ABSTRACT
Continuous development of software functions using software-in-the-loop (SiL) simulations up to their virtual validation or virtual release are key enablers for new technologies of autonomous and connected mobility. Due to the complexity of autonomous and predictive driving functions, the unit-under-test is often not encapsulated in one Electronic Control Unit (ECU), but distributed over multiple ECUs. Thus, a cross-domain vehicle simulation framework capable of simulating numerous connected virtual ECUs (vECUs) together with the corresponding physical models (e.g. vehicle dynamics) is required. In this contribution, this is realized by extending an existing MiL co-simulation environment consisting of a middleware-based coupling of proprietary tools with a toolchain for the generation of vECUs as Functional Mock-up Units (FMU) and a modular virtual bus implementation. This framework is tested by showing a smooth transition from model-in-the-loop to SiL simulation of a function distributed over two vECUs communicating via a virtual Controller Area Network bus.

Keywords: virtual validation, virtual ECUs, virtual automotive bus, FMI, Middleware

1 INTRODUCTION
Mobility is on the way to be autonomous and connected. The more autonomous functions a vehicle exhibits the more control loops are closed leading to a tighter coupling of the vehicle subsystems (here also called vehicle domains), e.g. powertrain, longitudinal dynamics, lateral dynamics and so on. Consequently, these vehicle subsystems and especially their controls can no longer be developed independent from each other. In fact, engineers have to not only consider the domain they are experts in but also all relevant other domains.
While in former times late integration led to suboptimal results (Plateaux et al. 2009), for upcoming systems early integration is a requirement. In other words, development of those functionalities requires cross domain methods. Moreover, due to the open context nature of the problem, real testing is not sufficient resp. possible anymore. Hence, simulation has to be used for development and validation, preferably with a smooth transition in between. Thus, cross domain vehicle simulation becomes an integral part of vehicle function development.

Furthermore, autonomous and connected functions demand for new validation concepts. The open context nature of the problem leads to impractical time spans for validation if conventional approaches are used (Pütz et al. 2017). In literature numbers of up to several billion test kilometers can be found (Wachenfeld and Winner 2015). Clearly, those amounts can not be driven on a test track or using hardware-in-the-loop (HiL) setups. Note that, parallelization is in principle possible with HiL setups but much too expensive to be practical. This is why a big trend towards virtual validation, that is conducting tests in pure virtual environments, can be observed.

Frameworks for virtual development and virtual validation of connected and/or autonomous functions need to be frameworks for cross domain vehicle simulation. Thus, models from the concerned vehicle domains have to be integrated into a cross domain model. Since especially in big companies systems of the different vehicle domains are engineered with varying simulation tools, the method of choice is often co-simulation (Gomes et al. 2018). That means the existing models of the different vehicle domains in the corresponding simulation tools are integrated into a cross domain setup. The concept of co-simulation also allows the coupling of black-box models which are prevalent in automotive industry for IP protection reasons. Besides the physical models of the vehicle, also the software functions have to be integrated. In early development phases these are mostly prototypes (model-in-the-loop (MiL)) and many effects are neglected, e.g. quantization due to data type conversions as well as rate discretization and delays by cause of communication. In later phases of the development process virtual Electronic Control Units (vECU) come into play. Here, the target application software (ASW) is used in simulation (software-in-the-loop (SiL)). By doing so, functional tests of the code can be performed (Jaikamal 2009). To increase test depth further, original basic software (BSW) can be used besides application software. If also the communication between vECUs is simulated, the communication-related effects mentioned above can be investigated using SiL.

In order to perform virtual validation with a SiL framework it should be integrated in the development process. In (Von Wissel et al. 2018) it is shown how the SiL integration platform Silver (tool from QTronic) can be integrated into an automated build process for engine control software. In (Kaijser et al. 2018) a workflow for this model based continuous integration is presented.

A topic that has gained less attention so far is the interconnection of several vECUs for SiL simulations. In (Schiller and Knoll 2015) a framework for network emulation for the communication between different vehicles is presented. This framework shall at least allow the extension to in-vehicle communication. However, it is developed from the network point view and thus (currently) lacking the support of open simulation standards such as Functional Mock-up Interface (FMI) (Blochwitz et al. 2012), which makes it hard to connect that framework to common simulation tools.

