CROSS-LAYER BEHAVIORAL MODELING AND SIMULATION OF E/E-ARCHITECTURES USING PREEVISION AND PTOLEMY II

Harald Bucher
Jürgen Becker

Institute for Information Processing Technologies (ITIV)
Karlsruhe Institute of Technology (KIT)
Engesserstr. 5
76131 Karlsruhe, Germany
{bucher, becker}@kit.edu

Simon Kamm

Vector Informatik GmbH
Philipp-Reis-Str. 1
76137 Karlsruhe, Germany
simon.kamm@vector.com

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, such as the current consumption of electronic control units (ECUs), the underlying network topology and execution aspects of functions. 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 general 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.

Keywords: Automotive E/E-Architectures, MBSE, State Charts, Ptolemy II, PREEvision.

1 INTRODUCTION

Automotive electric/electronic-architectures (EEAs) are steadily growing in complexity due to the integration of evermore functions (Schäuffele 2016). 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 (Blom et al. 2013, EAST-ADL Association 2013), EEA-ADL (Matheis 2010) (realized in the tool PREEvision (Vector Informatik GmbH 2018)) and Vehicle Systems Architect (Mentor Graphics 2018), each of which are compliant to the AUTOSAR standard (AUTOSAR Consortium 2018). Each of them offer sophisticated static modeling capabilities at different abstraction layers with dedicated viewpoints such as requirements, functional network, hardware/software architecture, wiring harness and topology. Further ADLs exist to describe cyber-physical system (CPS) architectures in general like introduced in (Rajhans et al. 2009).

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 (Schäuffele 2016). 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 (EAST-ADL Association 2013) and the AADL (SAE International 2011) or the integration of UML-compliant state charts into the latest PREEvision release v9.0 (Vector Informatik GmbH 2018). Therefore, recent research is focused on the generation of executable simulation models from these static descriptions (Bucher et al. 2017, Larsen et al. 2016, Lasnier et al. 2011, Marinescu et al. 2015, MAENAD Consortium 2014, Weissnegger et al. 2016).

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. The CPS ADL proposed in (Rajhans et al. 2009) supports the annotation of both Finite State Processes or Linear Hybrid Automata and the generation of a text file for a dedicated analysis tool. However, the ADL lacks to handle complexity with several abstraction layers as done with typical EEA ADL and a combination of distinct formalisms unified in the resulting simulation model is not discussed. A generic framework considering composition of heterogeneous models using semantic mappings based on the same CPS ADL is presented in (Rajhans et al. 2014). However, their focus lies on the formal analysis and verification of heterogeneous semantic hierarchies of verification models with distinct modeling formalisms related to a specific architecture view rather than on simulation. The possibility to refine a logical architectural view with behavior of a specific simulation model integrated in the EEA model and combine it with both state chart annotations and architecture properties from lower EEA layers such as current consumption of ECUs, the underlying network topology or execution aspects of functions in a single heterogeneous simulation model is not yet addressed.

In contrast, an approach which faces this challenge is presented in (Bucher et al. 2017) where heterogeneity of models is managed inherently by leveraging the capabilities of Ptolemy II (PtII) (cf. Section 2.1). It synthesizes and executes an integrated aspect-oriented heterogeneous simulation from static EEA descriptions designed in PREEvision starting from the realization-independent logical architecture. 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 properties from lower EEA abstraction layers are synthesized into a unified PtII simulation model. A concept to feed back the simulation data into PREEvision is extended and completes the contributions.

2 BACKGROUND

2.1 Baseline Approach

The overall baseline approach as proposed in (Bucher et al. 2017) 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 from requirements down to the vehicle's topology (cf. the EEA Data Model abstraction layers in Figure 1). 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 (Lee et al. 2003) from the PtII Actor Library. The library contains actors of the heterogeneous modeling and simulation tool Ptolemy II (Ptolemaeus 2014) and is imported as a separate library of logical block types into PREEvision. Actors are software components which execute concurrently and communicate with each other via ports. They can be atomic or composite to allow hierarchical composition and their execution and communication semantics are governed by a so-called Director

implementing a certain model of computation. A key element is the deterministic, hierarchical composition of distinct directors. The block types of the imported library are used to instantiate actors at the BLA layer. In combination with mappings from the Logical Architecture (LA) layer to lower layers they provide the connection of the behavioral blocks of the BLA to architecture properties at lower layers enabling the cross-domain simulation of the underlying network communication or even electric circuits (Bucher and Becker 2018) in an aspect-oriented manner. A variant-sensitive synthesis is also implemented supporting the analysis of architecture variants (Bucher et al. 2019).

