Cross-Layer Behavioral Modeling and Simulation of E/E-Architectures using PREEvision and Ptolemy II

Harald Bucher[1*], Simon Kamm[2], Jürgen Becker[1]

[1]Karlsruhe Institute of Technology – Institute for Information Processing Technologies, Engesserstr. 5, 76131 Karlsruhe; *bucher@kit.edu

[2]Vector Informatik GmbH, Philipp-Reis-Str. 1, 76137 Karlsruhe

SNE 29(2), 2019, 73 - 78, DOI: 10.11128/sne.29.tn.10472
Received: February 20, 2019 (Selected ASIM GMMS/STS 2019 Postconf. Publ.), Accepted: March 31, 2019
SNE - Simulation Notes Europe, ARGESIM Publisher Vienna, ISSN Print 2305-9974, Online 2306-0271, www.sne-journal.org

Abstract. In this paper, an approach for integrated behavior modeling and simulation within model-based electric/electronic-architecture (EEA) descriptions is presented. It leverages actor-oriented and UML state chart behavior modeling to address complex reactive systems. A key contribution is the aggregation of cross-layer behavior specified at the logical function architecture layer and at the hardware layer together with further properties of the EEA like the current consumption of electronic control units (ECUs) and the underlying network topology. The EEA and behavior modeling is done in the common industry tool PREEvision. Using these static descriptions, a unified simulation model is synthesized and executed using Ptolemy II, which extends a previously developed approach. In addition, a concept to feed back the simulation data into PREEvision is briefly described e.g., to further evaluate the gained results. Finally, a proof-of-concept is presented using an Adaptive Cruise Control application.

Introduction

Automotive electric/electronic-architectures (EEAs) are steadily growing in complexity due to the integration of evermore functions [1]. To cope with that complexity at system level, model-based architecture description languages (ADLs) and tools have been established in recent years such as the EAST-ADL [2], EEA-ADL [3] (realized in the tool PREEvision [4,1]) and Vehicle Systems Architect [5], each of which are compliant to the AUTOSAR [6] standard. Each of them offer sophisticated static modeling capabilities from several viewpoints such as requirements, functional network, hardware/software architecture, wiring harness and topology.

A common process is to start with the realization-independent and early stage modeling of the logical function architecture which typically stays stable for years and thus is the basis for further refinements in the development life cycle [1]. Another trend is the architecture-centric modeling of behavior integrated within the model-based EEA descriptions in order to have a common formal format for exchange and subsequent simulation analysis. The trend to amend this is underlined e.g., by the behavioral annexes of the EAST-ADL [2] and the AADL [7] or the integration of UML-compliant state charts into the latest PREEvision release v9.0 [4]. Therefore, recent research is focused on the generation of executable behavior from these static descriptions [8,9,10,11,12,13].

A downside of the behavioral annexes is that they only support the association of architectural components with simple, flat finite state machines (FSMs) which result in state and transition explosion with more complex systems. The mentioned approaches therefore often delegate detailed behavior to external descriptions which results in the loss of the integrated characteristics. In addition, it elicits inconsistencies between the architecture and behavior models and prevents the consideration of lower abstraction layers.

An approach which faces this challenge is presented in [8] by synthesizing and executing a cross-domain simulation from static EEA descriptions designed in PREEvision. In this work we extend that approach to support both actor-oriented and state chart behavior modeling to address complex reactive systems.

Concerning state charts the UML subset provided by PREEvision is leveraged and enhanced to support extended state machines to further handle complexity.

In addition, cross-layer behavior specifications and further EEA properties from lower abstraction layers are synthesized into a unified Ptolemy II (PtII) simulation model. A concept to feed back the simulation data into PREEvision is extended and completes the contributions.

1 Background

The overall baseline approach as proposed in [8] and the extensions addressed in this work are shown in Figure 1. Starting point is a data model e.g., as provided by PREEvision, which captures all relevant abstraction layers of an EEA. For modeling *executable* behavior integrated within the EEA model, a new layer called *Behavioral Logical Architecture (BLA)* is introduced that refines the static logical blocks with detailed behavior by reusing actors [14] from the *PtII Actor Library*. The library contains actors of the heterogeneous modeling and simulation tool Ptolemy II [15] and is imported as a separate library of logical block types into PREEvision. These block types are used to instantiate actors at the BLA layer. In combination with mappings from the LA layer to lower layers they provide the connection of the behavioral blocks of the BLA to domain-specific information at lower layers enabling the cross-domain simulation of the underlying network communication or even electric circuits [16] in an aspect-oriented manner. A variant-sensitive synthesis is also implemented supporting the analysis of architecture variants [17].

