SAE Technical Paper 2017-01-0006, © 2017 SAE International, doi: 10.4271/2017-01-0006.

An Integrated Approach Enabling Cross-Domain Simulation of Model-Based E/E-Architectures

Harald Bucher*, Clemens Reichmann[+], Jürgen Becker*
*Karlsruhe Institute of Technology (KIT), [+]Vector Informatik GmbH

Abstract

The increasing complexity of electric/electronic architectures (EEA) in the automotive domain raised the necessity of model-based development processes for the design of such heterogeneous systems, which combine different engineering principles with different viewpoints. High-level simulation is a great means to evaluate the EEA in the concept phase of the design, since it reduces costly real-world experiments. However, model-based EEA design and analysis as well as its simulation are often separate processes in the development lifecycle. In this paper, we present a novel approach that extends state-of-the-art model-based systems engineering principles of EEA by a behavior specification reusing library components. The specification is seamlessly integrated in the development process of a single source EEA model. Therewith, the starting point is the abstract logical function architecture of the EEA. Based on this single source EEA model we synthesize a unified high-level simulation model, which is capable of linking the behavioral model with lower level implementation details of other domains, e.g. the network communication of the underlying hardware topology. This cross-layer simulation enables an early but holistic system's behavior analysis of the dynamic changes which typically depend on the scenarios applied. Moreover, the integrated approach enables the potential to feedback the simulation results into suitable EEA metrics and benchmarks for further analysis and optimization as well as the seamless traceability of the behavioral specification to requirements and other abstraction layers. A driver assistance system use case demonstrates the proof-of-concept and the benefits of our methodology.

Introduction

The increasing complexity of automotive and avionic electric/electronic architectures (EEA) due to the integration of evermore features required the need of model-based development principles in the last years in order to cope with that complexity. Nowadays modern high-class vehicles are a distributed system of up to 100 ECUs communicating over different bus systems [1]. Latest communication technology like Car-2-X communication, safety standards like ISO26262, various innovations like driver assistance systems or autonomous driving as well as integrating multi-core technology into ECUs [2] makes the development of an EEA an even more difficult endeavor.

A successful *concept phase* determines about 80% of the costs of further lifecycle process steps in the development of EEA [3]. The concept phase is an iterative process that deals with the overall functionality, convenience, risks and costs of the vehicle that have to meet its requirements. Hence, early stage analysis and evaluation at system-level is necessary in order to avoid subsequent changes in the EEA which incur enormous costs, because of extensive re-designs.

For system modelling of EEA several approaches and methodologies have emerged in the last years, either based on general purpose modelling languages like SysML/MARTE [4] [5] or on domain-specific architecture description languages like EEA-ADL [6], EAST-ADL [7], or the AUTOSAR [8] meta-model. The named domain-specific ones all have a layered architecture where each layer represents a different abstraction level and view on the system. AUTOSAR is also based on abstraction layers and complements the implementation level e.g. of the EAST-ADL specification [7].

However, the major drawback of these approaches is the poor support for modelling the dynamic system behavior, which is a necessary prerequisite to perform system level simulation and evaluation. Specifically, they only specify end-to-end flows (*what* the system does) without defining the *semantics* for the model execution. Simulation is a great means to evaluate and test a system, since it provides an early feedback on the system dynamics and especially of non-functional properties which typically depend on the simulation scenario applied. Additionally, it reduces costly real-world experiments and time-to-market. Hence, there is a need to have a behavioral specification that not only captures the temporal end-to-end flows of the system functions (*what* the system does), but also one that is *executable* and shows *how* the system dynamics behave based on the implemented behavior.

Furthermore, there is a lack in state-of-the-art methodologies and tools covering both, modelling the system architecture and simulating the system's dynamic behavior linked with implementation details in an integrated development process. Simulation of functional behavior, controls or plants and domain-specific implementation details (e.g. the distributed execution of the function and the used bus protocol between the hardware) is typically done in separate domain-specific tools like MATLAB/Simulink [9] or Modelica [10]. They are following different abstraction levels, meta-models and *Models of Computations* (MoC) [11] which represent the individual domains. This makes it hard to

integrate them in a holistic manner or yields to fragile tool chains [12]. Hence, the heterogeneous nature of EEA would benefit from an integrated and holistic approach, where the focus lies on the interoperation of semantics of heterogeneous domains in a deterministic and unified manner [12] [13].