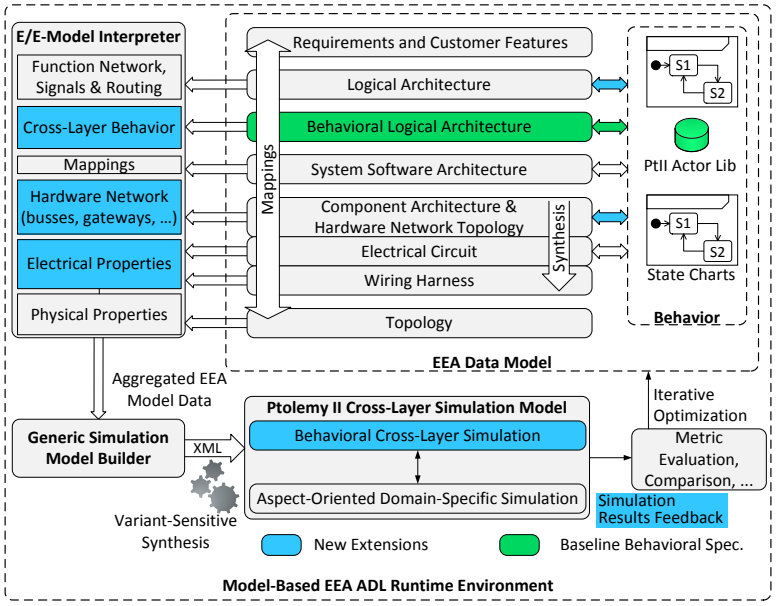

Figure 1: Approach for cross-domain simulation synthesis of model-based EEAs (Bucher et al. 2017) and new extensions to combine cross-layer behavior specifications using UML state charts and actor-oriented library components.

2.2 Architectural State Chart Refinement

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 layer, 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. 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. A similar modeling approach applies for hardware components except that state charts are annotated at instance level and data providers differ from data elements. The modeling is illustrated in Figure 2.

3 CONCEPTS

3.1 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 (Ptolemaeus 2014). 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`. A typical example is a counter variable incremented in an action expression and using a certain value of it in guard expressions.

3.2 Combining Actor-oriented and State Chart Behavior Modeling and 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 (Ptolemaeus 2014, Lee and Tripakis 2010) in PtII. Modal models basically represent a specialized composite actor containing a hierarchical state machine governed by a *FSMDirector*. Each state can contain another state machine refinement or even an actor-oriented sub-model following a distinct director. Modal models are also suitable to deterministically simulate hybrid systems (Ptolemaeus 2014). 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.

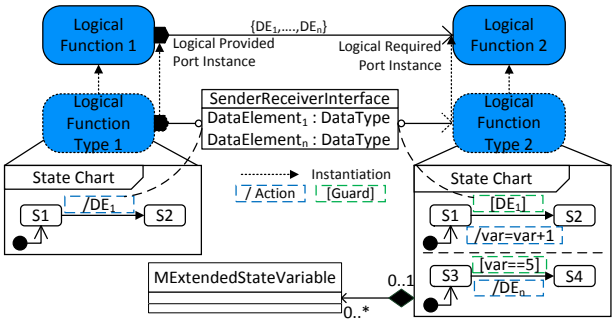

Figure 2: Modeling principle to refine logical function types with state charts. Communication is done via data elements.

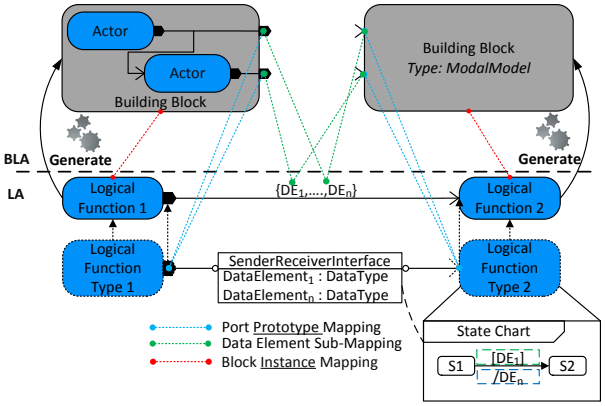

Figure 3: Generation of BLA building block stubs and mappings. State charts are encapsulated in a building block of type *ModalModel*

3.3 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 (Ptolemaeus 2014), which not only react on input events, a timeout guard expression (taken from PtII) is introduced e.g., to model the start-up time of the ECU. An example cross-layer model is shown in Figure 4.