2 Concepts

To amend the baseline actor-oriented modeling with state-based behavior we leverage the newly added capability of PREEvision v9.0 to refine architecture artifacts with state charts across several layers including the logical architecture and components of the hardware layer such as ECUs and internal processing units.

The basic principle is to annotate a state chart as child artifact to an architecture artifact. Dependent on the abstraction layer, the interfaces to the state chart comprise different data providers and consumers. At the logical architecture, for instance, communication between functions is done via typed ports, which have attached an interface. The interface specifies the actual data exchanged e.g., in terms of data elements. This follows the AUTOSAR standard.

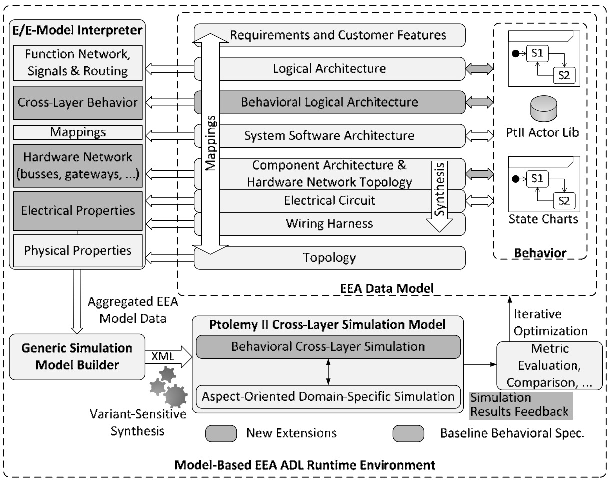
E/E-Model Interpreter
Function Network, Signals & Routing
Cross-Layer Behavior
Mappings
Hardware Network (busses, gateways, ...)
Electrical Properties
Physical Properties
Requirements and Customer Features
Logical Architecture
Behavioral Logical Architecture
System Software Architecture
Component Architecture & Hardware Network Topology
Electrical Circuit
Wiring Harness
Topology
Mappings
Synthesis
EEA Data Model
S1
S2
PtII Actor Lib
State Charts
Behavior
Aggregated EEA Model Data
Generic Simulation Model Builder
XML
Variant-Sensitive Synthesis
Ptolemy II Cross-Layer Simulation Model
Behavioral Cross-Layer Simulation
Aspect-Oriented Domain-Specific Simulation
Iterative Optimization
Metric Evaluation, Comparison, ...
Simulation Results Feedback
New Extensions
Baseline Behavioral Spec.
Model-Based EEA ADL Runtime Environment

Figure 1: Approach for cross-domain simulation synthesis of model-based EEAs [8] and new extensions to combine cross-layer behavior specifications using UML state charts and actor-oriented library components.

The specified data elements of each port are then available in the state chart of the function to use them in guard and action expressions. The modeling is illustrated in Figure 2. A similar modeling approach applies for hardware components except that state charts are annotated at instance level and data providers differ from data elements.

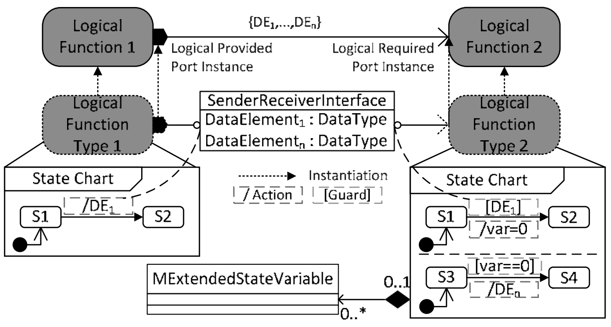

Figure 2: Modeling principle to refine logical function types with state charts. Communication is done via data elements.