In this paper, we address this issue and present a generic approach for the cross-domain simulation of model-based EEA. We introduce an additional modelling layer, which specifies the functional behavior, control or plant models reusing an *actor* library. The specification is seamlessly integrated in the development process of a single source EEA model [6]. The novelties and contributions are 1. the executable behavioral system model specification at a high abstraction level integrated in the EEA model design flow, 2. the synthesis and integrated conduction of a unified simulation model, which links the behavioral models to more detailed domain-specific information from lower abstraction levels (e.g. bus communication), 3. the reuse of library-based simulation components, 4. the extension of existing EEA's logical function descriptions by the behavioral specification via layer mapping principles and 5. the application to an Adaptive Cruise Control (ACC) use case as proof-of-concept. The simulation model is synthesized for the heterogeneous modelling and simulation framework *Ptolemy II* [14] (PtII) and conducted within the EEA design and analysis tool PREEvision [15] from Vector which is based on [6].

The remainder of the paper is organized as follows: first, we present some related works and give background information about the used approaches and tools in this work. Afterwards, we detail our model-based simulation model synthesis methodology and its individual components. Then we present our ACC case study and discuss our approach before we finally conclude the work.

Related Work

The benefits of general purpose approaches based on SysML and/or MARTE are the possibility to reuse existing tools and the capability to model behavioral end-to-end flows including timings. However, SysML only specifies the syntax of the behavioral models and not their *semantics* in order to execute them. Thus they are not suitable for high-level simulation. Similarly, the domain-specific approaches like the EAST-ADL and also the EEA-ADL used in this work do not support the explicit modelling of executable behavior, but only *what* the system does. Specifically, e.g. the EAST-ADL is referencing their *FunctionBehavior* blocks to an external behavior description in a tool-dependent format [7] without integrating it into the development process like it is done in our approach.

There are several previous works dealing with the generation of simulation models out of model-based EEA. Autonomie [16] is a tool which supports GUI based vehicle level modelling. It is capable of automatically assembling the vehicle model consisting of several Simulink sub-systems by means of a set of XML meta-data files. Although it provides similar features like an automatic model assembly, it is done on a different granularity. Moreover, non-functional properties in simulation and traceability features across all EEA abstraction layers due to the lack of integrated cross-layer mappings are not mentioned.

Works which address the generation of simulation models from EAST-ADL models are [17] [18] [19]. In [17] the authors use the SystemC/TLM [20] system level programming language to specify the behavior and map them to EAST-ADL layers. An automotive use case is used in their work. However, the approach only works well with discrete digital systems and lacks for the support of other domains, e.g. the continuous domain for modelling analogue components or plants. The authors in [18] proposed a mechanism to simulate EAST-ADL *FunctionBehavior* blocks by linking them to Functional Mock-up Units (FMUs) or Simulink models. The approach suffers from the poor behavior specification of the EAST-ADL by referencing the functions to external descriptions. Hence, in contrast to our approach, it is not integrated in the development process of the whole EEA model. Moreover, Simulink requires expensive licenses and supports only one MoC, namely continuous time with support for discrete time signals, whereas we benefit from the wide range of MoCs provided by PtII (see Background section for more details). This also applies for [16].

A recent approach relating to EAST-ADL was reported by the authors in [19], which is similar to the one in [17]. They additionally provide support for SystemC-AMS [21] to model and simulate analog/mixed-signal systems. Similar to our approach, they provide a library called *SystemComponentLibrary* for assembling the simulation model to increase development productivity. A text-to-model converter transforms the SystemC components into EAST-ADL behavioral *FunctionBehavior* and structural *HWComponentType* blocks. They apply their methodology to a brake-by-wire system. However, the generated EAST-ADL blocks still need to have a link to the external SystemC descriptions. Thus it is only partially integrated, but supported by the transformations. Moreover, the approach only covers the *Design Level* of EAST-ADL and does not mention links to lower levels. Our approach starts at a higher abstraction layer, linking the abstract *Logical Architecture* (LA, cp. Background section) description (*what* the system does) to the behavioral specification and more detailed lower level implementation layers, e.g. network topology and execution times. The *Logical Architecture* can be compared to the *Functional Analysis Architecture* layer above the *Design Level* of the EAST-ADL. We automatically synthesize the simulation model starting from this level, whereas the approach in [19] basically maps SystemC components to EAST-ADL blocks at the Design Level and finally parse the EAST-ADL model to construct the SystemC model.