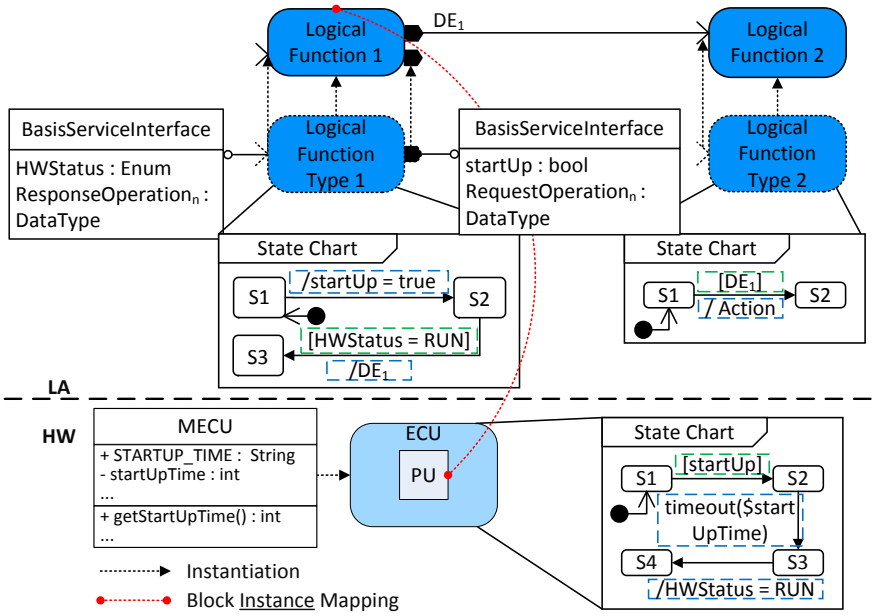

Figure 4: Cross-layer behavior specification using basis service interfaces on logical ports to communicate with the mapped hardware component state chart via additional data elements or operations.

Furthermore, 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 (cf. Figure 5). An additional output port is generated for the current consumption of the ECU (not shown in Figure 5).

In (Bucher et al. 2017), network communication between functions such as CAN is traced based on their mapping to the hardware and 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 and the resulting communication path. Similarly, in (Bucher et al. 2017), specified worst-case latency properties on logical functions are leveraged to simulate latencies using composite execution aspect actors (Akkaya et al. 2016), see the Latency Execution Aspect of Function 1 Composite in Figure 5. Aspects are junction points in the main model of interest where the simulation of the actual behavior is mediated to a sub-model refining the behavior with non-functional concerns such as execution time in a modular way without changing the structure of the main model. Specifically, execution aspects are decorated on actors and, if one is enabled, is conducted by a director always before the decorated actor is scheduled to be fired.

In the following, an outline is given how the usage of execution aspects can be advanced to respect the deployment of logical software functions to their microarchitecture in the EEA model to enable further design space exploration by considering shared execution resources. To achieve this, we added custom

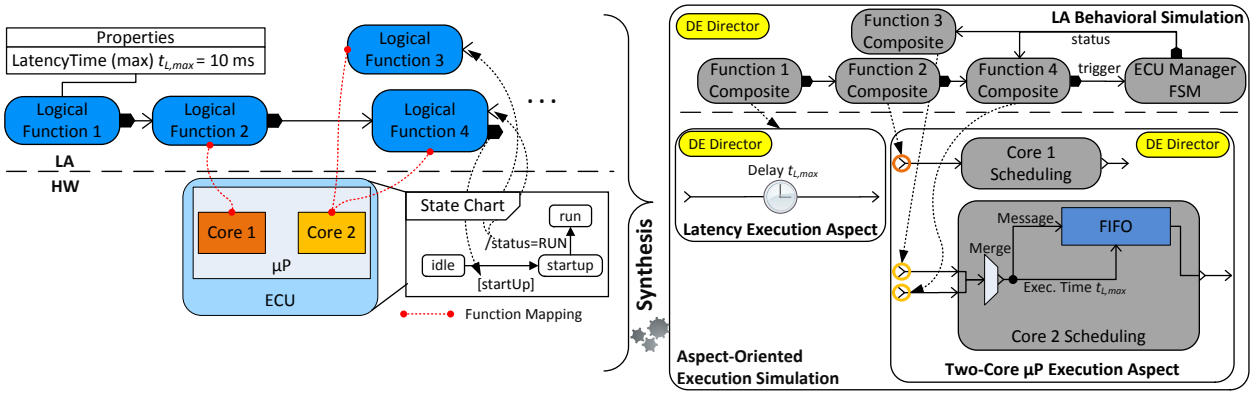

Figure 5: Function execution aspect synthesis dependent on the mapping to the conducting core.