2.1 Extended State Machines

A downside of the current state chart modeling capability is the missing support of extended state machines, which can significantly reduce the complexity [15]. Therefore, we propose a meta-model extension by extended state variables. This is indicated in Figure 2 by the composition of the state chart with the proposed meta-class `MExtendedStateVariable`.

2.2 Combining Actor-oriented and State Chart Behavior Simulation

In the baseline approach in Figure 1, behavior is specified by mapping an atomic logical function instance to a composite building block at the BLA layer. Executable actors are instantiated within that building block. Port prototype mappings are generated once to ensure the consistency between the interfaces of the atomic logical function and its refinement building block.

State charts are simulated using modal models [15,18] in PtII. Modal models basically represent a specialized composite actor containing a hierarchical state machine governed by an *FSMDirector*. Each state can contain another state machine refinement or even an actor-oriented sub-model following a distinct execution semantics i.e., a different model of computation (called *Director* in PtII). Modal models are also suitable to deterministically simulate *hybrid systems* [15]. Data exchange between modal models is done via ports. Therefore, a building block of type *ModalModel* is used to identify logical functions which contain a state chart description.

Additional data element sub-mappings are generated once in order to respect the interfaces of the logical functions and to connect the simulation model counterparts during simulation model synthesis. Each port of a building block represents a data element. See Figure 3.

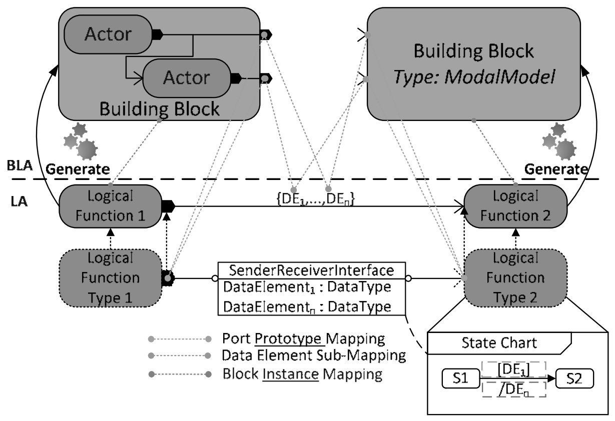

Figure 3: Generation of BLA building block stubs and mappings. State charts are encapsulated in a building block of type *ModalModel*.

2.3 Cross-Layer Behavioral Synthesis

To allow the simulation of cross-layer behavior we leverage the state chart refinements of hardware components. However, the link to higher layers i.e., the logical layer, is missing.

Therefore, we propose to use AUTOSAR-oriented `BasisServiceInterfaces` on logical ports in order to provide additional data elements or operations to communicate with state charts of mapped hardware components. In addition, we propose to reference ECU attributes as state chart variables. For instance, this enables the modeling and simulation of mode-based cross-layer behavior, where functions can request a certain operating mode of the ECU and only perform their functional behavior if the ECU responds it is ready to run. In order to allow spontaneous FSMs [15], which not only react on input events, a `timeout` guard expression (taken from PtII) is introduced e.g., to model the startup time of the ECU. A cross-layer model is exemplarily shown in Figure 4.

Finally, we propose the mapping of current consumption descriptions in terms of PREEvision's meta-class `MCurrentDescriptorType` on state transitions of hardware states. Together with the timed behavior, a mode-based current consumption can be simulated.

In the synthesized PtII model, the function and hardware state charts are encapsulated in distinct modal models communicating via ports which represent the basis service interfaces. An additional output port is generated for the current consumption of the ECU.

Hardware Network. In [8], network communication between functions such as CAN is traced based on their mapping to the hardware and is considered in the resulting simulation in an aspect-oriented way. Together with the state chart refinement of ECUs and processing units, it is possible to automatically include additional behavior along the communication path, such as gateways, by cascaded aspect-oriented simulations. Typically gateways have no logical function counterpart, since they are dependent on the mapping.

2.4 Simulation Data Feedback

To make use of the simulated results in PREEvision in order to perform further analysis and to relate the results with the original EEA model artifacts, a feedback approach is applied. The approach relies on OSGi [19] and is further described in [17]. We reuse and extend the approach by implementing a listener for modal model controllers focusing on the feedback of information about the simulated state machines such as timestamps, current state, previous state, output and variable actions.