The authors in [22] present an approach to synthesize component-based high-level and multi-domain AMESim [23] simulation models from a functional description based on the *Functional Model Language*. A set of available architectures is analyzed, which fit to the functional model and fulfil a set of requirements. Based on the specified flows, the proper simulation components of a given library are chosen. However, the work does not address the synthesis of lower level implementation details like network communication, execution time of components or the detailed algorithmic specification (*how* the components work).

Overall, none of the works presented support a fully integrated modelling process covering all levels of abstraction, synthesizing a high-level simulation model which covers several domains and considers lower level implementation details as well as non-functional properties at the same time. Furthermore, traceability across all abstraction levels in the EEA model is enabled as well as the reuse of existing LA descriptions by introducing a separate behavior specification layer.

Background

EEA-ADL

The *Electric/Electronic Architecture – Analysis Design Language* (EEA-ADL) [6] is another approach to holistically model EEA. This approach combines the EAST-ADL and AUTOSAR approaches in a single source EEA data model and also allows an ISO26262 compliant design. This is realized in the architecture design and analysis tool PREEvision [15], which is used for the proof-of-concept implementation described in this work. PREEvision v7.5 provides seven abstraction layers: 1. *Requirements*, *Customer Features* and *Feature-Functionality Network* which contain atomic requirements, features and their interaction. 2. The *Logical Architecture* is the starting point of the methodology presented in this work. It describes the vehicle's abstract logical function network and serves as a system decomposition of the later implementation in hardware (HW) and/or software (SW). It encompasses the specification of logical artifacts, e.g. *Sense*, *Actuation* and *Logical Functions* as well as their interconnection via *Logical Ports* and *Logical Assembly Connectors.* It also offers abstraction by introducing hierarchy via *Building Block* composites. Additionally, *Signal* definitions are performed describing which signals are exchanged between the logical blocks and later physically between the HW components 3. The *System Software Architecture* specifies AUTOSAR SW components as well as their interconnections via ports and their interfaces and complies with the AUTOSAR methodology. The components can be mapped to LA and HW components. 4. The *Hardware Component* and *Network Topology* describes all ECUs, sensors, actuators and their networking via bus systems used in the EEA. This layer also allows communication with conventional connections and the design of the abstract power distribution network. Here one connection abstracts from several wires or cables in the lower layers. 5. The *Electrical Circuit* and 6. *Wiring Harness* layers can be automatically synthesized from the network layer and contain the physical connections between the HW artifacts like wire types, schematic pin types, cable types and their physical properties like specific wire resistance. 7. The last layer realizes the *Geometrical Topology* of the EEA. Cross-layer links (mappings) between artifacts enable the comprehensive and consistent back-traceability across all modelling layers. A product line approach is used to support the complex EEA variant management. In addition to the EEA modelling, an integrated *Metric Framework* [24] is provided which enables the analysis and evaluation of architecture alternatives by customized metrics for non-functional properties like weight, cost or wiring harness diameter. This framework is also used for the implementation of the simulation synthesis methodology of this work. More details about the artifacts used for the implementation can be found in the Case Study section.