Nevertheless, connected and autonomous driving functions are often distributed over several ECUs. During virtual development, when abstracting the function and neglecting the communication delays (if possible), the transition from one ECU to several ECUs is not critical (Mikelsons and Samlaus 2017). For SiL simulation and especially virtual validation things are different. In that case there are simulation tasks (e.g. functional integration tests), where all relevant ECUs shall be virtualized including the basic software and moreover the vECUs shall communicate with each other using a virtual bus with correct bus behavior. This results in an architecture exemplary sketched in Figure 1. Here, the domains vehicle dynamics, brake system, steering and powertrain are integrated in a co-simulation setup together with sensor and
environment models. Note that the control of the steering system and the powertrain is directly integrated into the co-simulation setup as mockups (or MiL). In contrast the vehicle computer (for higher level functions like object recognition and/or motion planning) and two ECUs controlling the brake system are virtualized and communicate with each other via a virtual CAN bus. Communication between vECUs and the co-simulation setup has also to be possible for signals that are not transmitted via a bus in the real setup, e.g. analog I/Os.

Figure 1: Exemplary virtual validation framework architecture with two vECUs for braking control and a co-simulation setup inside the big black box

This contribution presents a method to set up such a framework using FMI as an open standard. In particular FMI is used for the integration of the vECUs as well as for the integration of the virtual bus. In both cases, FMI serves as the interface for timing and signals from the plant models. The approach is presented using a case study, i.e. recuperation control of an electric vehicle using two vECUs communicating via a virtual bus.

The remainder of the paper is structured as follows. In section 2 the vECUs, the used virtual bus and their provisioning as FMUs is described. The case study is presented in section 3. The contribution closes with a conclusion, open questions and challenges.

2 INTEGRATION OF COMPLETE SOFTWARE AND VIRTUAL AUTOMOTIVE BUSSES INTO CLOSED-LOOP SIMULATION

vECUs integrate the complete ECU software into simulation. The vECU approach aims at running target ECU code on standard x86 systems by virtualization of hardware-dependent parts. This is either achieved by running original code, which has compiled for the target platform and is executed using hardware simulators, or by replacing hardware specific parts with corresponding simulation models that allow compilation for x86 platforms. The former approach enables highly accurate simulations (Bücs et al. 2016), but requires hardware models, which are not always available and, if available, typically computationally expensive. The second approach neglects detailed hardware behavior of the ECU by integrating the ECU code with simplified simulation models for target specific parts. This approach is faster, but less accurate.

2.1 Interfaces of the vECUs

Figure 2 illustrates vECU coupling possibilities that allow realization of different use cases. The XCP protocol was developed to measure and calibrate parameters of the ECU software (Lemon 2003), e.g. to...
optimize the gasoline injection for a certain engine type. Test APIs furthermore enable automatic testing of the ECU software. Examples for commonly used testing tools are TPT (Bringmann and Krämer 2008) and ECU-TEST. The support for higher-level languages like Java enables rapid test case development for application software. Virtual communication busses (e.g. CAN, LIN, ...) enable the connection of multiple application models via simulated bus models for simulation of distributed control loops. Depending on the level of detail of these bus models realistic delay and packet loss can be introduced. Therefore, engineers can use such a virtual setup for early detection of misbehavior due to platform resources. Additional tools like CANoe or Busmaster enable rest-bus simulation by mocking additional ECUs in a network. The integration of vECUs in simulation frameworks can be realized via FMI.

![Figure 2: Interfaces of vECUs](image)

2.2 Categories of vECUs

The Automotive Open System Architecture (AUTOSAR) allows different portions of the ECU software to be put into a vECU. Each of these cuts through the ECU software defines a vECU category. Such categories are required since, as always in modeling and simulation, it is not recommended to use the most detailed model for every kind of simulation. Consequently, in pure functional simulations that only evaluate the functional integration of applications without resource restrictions, typically vECUs only contain the ASW and a modified RTE to link functions. More detailed tests (including communication latency etc.) need to include complex device drivers, BSW, or MCAL layers to interface with the functions under test. Virtual ECUs are therefore classified into three categories:

1. vECUs that contain only the application software (ASW), the run-time-environment (RTE) and optionally an operating system. This aims at quick testing of basic functions and ASW integration without using base software (BSW) components. Using the AUTOSAR platform, it is easy to create vECUs for this use-case. However, no realistic estimation of the execution behavior on real ECUs can be derived with this kind of vECUs. They are typically used for functional tests without resource restrictions.

2. vECUs that consists of ASW, RTE, BSW, operating system and a virtual Microcontroller Abstraction Layer (MCAL) for x86 systems. A more realistic behavior of the real ECU can be simulated with this kind of vECU. The scheduling of tasks, as well as communication timing is considered, and the functionality of the BSW can be tested. Typically, such vECUs are used to perform functional integration tests of a network of ECUs.

