 1994.

[7] P. Derler, E. A. Lee, and A. S. Vincentelli. Modeling cyber–physical systems. *Proceedings of the IEEE*, 100(1):13–28, 2012.

[8] J. C. Jensen, D. H. Chang, and E. A. Lee. A model-based design methodology for cyber-physical systems. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, pages 1666–1671. IEEE, 2011.

[9] G. Karsai and J. Sztipanovits. Model-integrated development of cyber-physical systems. In *IFIP International Workshop on Software Technolgies for Embedded and Ubiquitous Systems*, pages 46–54. Springer, 2008.

[10] P. G. Larsen, J. Fitzgerald, et al. 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)*, pages 1–6, April 2016.

[11] P. G. Larsen, J. Fitzgerald, J. Woodcock, C. Gamble, R. Payne, and K. Pierce. Features of Integrated Model-Based Co-modelling and Co-simulation Technology. In *Software Engineering and Formal Methods*, pages 377–390. Springer, 2018.

[12] Z. Lu, M. M. Bezemer, and J. F. Broenink. Model-Driven Design of Simulation Support for the TERRA Robot Software Tool Suite. In *Communicating Process Architectures 2015, Canterbury, UK*, pages 143 – 158, England, Aug. 2015. Open Channel Publishing Ltd.

[13] T. Nägele, J. Hooman, T. Broenink, and J. Broenink. CoHLA: Design Space Exploration and Co-simulation Made Easy. In *IEEE 1st Industrial Cyber-Physical Systems (ICPS 2018)*, pages 225–331, May 2018.

[14] H. Neema, J. Gohl, et al. Model-Based Integration Platform for FMI Co-Simulation and Heterogeneous Simulations of Cyber-Physical Systems. In *Proceedings of the 10th International Modelica Conference*, number 96, pages 235–245. Linköping University Electronic Press; Linköpings universitet, 2014.

[15] Y. Ni. *System Design Support of Cyber-Physical Systems, a co-simulation and co-modelling approach*. PhD thesis, University of Twente, Enschede, Netherlands, June 2015.

[16] K. G. Pierce, C. J. Gamble, Y. Ni, and J. F. Broenink. Collaborative Modelling and Co-Simulation with DESTECS: A Pilot Study. In *3rd IEEE track on Collaborative Modelling and Simulation, in WETICE 2012*, pages 280 – 285. IEEE-CS, June 2012.

[17] B. D. Theelen, O. Florescu, et al. Software/Hardware Engineering with the Parallel Object-Oriented Specification Language. In *5th Conference on Formal Methods and Models for Codesign*, MEMOCODE '07, pages 139–148. IEEE Computer Society, 2007.

[18] J. Wan, A. Canedo, and M. A. Al Faruque. Functional model-based design methodology for automotive cyber-physical systems. *IEEE Systems Journal*, 11(4):2028–2039, 2017.

[19] F. Yilmaz, U. Durak, K. Taylan, and H. Oğuztüzün. Adapting Functional Mockup Units for HLA-compliant distributed simulation. In *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*, number 096, pages 247–257. Linköping University Electronic Press, 2014.