Heterogeneous Modelling and Simulation - Ptolemy II

Ptolemy II [14] is an open-source modelling and simulation framework for heterogeneous embedded systems with focus on concurrent components as well as the deterministic use and composition of heterogeneous MoCs. Deterministic in the sense that the same inputs always result in the same outputs. PtII follows an actor-oriented approach [25]. *Actors* are components that execute concurrently and communicate with each other via *ports* and *relations*. They can be *atomic* or *composite*. Atomic actors cannot be refined whereby composites enable hierarchical nesting of actors. The semantics for the execution of and communication between actors is governed by a specific MoC. The MoC within the model or a composite actor (sub-model) is realized by a component called *Director*. Distinct directors can be composed hierarchically in a single model at each level of the hierarchy. A sub-model controlled by an individual director is also called *domain* [12]. There are a variety of MoCs supported by PtII including discrete event (DE), which is especially suitable to model discrete systems like hardware architectures or communication networks. Besides DE, there exist other MoCs like continuous-time (CT) which is suitable for analogous components like sensors or physical dynamics, various dataflow MoCs for signal processing, finite state machines (FSMs), process networks for asynchronous distributed systems or synchronous/reactive for safety-critical concurrent software modelling. The hierarchical combination of MoCs with FSMs enables modal models [26]. These basically contain a FSM where each state can be refined with a sub-model containing a distinct director or again a FSM interoperating with the FSM director using well-defined interfaces. This allows the construction of *hybrid system* models, which capture discrete behavior with continuous physical processes with rigorous deterministic semantics. Furthermore, event based situations such as environmental uncertainties or faults and reactions to them can be handled by modal models [12]. This is especially true for the heterogeneous nature of EEA, which additionally adds a distributed communication network and the interaction with its environment. All these domains are covered by PtII and it provides an easily extensible actor library through its open-source nature and well-defined interfaces as well as documentation. A *concrete syntax* to represent models in PtII is the XML-based MoML (Modelling Mark-up Language), which provides a human readable format though eases portability, reuse, verbosity and model transformations. The *abstract syntax* [14] [27] of hierarchical actors with ports and interconnections is close to the LA of the EEA-ADL which makes it intuitive to map a PtII model to an LA model. All the previously described features are the reasons why we chose PtII as integrated simulation backbone of model-based EEA, e.g. instead of Simulink, which supports only one but sophisticated CT MoC but lacks for heterogeneous MoC composition.

Model-Based EEA Simulation Model Synthesis Methodology

The starting point for cross-domain simulation of model-based EEA is an EEA data model, which captures all relevant information necessary to synthesize an executable simulation model. This can be achieved with e.g. the EAST-ADL or the EEA-ADL previously described. In the following we provide an overview of our proposed methodology and necessary extensions to state-of-the-art domain-specific EEA description languages in order to enable early but holistic cross-domain simulation. We detail the individual components using the example of the EEA-ADL.

Overview

Because of the poor support for modelling behavior in the presented ADLs, an opportunity to explicitly specify the behavior is a necessary prerequisite in order to enable simulation. Therefore, we introduce a new layer, called *Behavioral Logical Architecture* (BLA), that refines the abstract LA's logical blocks (*what* the system does) with detailed behavior (*how* the abstract functions are working) by reusing actors from the *PtII Actor Library*. Mappings of BLA artifacts to the LA layer and from the LA layer to requirements or artifacts of lower layers enable the seamless traceability across all EEA layers. Additionally, it is a prerequisite to enable cross-domain simulation synthesis, because the mappings establish the links to the lower layers providing detailed domain-specific information.

The *E/E-Model Interpreter* extracts all necessary information from the relevant layers of the underlying EEA data model including the mappings as well as signal routing information in order to synthesize the simulation model. It serves as a front-end to interpret the underlying EEA data model and stores the meta-info such as artifact mappings in a database.

The *Generic Simulation Model Builder* uses the extracted E/E meta-info, translates/maps them to the target simulation model and synthesizes the unified cross-domain simulation model. It serves as a back-end for the target simulation model to be built. In this work this is a single XML file containing the MoML description for PtII.

The synthesized *Cross-Domain Simulation Model* is executed using PtII. It is twofold: it contains the behavioral simulation specified at the BLA layer. Beyond that it performs the domain-specific simulation of the lower layers, e.g. the mapping dependent network communication between the logical functions or physical/electrical processes as well as non-functional properties like execution time of the logical functions. The domain-specific and non-functional simulations are performed in an aspect-oriented way in combination with the behavioral simulation.

Finally, the integrated approach enables the feedback of the simulation results into suitable EEA metrics or benchmarks enabling iterative optimizations. It also enables the integrated visualization of the simulation data. The described methodology is shown in Figure 1. In the following we detail the components of our contributions with green background.