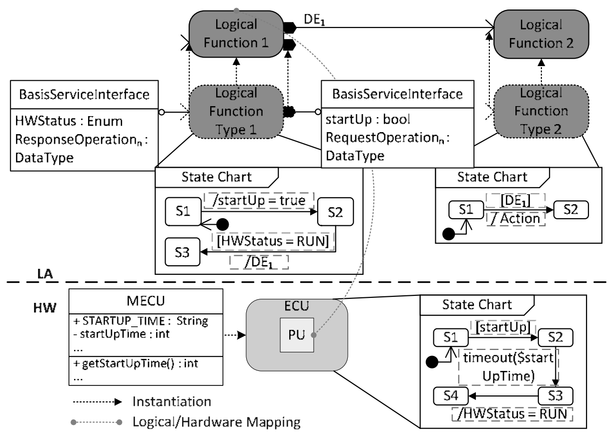

Figure 5: Cross-layer behavior specification using basis service interfaces on logical ports to communicate with the mapped hardware component state chart via additional data elements or operations.

2.5 Transformation Rules

In Table 1 the basic transformation rules between PREEvision's UML state chart subset and modal model artifacts in PtII are summarized.

Note that each generated PtII artifact is suffixed by the UUID of the original EEA model artifact in order to uniquely relate the artifacts and avoid name conflicts on PtII side.

3 Use Case Results

In this chapter, the concepts are demonstrated by means of an *Adaptive Cruise Control (ACC)* application presented in [8] which is enhanced based on Figure 4. The logical function architecture is shown in Figure 5.

UML State Chart Subset	Ptolemy II Modal Models
simple state, choice & junction pseudo-state	state
initial pseudo-state	state with property *isInitialState*
final state	state with property *isFinalState*
composite state	*state machine refinement* state
orthogonal state	*default refinement* state containing a discrete-event director and a modal model composite for each parallel region. Data dependencies between regions are analyzed and communicated via ports between the affected modal models. [18]
deep history state	history transition
state transition	ordinary transition
guard condition	guard expression
IO/variable action	output/set action expression

Table 1: Basic transformation rules between PREEvision's UML state chart subset and PtII modal models.

The *ACC_Testbench* generates the stimuli for the vehicle speed and radar speed sensor functions as well as for the ACC controller in a closed-loop fashion based on the calculated acceleration of the ACC controller. The stimuli values are generated with a sample period of $100ms$. The initial speeds and the distance are set to $15m/s$ and $190m$ respectively. Each of the functions offer `BasisServiceInterfaces` to request or retrieve a certain operating mode of the state chart of their mapped hardware component. The corresponding BLA building block stubs and mappings are generated according to Figure 3.

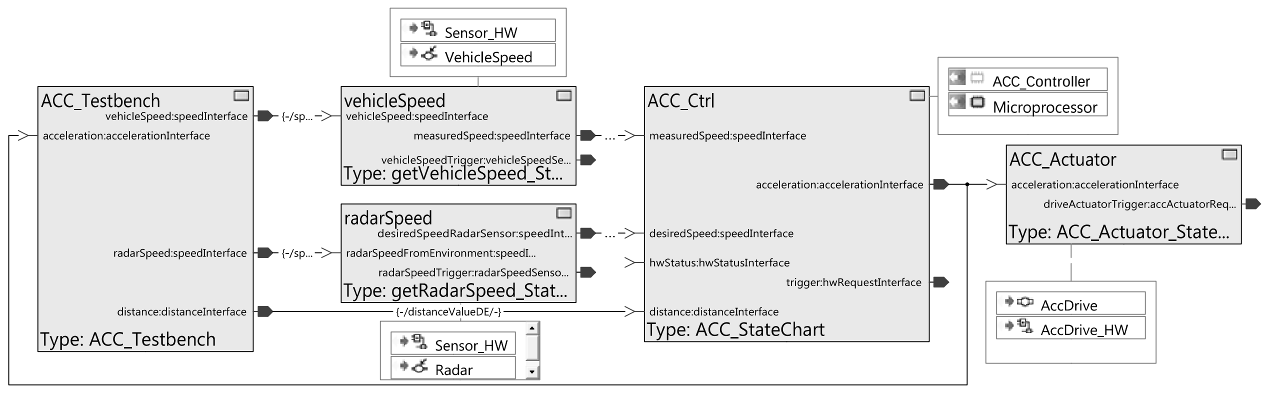
Sensor_HW
VehicleSpeed
ACC_Testbench
vehicleSpeed:speedInterface
acceleration:accelerationInterface
radarSpeed:speedInterface
distance:distanceInterface
Type: ACC_Testbench
vehicleSpeed
measuredSpeed:speedInterface
vehicleSpeedTrigger:vehicleSpeedSe...
Type: getVehicleSpeed_St...
radarSpeed
desiredSpeedRadarSensor:speedInt...
radarSpeedFromEnvironment:speedI...
radarSpeedTrigger:radarSpeedSenso...
Type: getRadarSpeed_Stat...
{-/distanceValueDE/-}
Sensor_HW
Radar
ACC_Ctrl
measuredSpeed:speedInterface
acceleration:accelerationInterface
desiredSpeed:speedInterface
hwStatus:hwStatusInterface
trigger:hwRequestInterface
distance:distanceInterface
Type: ACC_StateChart
ACC_Controller
Microprocessor
ACC_Actuator
acceleration:accelerationInterface
driveActuatorTrigger:accActuatorReq...
Type: ACC_Actuator_State...
AccDrive
AccDrive_HW