sub-mappings in PREEvision to deploy functions not only to a microprocessor but also to internal cores. In the example of Figure 5, a composite execution aspect governed by a discrete-event (DE) director is synthesized for the enclosed microprocessor containing a composite actor for each core in the EEA model. Logical Function 2 is mapped to Core 1 and redirected to its execution request port connected to the Core 1 composite actor. Logical Function 3 and 4 are mapped to Core 2 and therefore redirected to individual execution request ports connected to the Core 2 composite. The synthesized PtII models of the cores in Figure 5 represent a default behavior following the implementation presented in (Akkaya et al. 2016): the execution requests of the actors are merged and scheduled on a FIFO server with a service time equal to the specified execution time parameter of the decorated actor. However, more elaborate execution models could be reflected. For instance, a priority-based scheduling policy with priorities derived from the AUTOSAR-compliant software layer where logical functions can be mapped to. Or enhanced composite execution aspects which are synthesized from explicitly modeled behavior by introducing the possibility to refine internal processor components like cores with further detailed state chart or actor-oriented models. Currently, the advanced execution aspect synthesis is ongoing work and will be investigated further in future work.

3.4 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 (Wütherich et al. 2008) and is further described in (Bucher et al. 2019). 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.

3.5 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 Universally Unique Identifier (UUID) of the original EEA model artifact in order to uniquely relate the artifacts and avoid name conflicts on PtII side.

An orthogonal state, also referred to as AND-state, is transformed into a *default refinement* state containing a modal model composite for each parallel region and a discrete-event (DE) or a synchronous-reactive (SR) director specifying the concurrent execution semantics. Data dependencies between regions are analyzed and communicated via ports between the affected modal models (Lee and Tripakis 2010). Data dependencies

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

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
deep history state	history transition
(external) state transition	ordinary transition
guard condition	guard expression
IO/variable action	output/set action expression

can be analyzed based on the referenced data elements as explicit meta-model artifacts, in contrast to "name matching" in the UML semantics (Lee and Tripakis 2010). The explicit references allow to uniquely identify the providing and consuming regions of the data elements and increases modularity of the state chart design. Please note that other UML state chart elements such as internal or local transitions are not yet supported and the semantics of the state chart execution is defined by the semantics of modal models.

4 USE CASE RESULTS

In this section, the concepts are demonstrated by means of an Adaptive Cruise Control (ACC) application presented in (Bucher et al. 2017) which is enhanced based on Figure 4. At first, experiments without core mappings are performed, execution aspects are considered in Section 4.3. The logical function architecture is shown in Figure 6. 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 at a sample rate of $1/100ms$. 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.

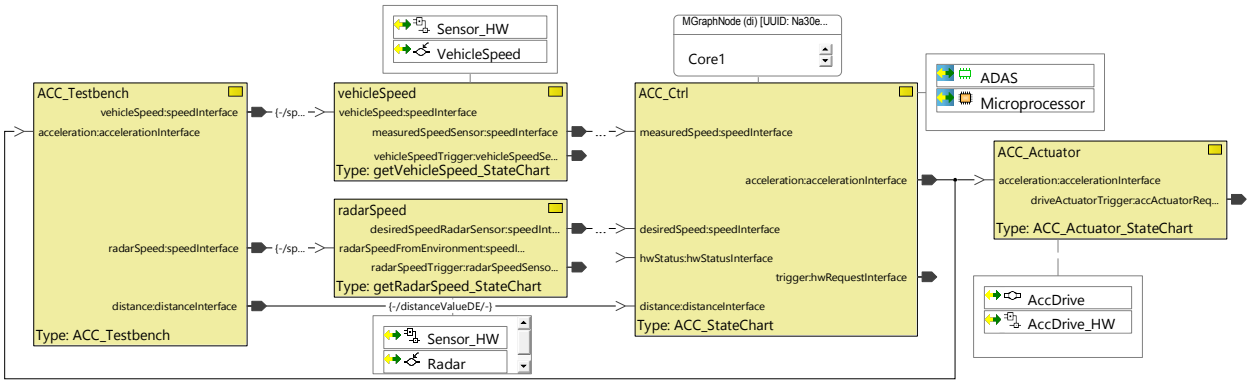

Figure 6: Logical function architecture of the ACC application. Each of the logical functions except for ACC_Testbench is refined by a state chart. Their mapping to the hardware layer is illustrated by the annotated text boxes. The behavior of the ACC_Testbench is modeled actor-oriented at the BLA layer and is not mapped to the hardware. Thus, a combined actor-oriented and state chart modeling is applied.

4.1 State Charts

The important state charts are the one of the ACC controller shown in Figure 7 and its corresponding ECU state chart realizing an ECU Manager depicted in Figure 8. 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.