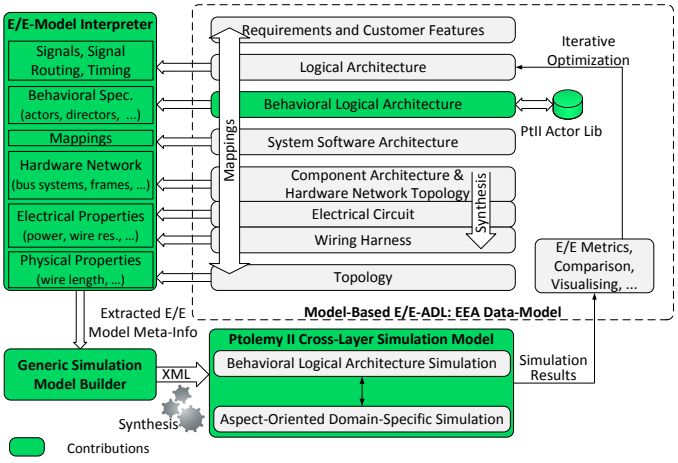

Figure 1. Overview of the proposed cross-domain simulation synthesis methodology of EEA.

Behavioral Logical Architecture Layer

This new layer called BLA introduces a refinement of the LA layer in the development process of the EEA by explicitly specifying the detailed functional behavior, plant and control models. Therewith, the same artifacts used to model the LA are reused, mainly *Sense*, *Logical Function*, *Actuation* and *Building Block* and their interconnection via *Logical Ports* and *Logical Assembly Connectors*. This has the advantage that the underlying EEA meta-model need not to be changed or extended by behavioral specific classes of the target simulation model. The LA's logical blocks are refined by instances of the PtII Actor Library (see PtII Actor Library section) which are encapsulated in a Building Block. Additional introduced mappings between the LA and BLA artifacts establish cross-layer links. In this way, the BLA layer together with the mappings enable a modular architecture where the LA can exist independently of the BLA but can be refined by the BLA where necessary. Additionally, several implementations of a single LA logical function can be exchanged by simply changing the mappings.

PtII Actor Library

The goal of this library is to increase productivity by reusing tested PtII actors ensuring that an engineer does not need to develop functions from scratch. The shipped PtII library contains a variety of actors necessary to model the detailed behavior, e.g. basic arithmetic, mathematical, logical actors but also domain-specific complex actors like FFT, filters, controllers, actuators like DC motors and many more. In addition, source actors to model stimuli for the behavior and sink actors to visualize, monitor or record/store simulation data are available as well. Recorded data can be used, e.g. in metrics in a later stage after the simulation to further evaluate the results or by using them in benchmarks. The library is extensible by either writing own Java actors or creating composite actors made of atomic and/or again composite actors and storing them in a MoML description file.

In order to use the actors at the BLA layer and to properly synthesize the MoML out of the BLA specification, the BLA artifacts have to be mapped to PtII artifacts. PtII follows a *class-instance* principle similar to object-oriented programming languages. The LA and hence the BLA follows a similar approach called *type-instance* principle. Therefore, actors are stored as logical function types, which define the actor class of the instance modeled at the BLA.

An *AbstractLogicalFunction* is an instance of a logical Sense, Function or Actuation block and represents an actor instance of a specific type in the BLA model. A Building Block represents an instance of a composite actor, independent on its defined type - except for Modal Models. Since the Building Block type defines only the interfaces and can have different implementations, it is useful to distinguish the same behavioral sub-system in the model with different realizations. This is also shown in our case study. Once a Building Block has been modeled at the BLA realizing a specific function it can be stored back in the library. This increases reuse of already created artifacts among engineers and across LA sub-systems.

Logical Provided-/Required Ports and Logical Assembly Connectors are straightforward mapped to output/input ports and to relations in PtII, respectively. Since most of the PtII actors offer parameters to configure them and also customized parameters can be added to a model or sub-model (composite), this possibility should also be present in the BLA. This is done via Generic Attributes of the EEA-ADL, which can be configured with at least a name, type and value and complies with PtII parameters. As directors are also a kind of attributes, but are only valid for a complete model or sub-models (composites) they should not be specified for a single actor. A different mapping compared to the parameters is necessary. Therefore, we introduce a custom defined *Domain Attribute* in the BLA, which is only valid for logical block owners to specify the director, i.e. the MoC, used to simulate the enclosing building block. Common attributes like start and stop time can be defined as well.