Figure 4: Logical function architecture of the ACC application. Each of the logical functions except for *ACC_Testbench* is refined by a state chart. Their mapping to the hardware layer is illustrated by the annotated text boxes. The behavior of the *ACC_Testbench* is modelled actor-oriented at the BLA layer and is not mapped to the hardware. Thus, a combined actor-oriented and state chart modeling is applied.

3.1 State Charts

The most important state charts are the one of the ACC controller shown in Figure 6 and its corresponding ECU state chart realizing an ECU Manager depicted in Figure 7. The remaining state charts of the sensor and actuator functions and their hardware are modeled simple. They only forward/retrieve the speed/acceleration values and request the sensors/actuator to run as long as they receive values. The ACC is calculating the acceleration only if the ECU is ready to run. The orthogonal state *operate* limits the calculated acceleration by the state variables *aMin* and *aMax*. In the *freeRoad* state the radar detects no vehicle, the own speed reached the desired speed and the sleep mode is requested. A wakeup is triggered when the radar detects a new vehicle. A shutdown is requested when the vehicle stands still.

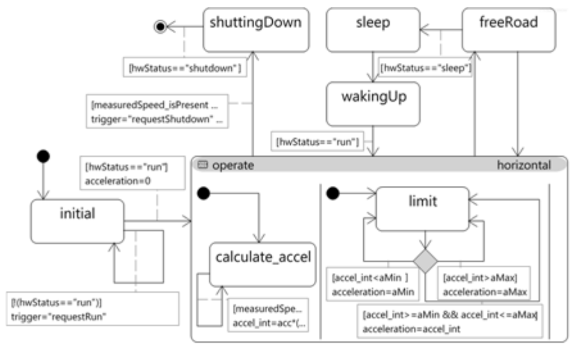

Figure 6: ACC controller state chart. Some transition actions are omitted for space reasons.

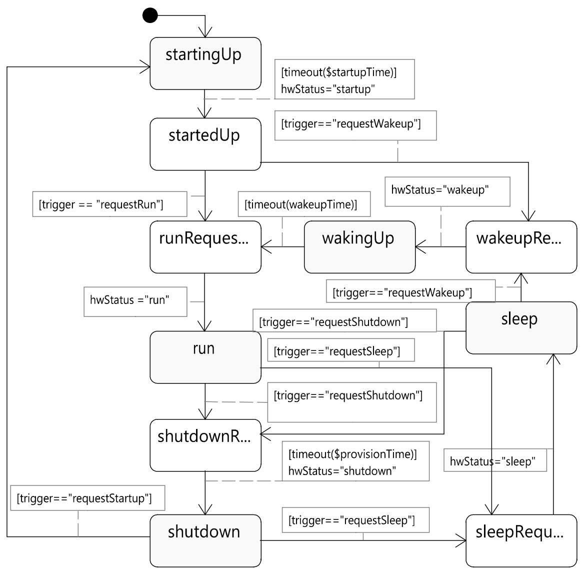
startingUp
[timeout($startupTime)]
hwStatus="startup"
startedUp
[trigger=="requestWakeup"]
hwStatus="wakeup"
[trigger == "requestRun"]
[timeout(wakeupTime)]
runReques...
wakingUp
wakeupRe...
[trigger=="requestWakeup"]
hwStatus ="run"
sleep
[trigger=="requestShutdown"]
run
[trigger=="requestSleep"]
[trigger=="requestShutdown"]
shutdownR...
[timeout($provisionTime)]
hwStatus="shutdown"
hwStatus="sleep"
[trigger=="requestStartup"]
[trigger=="requestSleep"]
shutdown
sleepRequ...