3. If a more realistic behavior of the vECU is desired, a detailed simulation of the ECU hardware (virtual hardware) is necessary. By using such virtual hardware, all software components of the real
ECU can be tested. The MCAL for the target platform connects to detailed hardware models and enables simulation of real timing behavior. Another benefit is the ability to perform fault injection testing. This is problematic when done with real hardware, since the injected faults could cause damage to the devices. A drawback of using hardware models is significantly reduced simulation speed due to simulation model complexity. These kind of vECUs is typically used for BSW development.

For the case study in section 3 vECUs of category 2 are used. Since the ECU hardware is not modeled (no virtual hardware) in those vECUs, it cannot be assumed that the execution times of the AUTOSAR runnables in the simulation are the same as for the real ECU. Therefore inter-runnable dependencies are not correctly simulated and must be validated separately. But it is assumed that the here presented functional tests of interconnected vECUs can be done independently from the validation of the inter-runnable dependencies for which a corresponding timing simulation (Krause et al. 2007) is needed.

2.3 Integrating vECUs into Co-Simulation Environments

In order to use vECUs in simulation environments, a FMU generator has been developed. The generator is integrated into the vECU build toolchain and creates a FMU based on generated build artifacts. The user only has to manually define inputs, outputs and configuration parameters for the communication between the FMI interface library and the vECU’s interface. Since FMI does not support bus systems, the drivers of the vECU have been adapted for direct connection to the virtual bus presented in section 2.4.

2.4 Virtual Bus

An automotive virtual bus simulates bus behavior and integrates different kinds of vECUs and simulation models into one system. The used virtual bus exhibits a modular architecture, where each module simulates a specific part of the communication stack. We use simulator coupling for simulation time synchronization of our modules as described in (Kuhr et al. 2013). Different levels of detail are realized by employing different combinations of these modules. Figure 3 illustrates this architecture and three possible interfaces for connecting vECUs on three different levels of abstraction.

Figure 3: Architecture of bus simulation

The network interface component is instantiated once per simulated device. It consists of a transmit queue, and the medium access control (MAC) protocol of the simulated bus. The transmit queue implements the queuing behavior, e.g. it decides if specific messages get priority or overwrite each other. MAC compo-
Baumann, Samlaus, Mikelsons, Kuhn, and Jahic

...elements of all network interfaces that belong to the same bus simulation connect to the same instance of the corresponding medium. The medium component simulates the physically shared communication medium. Simulation components (i.e. models of vehicle domains) and vECUs connect to the virtual bus via interfaces on different levels of abstraction, e.g. depending on their own implementation:

1. **Functional Layer (High-level Application)**, e.g. FMUs only communicate on a functional level of abstraction. From the point of view of the virtual bus, they transmit payload data, which are individual functional signals without any encoding. During deployment to concrete communication systems, multiple functional signals are usually multiplexed into the same protocol frame and factors and offsets are applied to ensure that only the relevant part of the functional signal is transmitted for efficiency reasons. The virtual bus provides transmitPDU/receivePDU functions to connect high-level models. Function transmitPDU(payload:byte[], size:int, id:String) transmits PDU payload with defined size and a network specific ID to the En-/Decoder component. The ID is for example the message ID type in CAN bus networks, or the destination MAC address in Ethernet networks.

2. **Protocol Layer (Intermediate-level Application)** Coupled components which exchange protocol frames with the virtual bus are called Intermediate-level Application. Since a hardware abstraction layer (HAL) is present, these frames already contain bus specific protocol header data. Intermediate level models directly access the queue with transmitted and received frames to/from the virtual medium. These frames consist of complete protocol frames with all header information. Intermediate-level models do not implement the low-level transmission behavior of the platform base software (BSW). Simulation of queue semantics is therefore provided by the virtual bus inside the network interface. The application uses the add(frame: byte[]) function to place complete protocol frames into the transmit queue.

3. **Driver Layer (Low-Level Application)** Coupled components which transmit protocol frames and also have a transmission queue implemented in the HAL are called Low-Level Application. Consequently, a low-level interface between the virtual bus and the vECU under test is necessary. Communication between low-level models is realized via getNextMessage(id:String) callback. This callback is invoked by the MAC component to get the next frame for transmission. The parameter indicates whether a frame with a specific ID is requested, or if any ID is accepted. Furthermore, MAC layers also may request frames with highest or lowest IDs.