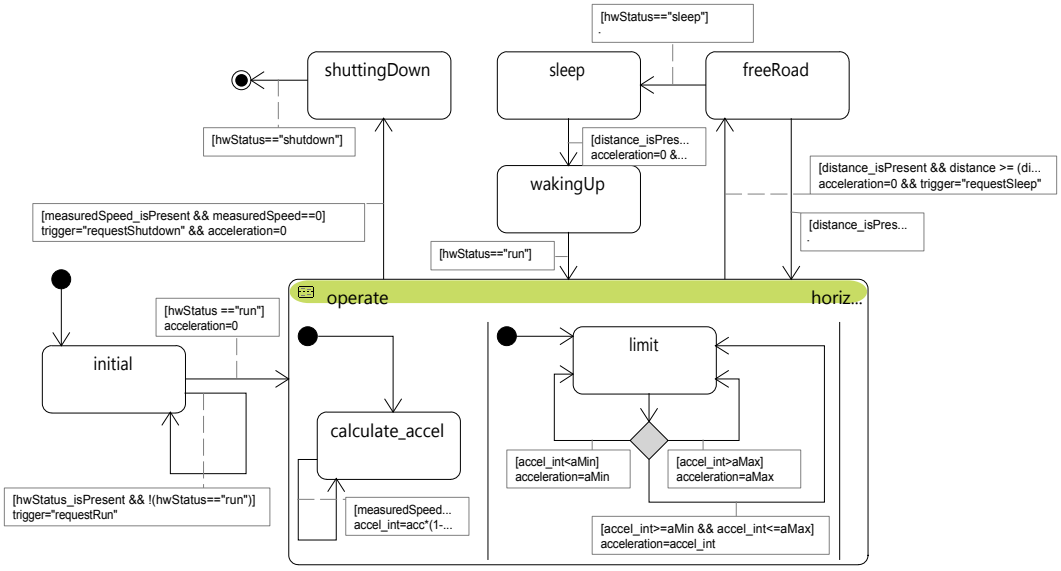

Figure 7: ACC controller state chart. Some transition actions are omitted for space reasons.

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 wake-up is triggered when the radar detects a new vehicle. A shutdown is requested when the vehicle stands still. 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 start-up- and provision time attributes of the ECU (50*ms* and 200*s*) are referenced as well as a *wakeupTime* state variable (10*ms*) which are used as timeouts. Provision time is the time a component stays active after its shutdown is requested.

4.2 Simulation Results

Figure 9 shows the PtII plot of the ACC simulation. Until 250*s* the vehicle is following the leading vehicle. Then the leading vehicle disappears and the ACC accelerates to its desired speed at free road. At 300*s* a new vehicle is detected at a distance of 200*m*. At 350*s* the vehicle is decelerating until it stands still at 380*s*. At 350*s* the acceleration is limited to *aMin*. Figure 10 and Figure 11 show the mode-based current consumption of the hardware components at the key time-points with synthetic values. The sensors are operating until their shutdown. Until 50*ms* the ACC ECU and Actuator are starting up before they are ready to run. At 253.6*s* the vehicle has reached its desired speed at free road and the ACC ECU goes to sleep mode. At 300*s* it wakes up for 10*ms*. At 380*s* the vehicle stands still but all hardware components stay active for the same provision time before they shutdown at 580*s*.

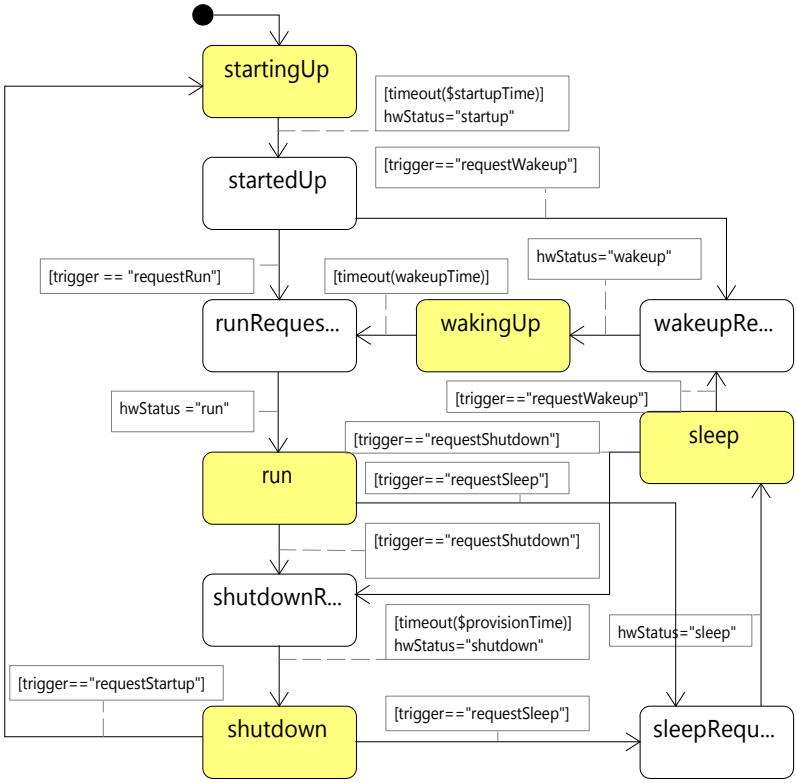

Figure 8: ACC ECU state chart realizing an ECU Manager oriented on the AUTOSAR fixed ECU Manager. Transitions to the yellow states have mapped a current descriptor type in order to simulate a mode-based current consumption.