Figure 7: ACC ECU state chart realizing an ECU Manager oriented on the AUTOSAR fixed ECU Manager. Transitions to the yellow states have mapped a current descriptor type in order to simulate a mode-based current consumption.

The ACC ECU state chart represents the different operating modes which can be requested by the ACC controller state chart and sends back the current status via the `BasisServiceInterfaces`. In addition, the startup- and provision time attributes of the ECU ($50ms$ and $200s$) are referenced as well as a *wakeupTime* state variable ($10ms$) which are used as timeouts. Provision time is the time a component stays active after its shutdown is requested.

3.2 Simulation Results

Figure 8 shows the PtII plot of the ACC simulation. Until $250s$ the vehicle is following the leading vehicle. Then the leading vehicle disappears and the ACC accelerates to its desired speed at free road. At $300s$ a new vehicle is detected at a distance of $200m$. At $350s$ the vehicle is decelerating until it stands still at $380s$. At $300s$ the acceleration is limited to *aMin.*

Figure 9 shows the mode-based current consumption of the hardware components at the key time-points with synthetic values.

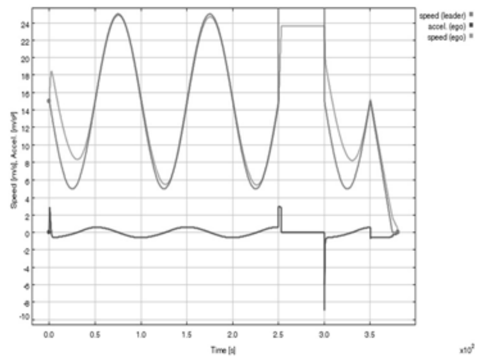

Figure 8: ACC simulation showing the speed of the leading vehicle (red) and the ego vehicle (green) in m/s as well as the acceleration calculated by the ACC (blue) in m/s^2.

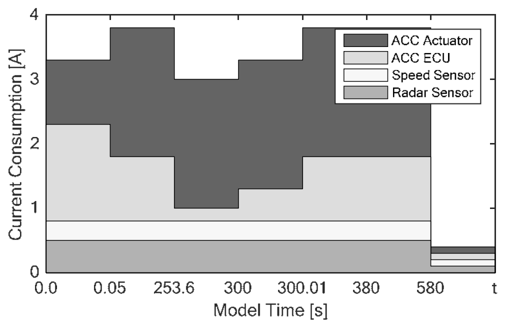

Figure 9: Current consumption of the mapped ACC hardware dependent on the operating mode. Created based on the fed back simulation data which is written to a CSV file in PREEvision.

The sensors are operating until their shutdown. Until $50ms$ the ACC ECU and Actuator are starting up before they are ready to run. At $253.6s$ the vehicle has reached its desired speed at free road and the ACC ECU goes to sleep mode. At $300s$ it wakes up for $10ms$. At $380s$ the vehicle stands still but all hardware components stay active for the same provision time before they shutdown at $580s$.

4 Conclusion

In this paper, we presented a set of concepts and their evaluation by an ACC application to model and simulate behavior of model-based EEAs in an integrated manner. The key contribution is the combination of actor-oriented and state chart based behavior across several abstraction layers. This enables new possibilities to analyze model-based EEAs in early development stages dependent on architectural decisions and information.