In case of a CAN bus, the virtual bus simulates the CAN bus communication on frame level using a discrete event Model of Computation and Communication (MOCC). This MOCC puts a focus on relevant communication events, while it skips time intervals without waiting events. Discrete events in different simulation components are globally ordered. As illustrated in Figure 3, the virtual CAN bus is split into network interface and a shared medium. The network interface consists in most cases of a queue for outgoing transmissions and the CAN MAC. All network interfaces (unless multiple CAN busses are present) connect to the same CAN medium. The CAN bus queue stores messages that wait for transmission. These messages are ordered by the message priority, which is the message CAN id. If multiple messages with the same CAN id are waiting, the queue supports two operation modes: either all messages are stored using FIFO principle, or only the most recent value for a message ID is stored, and older values are overwritten. The queue furthermore decouples application models from the CAN MAC, which requires a tight and timely accurate interaction with its transmission buffer. Low-level applications therefore require tight couplings with the communication interface. These couplings are expensive with respect to simulation speed. The virtual bus thereby reacts on time-ticks received from the offered FMI interface.

The CAN MAC component simulates the behavior of a CAN controller. It periodically monitors the medium. When the medium gets idle, the MAC component checks its queue for the next waiting highest-priority frame. Every MAC controller that receives a waiting frame from the queue actively participates in
medium arbitration to identify the waiting frame with highest-priority. The CAN medium arbitration simulates the first two bits on the medium, which are SOF and the first arbitration bit. Subsequent arbitration bits are skipped; instead, every MAC that participates actively in the arbitration transmits its frame to the medium component. This component evaluates all frame priorities and notifies the MAC components about which frame did win the arbitration, as well as about the end of a transmission. Afterwards, the cycle is repeated as long as queues still contain CAN bus frames.

3 CASE STUDY

This case study shows the seamless transition from a MiL simulation to a SiL simulation with multiple vECUs where the functionality and configuration of the tested functions is identical. This way, SiL testing with multiple vECUs, including the investigation of bus related effects as communication delays and quantization errors, can be done in simulation reusing MiL setups, which shortens and cheapens the development cycle. The transition from MiL to SiL achieved by coupling multiple vECUs is a necessary step for the virtual validation of cross domain functions since here effects such as communication delays between ECUs and quantization errors in the coupling signals have to be taken into account for certain validation tasks.

The remainder of this section is structured as follows. First, the cross domain vehicle model is introduced. It contains two driving functions for which the transition from MiL to SiL is carried out in the following. Afterwards the simulation results of the MiL and SiL models are compared and bus related effects are discussed.

3.1 Cross Domain Vehicle Model (MiL)

Starting point is a cross domain co-simulation model of an electric vehicle coupled to a microscopic traffic simulation, which is used at Bosch to study the energy consumption in certain traffic scenarios (Boumans et al. 2019, Baumann et al. 2019). Since the vehicle model contains various submodels from different tools, a middleware based architecture for the co-simulation was chosen and a coupling configuration using advanced coupling schemes was developed (Benedikt et al. 2013). This means that every submodel is connected to a middleware (here: Model.Connect from AVL), which coordinates a numerical correct data exchange between submodels during simulation. Next the most important subsystems are roughly explained.

A CarMaker (tool developed from IPG) model contains the vehicle dynamics as well as a generic CarMaker adaptive cruise control function for the longitudinal control of the vehicle. Since the CarMaker driver is not able to change lanes strategically on a multi-lane road, a Bosch software function contains the lane change logic and also controls the lateral movements of the vehicle. Other control functions as the battery management and the recuperation strategy are combined in a Simulink model called "Controls". The brake actuation control is modeled in a separate Simulink model. To create test scenarios in a microscopic traffic environment the open source tool "Simulation of Urban MObility" (SUMO) (Behrisch et al. 2011) is also coupled to the middleware. The coupling interface between SUMO and Model.Connect was developed to control the traffic simulation and to exchange data during simulation. With this setup the virtual vehicle drives autonomously on a highway under realistic traffic conditions. Figure 4 shows a screenshot taken from CarMaker during the simulation.

3.2 Transition from MiL to SiL using vECUs

For the transition from MiL to SiL two strongly connected driving functions are chosen as units under test. One is a driving strategy function, more precisely the recuperation control of the vehicle and the other is the actuation control function of the friction brake. The other control functions (e.g. powertrain control) remain
as MiL models, because there is no direct dependency between them and the units under test. Generally speaking, it is up to the engineer to decide which ECUs have to be virtualized and for which a MiL model is sufficient. This is a trade off between accuracy and complexity of the co-simulation.