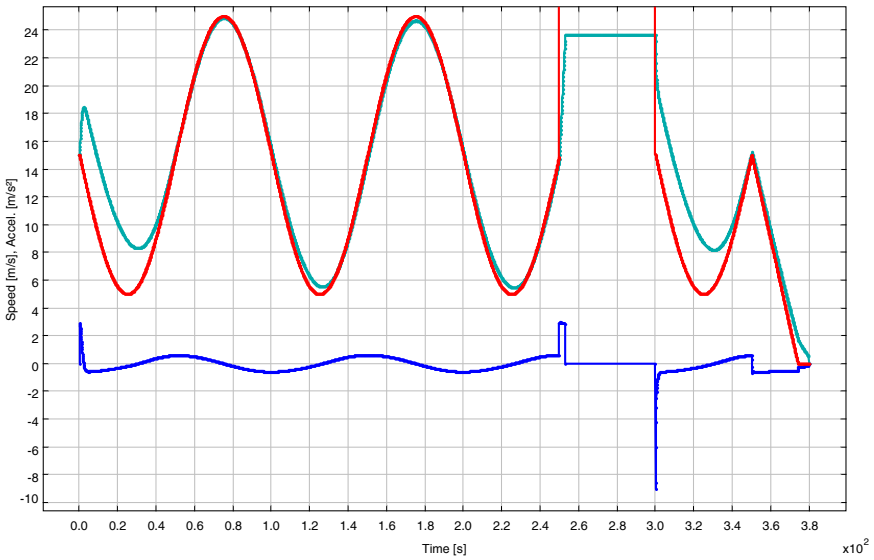

Figure 9: ACC simulation showing the speed of the leading vehicle (red) and the ego vehicle (green) in m/s as well as the acceleration calculated by the ACC (blue) in m/s^2.

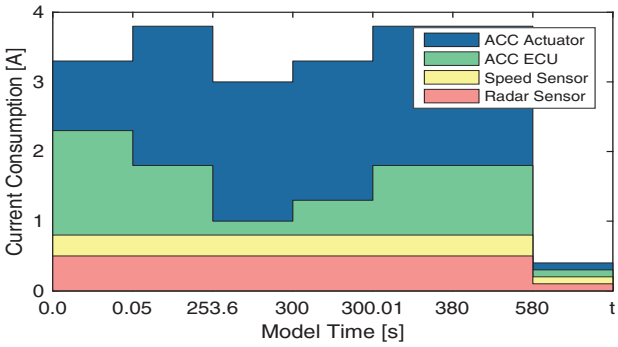

Figure 10: Current consumption of the mapped ACC hardware. Created based on the fed back simulation data which is written to a CSV file in PREEvision.

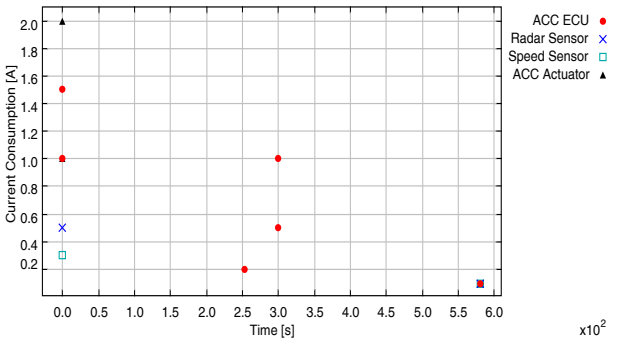

Figure 11: Simulation plot of the current consumption of the mapped ACC hardware dependent on the operating mode.

4.3 Execution Aspects

Preliminary experiments showed that in the latest PtII version execution aspect support needed to be enhanced for modal models compared to regular composites. The reason is the modal model execution semantics (Ptolemaeus 2014): in each firing, a state transition requests a re-firing in the executive director of the modal model with the same model timestamp independent on whether additional input port events will occur. This ensures that a destination state with an immediately enabled transition e.g., without a guard will take that transition without advancing model time. Consequently, however, this triggers the execution aspect of the modal model once again causing an additional artificial execution time which is not reasonable for such state transitions. Hence, these so-called pure events, stemming from such transitions and without any connected input tokens, are bypassed and scheduled without consulting the execution aspect. This support is implemented in the DE director as executive director of the modal models and demonstrated in the following.