Future work could include enhanced support of UML state charts, the integration and consideration of behavior at the AUTOSAR-compliant system software architecture layer and envisioning their code generation.

References

[1] Schäuffele J, "E/E Architectural Design and Optimization using PREEvision," in *SAE Technical Paper 2016-01-0016*, 2016. [Online]. https://doi.org/10.4271/2016-01-0016

[2] EAST-ADL Association. (2013) EAST-ADL Domain Model Specification. [Online]. http://www.east-adl.info/Specification

[3] Matheis J, „Abstraktionsebenenübergreifende Darstellung von Elektrik/Elektronik-Architekturen in Kraftfahrzeugen zur Ableitung von Sicherheitszielen nach ISO 26262," Karlsruhe Institute of Technology, Ph.D. thesis ISBN: 978-3-8322-8968-3, 2010.

[4] Vector Informatik GmbH, "PREEvision v9.0 Manual," 2018.

[5] Mentor Graphics. (2018, Oct.) Volcano™ Vehicle Systems Architect. [Online]. https://www.mentor.com/products/vnd/autosar-products/volcano-system-architect

[6] AUTOSAR Consortium. (2018) AUTOSAR 4.4 (Automotive Open System Architecture) Specifications. [Online]. http://www.autosar.org

[7] SAE International, "SAE Architecture Analysis and Design Language (AADL) Annex Volume 2: Annex B: Data Modeling Annex, Annex D: Behavior Model Annex, Annex F: ARINC653 Annex," USA, Standard AS5506/2, Jan. 2011.

[8] Bucher H, Reichmann C, Becker J. "An Integrated Approach Enabling Cross-Domain Simulation of Model-Based E/E-Architectures," in *SAE Technical Paper 2017-01-0006*, Mar. 2017. [Online]. http://papers.sae.org/2017-01-0006/

[9] Weissnegger R et al. "Simulation-based Verification of Automotive Safety-critical Systems Based on EAST-ADL," *Procedia Computer Science*, vol. 83, pp. 245-252, 2016.

[10] Marinescu R et al. "Analyzing Industrial Architectural Models by Simulation and Model-Checking," in *Formal Techniques for Safety-Critical Systems*.: Springer International Publishing, 2015, vol. 476, pp. 189-205.

[11] MAENAD Consortium, "MAENAD Analysis Workbench," Deliverable D5.2.1 V4.0 2014. [Online]. http://www.maenad.eu/public/Deliverables

[12] Lasnier G, Pautet L, Hugues J, Wrage L. "An Implementation of the Behavior Annex in the AADL-Toolset Osate2," in *2011 16th IEEE International Conference on Engineering of Complex Computer Systems*, Apr. 2011, pp. 332-337.

[13] Larsen PG et al. "Integrated Tool Chain for Model-Based Design of Cyber-Physical Systems," in *The 14th Overture Workshop: Towards Analytical Tool Chains*, vol. 4/28, Nov. 2016, pp. 63-79.

[14] Lee EA, Neuendorffer S, Wirthlin MJ. "Actor-Oriented Design Of Embedded Hardware And Software Systems," *Journal of Circuits, Systems, and Computers*, vol. 12, pp. 231-260, 2003.

[15] Claudius Ptolemaeus, *System Design, Modeling, and Simulation using Ptolemy II*.: Ptolemy.org, 2014. [Online]. http://ptolemy.org/books/Systems

[16] Bucher H, Becker J. "Electric Circuit- and Wiring Harness-Aware Behavioral Simulation of Model-Based E/E-Architectures at System Level," in *2018 IEEE International Systems Engineering Symposium (ISSE)*, 2018, pp. 1-8.

[17] Bucher H, Neubauer K, Becker J. "Automated Assessment of E/E-Architecture Variants using an Integrated Model- and Simulation-based Approach," in *SAE Technical Paper 2019-01-0111*, 2019.

[18] Lee EA, Tripakis S. "Modal Models in Ptolemy," in *Proceedings of the 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, Oct. 2010, pp. 11-21.

[19] Wütherich G, Hartmann N, Kolb BJ, Lübken M. *Die OSGi-Service-Platform: eine Einführung mit Eclipse Equinox*, 1st ed.: dpunkt Verlag, 2008.