For each unit under test a vECU is generated. The vECUs belong to category 2 (explanation in section 2.2) and include the ASW and RTE as well as a BSW and the necessary operating system and driver. In section 2 the tool chain for generation and provision of the vECUs as FMUs is described. It has to be mentioned that recuperation control as well as the brake actuation ASW are simplified and do therefore not contain series code. But, the feasibility of the MiL to SiL transition can be shown in this case study although it is not a real world use case. The integration of series ASW is part of current work.

The bus simulation model introduced in section 2.4 interconnects the two vECUs, and enables a bilateral data exchange via a virtual CAN bus. The virtual bus is synchronized to Model.Connect via FMI to ensure that the virtual bus reacts to the timing-ticks of Model.Connect. This way, the simulation times of all components of the framework are synchronized. Since a simplified ASW is used, the bus configuration is done manually. Configuration of the virtual bus using dbc-files or AUTOSAR files is part of current work and goes inline with the use of series ASW. The data exchange between the vECUs and the other subsystems of the co-simulation model is done using the FMU interface of Model.Connect. The final architecture of the cross domain co-simulation model with the vECUs and the CAN bus simulation is shown in Figure 5.

Figure 4: Ego vehicle (red car) before (left) and after (right) an overtaking maneuver on a highway (source: IPG Movie)

Figure 5: Middleware based model architecture using Model.Connect from AVL. The name of the tool in which the corresponding submodel was created is in parenthesis, SL is the abbreviation for Simulink (from Mathworks) and CM for CarMaker (from IPG). The first green bubble indicates the vECUs introduced in section 2 and the second bubble marks the virtual CAN bus presented in section 2.4.
3.3 Discussion of Simulation Results

3.3.1 Driving Task

The investigated driving task of the electric vehicle is an overtaking maneuver on a highway with three lanes. The ego vehicle starts on the rightmost lane and tries to overtake a slower driving vehicle. Because the middle lane is not free, the ego vehicle has to reduce its speed by braking before it can overtake safely. The recuperation control function divides the demanded brake torque into a recuperation torque and a torque which must be absorbed by the friction brakes. The brake actuation control function distributes this torque on the four friction brakes.

3.3.2 Comparison MiL and SiL

To investigate the differences between the MiL and the SiL system the driving task was driven with both setups using exactly the same configuration e.g. macro step sizes. Figure 6 shows a comparison between one of the four brake actuation torques, which is an output of the brake actuation control function. Since this signal is calculated by a vECU in the SiL setup and also directly depends on a signal which is exchanged on the virtual CAN bus it is very suitable for a comparison. By closely looking at the brake torque, a small deviation can be seen. The discrepancy is so small, that it has hardly any effect on the maneuver of the ego vehicle itself, as can be seen in the velocity curve in Figure 7. Because the virtual bus is the only difference between the two systems, the deviation in the signal can be attributed to bus related effects as communication delays and quantization errors. Both effects are discussed in the following.

Communication Delays  For the investigation of time delays it has to be distinguished between two groups of delays. On the one hand synthetic delays that occur in simulation due to the architecture and the execution scheme of the co-simulation and on the other hand communication delays which are introduced
in the SiL setup due to the CAN bus simulation. The synthetic co-simulation delays of the MiL setup are investigated first.

Here, our two functions of interest are modeled in two different Simulink models, which are executed in parallel. Therefore, if the recuperation control demands a brake torque at time $t_k$, it takes a macro time step $t_h$ until the brake actuation function can forward this demand to the brakes at time $t_k + t_h$. It follows a synthetic time delay of $t_h$ seconds between the two functions in the MiL case.

In the SiL systems the functions are integrated in two separate vECUs, which have their own scheduling. The cycle of reading the bus, doing calculations and writing on the bus again, happens every $t_h$ seconds. This is the smallest possible time delay which occurs during communication from one vECU to the other one, because the two vECUs do their calculation in parallel and are synchronized to each other. But the communication delay introduced by the virtual CAN bus $t_b$ also has to be considered. In a CAN bus the communication delay depends on the bit-rate of the bus as well as the bus load which in turn depends on the number of signals transmitted via the bus. In the proposed SiL setup only one signal is transmitted via the bus. Therefore it applies $t_b << t_h$. Consequently, the time $t_h$ between two vECU evaluations is long enough to transmit the signal via the virtual CAN bus without introducing additional communication delays. In summary, the comparison of the time delays in the MiL and SiL system shows, that they are equal, although they have different causes. The time delay is therefore not responsible for the deviation in the brake torque which was seen in Figure 6. Note that the harmonization of both delay types is part of future work.