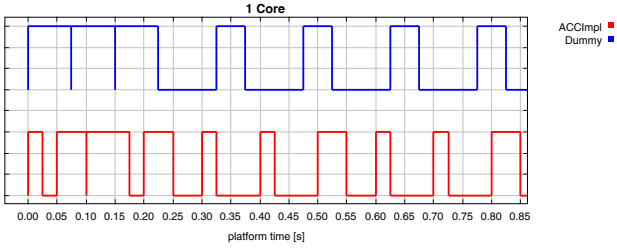

Figure 12: Execution times of the two functions running on a shared core.

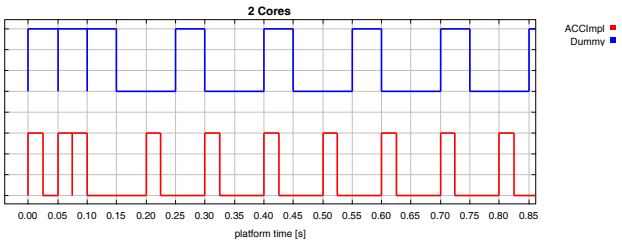

Figure 13: Execution times of the two functions running on individual cores.

Figure 12 and Figure 13 show the execution times of the ACC_Ctrl function and an additionally modeled dummy function mapped to the same core and to distinct cores respectively. The execution aspect illustrated in Figure 5 is synthesized and the execution times in the plots are tracked by monitors as presented in (Akkaya et al. 2016). The dummy state chart requests its execution every 100*ms* as soon as it received the ECU *run* state. The specified worst-case latency is 50*ms* and the one of ACC_Ctrl 25*ms*. The plots show that each received ECU Manager state and stimuli event for both modal models are scheduled once even when transitions occur e.g., also internally in composite states. It can be seen that for this specific parameter setup both schedules are feasible except for the start-up phase. Using two cores, for instance, the ACC controller consumes the stimuli data and *run* state in due time at 0.1*s*. However, the acceleration can not yet be calculated using the stimuli data since the ACC is still in its *initial* state and about to take the transition into the *operate* state. Subsequently, the stimuli data is processed in due time every 0.1*s* in both schedules.

5 CONCLUSION

In this paper, we presented a set of concepts and an evaluation of an ACC application to model and simulate behavior of model-based EEAs in an integrated manner. The key contributions are the combination of actor-oriented and state chart behavior as well as their aggregation with further architectural properties across the logical and hardware EEA 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 of properties and behavior at the AUTOSAR-compliant software architecture layer and envisioning its code generation. Currently, the advanced usage of execution aspect simulation is ongoing work and will be investigated further to reflect shared execution resources and different scheduling policies of functions dependent on their hardware/software mapping in order to improve early-stage evaluation of industry-scale EEA models.

REFERENCES

Akkaya, I., P. Derler, S. Emoto, and E. A. Lee. 2016, May. "Systems Engineering for Industrial Cyber-Physical Systems Using Aspects". *Proceedings of the IEEE* vol. 104 (5), pp. 997–1012.

AUTOSAR Consortium 2018. "Automotive Open System Architecture". [online; https://www.autosar.org/about/, accessed: 23.02.2019].

Blom, H., H. Lönn, F. Hagl et al. 2013. "EAST-ADL - An Architecture Description Language for Automotive Software-Intensive Systems". White paper version 2.1.12, EAST-ADL Association.

Bucher, H., and J. Becker. 2018, October. "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)*, pp. 1–8, IEEE.

Bucher, H., K. Neubauer, and J. Becker. 2019, April. "Automated Assessment of E/E-Architecture Variants Using an Integrated Model- and Simulation-Based Approach". *SAE Technical Paper 2019-01-0111*.

Bucher, H., C. Reichmann, and J. Becker. 2017, March. "An Integrated Approach Enabling Cross-Domain Simulation of Model-Based E/E-Architectures". *SAE Technical Paper 2017-01-0006*.

EAST-ADL Association 2013. "EAST-ADL Domain Model Specification V2.1.12". [online; http://www.east-adl.info/Specification, accessed: 23.02.2019].

Larsen, P., C. Thule, K. Lausdahl et al. 2016, November. "Integrated Tool Chain for Model-Based Design of Cyber-Physical Systems". In *The 14th Overture Workshop: Towards Analytical Tool Chains*, edited by P. Larsen et al., Volume 4/28, pp. 63–79, Aarhus University, Department of Engineering.