As stated previously, a building block type can be custom defined, except for Modal Models. As these are a special type of composites we interpret a building block of the type *ptolemy.domains.modal.ModalModel* as a modal model. Therewith, logical functions do not represent ordinary actors anymore, but states of the FSM. Their interconnection via ports and logical assembly connectors define the transitions between the states. A label of a logical assembly connector is then interpreted as the transition conditions and actions between the connected states. If a building block is used as a state, it represents a refinement state, either of a customized type or again a modal model type. That refinement then realizes a sub-model with a distinct domain or a hierarchical FSM respectively. The described mappings are summarized in Table 1., except for the modal model specific instance mappings.

Transition between LA and BLA

The refinement of LA artifacts and the modular approach of the behavioral specification at the BLA requires the establishment of cross-layer mappings to relate the LA and BLA artifacts to each other.

For the transition from the LA to the BLA layer we introduce two kinds of artifact mappings: *port prototype* and *block* mappings.

Table 1. Mapping of BLA artifacts to PtII artifacts.

EEA-ADL::BLA		**Ptolemy II**
Artifact	**Purpose**	**Artifact**
Abstract Logical Function	*Sense, Function* or *Actuation* logical blocks	Atomic or Composite Actor Instance
Abstract Logical Function Type	Specifies the type of a logical function	Actor Class
Building Block	Encapsulates atomic logical functions or again building blocks	Composite Actor
Building Block Type	Specifies the type of a building block	Composite Actor or Modal Model class
Logical Provided Port	Sender port	Output Port
Logical Required Port	Receiver port	Input Port
Logical Block Owner Domain Attribute	Custom attribute; specifies the MoC used to simulate this building block	Director
Logical Assembly Connector	Connects logical ports	relation
Generic Attribute	Provides parameters to logical blocks	Attribute / Parameter

Basically, each atomic logical block (Sense, Logical Function or Actuation) of the LA represents a borderline to the BLA and can be mapped to a building block, i.e. a composite actor, at the BLA comprising one or more interconnected (composite) actors. Therewith, it is possible to either perform a *1-to-1* or *n-to-1* mapping of LA atomic block(s) to one BLA building block. In addition, the BLA building block need to provide at least the number of input/output ports of the corresponding LA block(s). Building blocks of the LA are not allowed to be mapped, because they serve as an abstraction of the overall logical system decomposition. The *port prototype mappings* follow a 1-to-1 mapping principle exclusively in order to ensure the interface consistency between the LA atomic block(s) and the BLA building block. Another benefit of the 1-to-1 port prototype mappings is that it enables the automatic connection of the top-level target BLA building blocks, i.e. the top-level composite actors in the synthesized behavioral MoML, based on the connections of the corresponding LA atomic block(s). Hence only the internal actors and its interconnections have to be modeled as well as the port prototypes have to be mapped only once as long as the LA block's interfaces do not change. This automatic approach increases the reusability of already mapped building blocks in possibly several LA sub-systems or even in other product lines of the EEA-ADL without creating manual connections of the corresponding BLA top-level building blocks in that sub-systems. For simplicity reasons we refer to the port prototype mappings simply as port mappings. The described mappings are illustrated in Figure 2. and Figure 3.

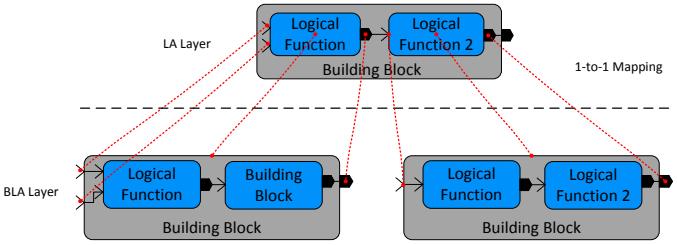

Figure 2. Illustration of the 1-to-1 block and port mapping.