Quantization Effects CAN messages transmitted via a CAN bus contain up to 64 bits of data (Zimmermann and Schmidgall 2006). Therefore, in order to keep the bus load low, signals that are exchanged via a CAN bus are usually quantized in such a way that they consume as little bit of data as possible. This way several signals can be transmitted in the same CAN message. It follows that a quantization is a compromise between accuracy and required amount of data for transmission. The brake torque demand, which is transmitted via the virtual CAN bus in the introduced SiL setup, uses 16 bit of data of CAN message. This allows an accuracy of the signal of roughly 0.2 Nm. Though this is more accurate than in most real world applications, this is significantly less accurate than the accuracy which is achieved by exchanging the signal as a floating point number as it is done in the MiL setup. The resulting quantization error leads to the deviations of the brake actuation torque seen in Figure 6.

This case study presents an approach for SiL simulation of a distributed function. Thereby, this function is integrated as two vECUs and can be tested in a virtual traffic scenario, using a cross domain vehicle simulation as boundary model. By using the BSW in the vECUs and the virtual bus, all software and bus related effects (e.g. bus delays and quantization) can be investigated virtually. This allows for example for functional integration tests. The hardware of the vECUs is not modeled, since its simulation requires high computational effort and its effects can usually be examined without a complete system (i.e. vehicle) simulation. Depending on level of detail and reliability of the boundary models, certain validation tasks can be fulfilled with this approach.

4 CONCLUSION AND OUTLOOK

This contribution presents a simulation framework capable of realizing a smooth transition from MiL to SiL simulation including bus simulation. The virtual bus models as well as the vECUs are integrated via an open standard, i.e. FMI. The approach has been validated using vECUs with series BSW, but simplified application software. In a next step the approach will be tested in a real world (in particular series application software) example. By doing so, the bus load will increase due to the increased number of signals that have to be transmitted. This demands for coupling of the virtual bus models to common rest bus simulation and analysis tools (e.g. CANoe from Vector informatics), which is already part of current work.
In future work the co-simulation delays and software related timing effects will be harmonized. While latter are explicitly modeled and include the performance behavior of the AUTOSAR software (Denil et al. 2017) as well as communication delays, the former are an unwanted effect of co-simulation. Currently, both are not harmonized, which means that co-simulation increases the modeled communication delays. Note that, for this harmonization the co-simulation semantics (e.g. gauss seidel or jacobi) has to be taken into account and also signals semantics must be adapted, if e.g. a sensor is not explicitly modeled. Moreover, the integration of the approach in continuous integration processes and massive parallelization will be investigated.

REFERENCES


**AUTHOR BIOGRAPHIES**

**PETER BAUMANN** is a Ph.D. student at the University of Duisburg-Essen. He holds a Master of Science in Engineering Cybernetics from the University of Stuttgart. His research interests include spatially distributed co-simulation. His email address is peter.baumann5@de.bosch.com.

**ROLAND SAMLAUS** is an employee of Robert Bosch GmbH. He holds a diploma in Computer Science from the technical University of Dresden and a Ph.D. of Computer Science from the University of Linköping. His current fields of activities include simulation environments for virtual validation of automated driving functions. His email address is roland.samlaus@de.bosch.com.

**LARS MIKELSONS** is professor for Mechatronics at the University of Augsburg. He holds a diploma in Mathematics and a Ph.D. in Engineering, both from the University of Duisburg. After his dissertation he worked in industry in the field of virtual engineering in the automation and automotive domain. His current fields of activities include virtual validation, mixed real/virtual prototypes and hybrid (mixed data and physical) modeling. His email address is lars.mikelsons@informatik.uni-augsburg.de.

**THOMAS KUHN** heads the Embedded Systems division at Fraunhofer Institute IESE in Kaiserslautern. He holds a diploma in computer science and a Ph.D. from University of Kaiserslautern. His research focus is virtual engineering and the aggregation of simulation models into digital twins. His email address is thomas.kuhn@iese.fraunhofer.de.

**JASMIN JAHIC** works in the Embedded Software Engineering department at Fraunhofer IESE in Kaiserslautern. He holds a Master in electrical engineering. His Ph.D. research focuses on concurrency and detection of parallelization errors. His email address is jasmin.jahic@iese.fraunhofer.de.