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

Lee, E. A., S. Neuendorffer, and M. J. Wirthlin. 2003. "Actor-Oriented Design Of Embedded Hardware And Software Systems". *Journal of Circuits, Systems, and Computers* vol. 12 (3), pp. 231–260.

Lee, E. A., and S. Tripakis. 2010, October. "Modal Models in Ptolemy". In *Proceedings of the 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pp. 11–21, Linköping University Electronic Press.

MAENAD Consortium 2014. "MAENAD Analysis Workbench". Deliverable d5.2.1 v4.0.

Marinescu, R., H. Kaijser, M. Mikucionis et al. 2015. "Analyzing Industrial Architectural Models by Simulation and Model-Checking". In *Formal Techniques for Safety-Critical Systems: Third International Workshop, FTSCS 2014. Revised Selected Papers*, edited by C. Artho and C. P. Ölveczky, pp. 189–205. Cham, Springer International Publishing.

Matheis, J. 2010. *Abstraktionsebenenübergreifende Darstellung von Elektrik/Elektronik-Architekturen in Kraftfahrzeugen zur Ableitung von Sicherheitszielen nach ISO 26262*. Ph. D. thesis, Universität Karlsruhe (TH).

Mentor Graphics 2018. "Volcano™ Vehicle Systems Architect". [online; https://www.mentor.com/products/vnd/autosar-products/volcano-system-architect, accessed: 23.02.2019].

Ptolemaeus, C. (Ed.) 2014. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org.

Rajhans, A., A. Bhave, I. Ruchkin et al. 2014. "Supporting Heterogeneity in Cyber-Physical Systems Architectures". *IEEE Transactions on Automatic Control* vol. 59 (12), pp. 3178–3193.

Rajhans, A., S.-W. Cheng, B. Schmerl et al. 2009. "An Architectural Approach to the Design and Analysis of Cyber-Physical Systems". *Electronic Communications of the EASST* vol. 21, pp. 10.

SAE International 2011, January. "SAE Architecture Analysis and Design Language (AADL) Annex Volume 2: Annex B: Data Modeling Annex, Annex D: Behavior Model Annex, Annex F: ARINC653 Annex". Standard as5506/2.

Schäuffele, J. 2016. "E/E Architectural Design and Optimization using PREEvision". *SAE Technical Paper 2016-01-0016*.

Vector Informatik GmbH 2018. *PREEvision® Version 9.0 Manual*.

Weissnegger, R., M. Schuss, C. Kreiner et al. 2016. "Simulation-based Verification of Automotive Safety-critical Systems Based on EAST-ADL". *Procedia Computer Science* vol. 83, pp. 245 – 252.

Wütherich, G., N. Hartmann, B. Kolb, and M. Lübken. 2008. *Die OSGi Service Platform: eine Einführung mit Eclipse Equinox*. dpunkt Verlag.

AUTHOR BIOGRAPHIES

HARALD BUCHER graduated from Karlsruhe Institute of Technology (KIT), Germany in 2012 and received his M. Sc. degree in electrical engineering and information technology with distinction. From 2012 to 2015 he was stipendiary of the German Research Foundation (DFG) within the Research Training Group 1194. Since 08/2012 he is a research associate at the Institute for Information Processing Technologies (ITIV) at KIT. His research interests focus on model-based systems engineering and modeling and simulation, especially in the automotive domain with regard to E/E-architectures, Car-to-X communication and cyber-physical systems. His email address is bucher@kit.edu.

SIMON KAMM graduated from Karlsruhe Institute of Technology (KIT), Germany in 2018 and completed his studies with his master's thesis on cross-layer behavioral modeling within model-based E/E-architectures. Since 07/2018 he is working as a software development engineer at Vector Informatik GmbH on PREEvision, a tool supporting the model-based development of automotive E/E-architectures. His email address is simon.kamm@vector.com.

JÜRGEN BECKER received the Diploma and Ph.D. (Dr.-Ing.) degree from Technical University Kaiserslautern, Germany. He is a full professor for embedded electronic systems and Head of the Institute for Information Processing Technologies (ITIV) at Karlsruhe Institute of Technology (KIT). From 2005-2009 he has been appointed as Vice President for Education at Universität Karlsruhe (TH) and Chief Higher Education Officer (CHEO) at KIT from 2009-2012. From 2012 to 2014 he served as Secretary General of CLUSTER, an association of 12 leading technical universities in Europe. In 2013, Prof. Becker received the Honorary Doctor award (Dr. h.c.) from Technical University Budapest, Hungary and he currently is coordinator of European and national project initiatives. His research interests include hardware/software Systems-on-Chip (SoC), cyber-physical systems, heterogeneous multicore architectures, reconfigurable computing, fast data acquisition and Networks-on-Chip (NoC). His email address is becker@kit.edu.