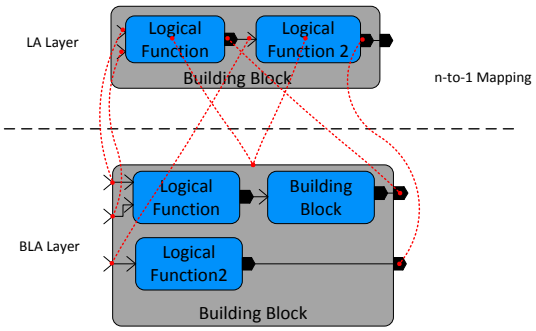

Figure 3. Illustration of the 1-to-1 port and n-to-1 block mapping.

Links to other Layers

By exploiting the previously introduced LA-BLA mappings together with the mapping of the LA to lower layers, it is possible to extract more detailed domain-specific information in order to decorate the pure behavioral BLA building blocks with it. We outline this with the help of Figure 4. Therewith, the dotted red lines represent the cross-layer mappings between the LA, BLA, Hardware Network Topology and Topology layers. For simplicity reasons, the port mappings are not shown and we assume that the LA logical block's port are properly mapped to the BLA building block's counterparts. Note that the shown model is not fully mapped and we are focusing on the LA *Building Block*, which contains two logical functions, namely *Logical Function* (LA-LF) and *Logical Function 2* (LA-LF2). *LA*-LF is mapped to the BLA *Building Block 2* (BLA-BB2), whereas LA-LF2 is mapped to the BLA *Building Block 3* (BLA-BB3).

LA-LF is additionally mapped to the *ECU1* at the HW layer. In contrast, LA-LF2 and therefore BLA-BF3, is distributed among *ECU2* and *ECU3*. ECU1 is connected to ECU2 and ECU3 via two separate CAN bus systems, namely *CAN1* and *CAN2*. Because of these mappings it follows, that BLA-BB2 has to communicate with BLA-BB3 via CAN1 and CAN2 (see green markers in Figure 4.). The direction of the communication is derived by the directed connection of LA-LF to LA-LF2. Moreover, we consider the LA *Actuation* block. This is mapped to the BLA *Building Block4* (BLA-BB4) and to the HW *Actuator* block. In this case, BLA-BB4 is communicating with the Actuator via a conventional connection (see orange marker in Figure 4.). Thereby, a conventional connection not only transports information but also energy, or more concrete, currents. The capability of the EEA-ADL to model the power distribution network additionally enables the possibility to derive electrical properties of the whole HW network.

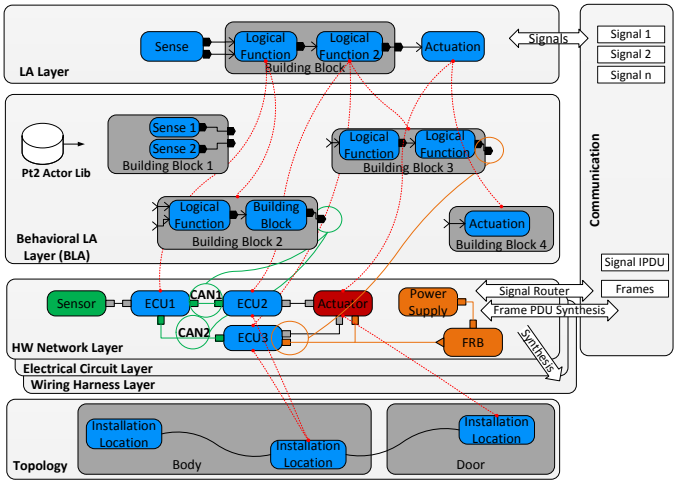

Figure 4. Cross-layer links between the BLA and lower layers.

Overall, logical assembly connections represent not only pure data flow. As previously described they are decorated with domain-specific aspects, which need to be extracted in order to enable cross-domain simulation of the EEA. The extraction of these aspects is performed within the *E/E-Model Interpreter*.

E/E-Model Interpreter

The E/E-Model Interpreter serves as a front-end interpreter of the underlying EEA data model that extracts and collects all necessary information as E/E meta-info. This meta-info comprises mainly those shown in Figure 1. The LA together with the BLA layer and its mappings to each other provide the behavioral specification which is synthesized according to the modeled instances of PtII actors (see section PtII Actor Library). The LA additionally provides information about latency times of the logical blocks as well as the signals which are exchanged between the logical blocks and later between the mapped HW components (signal routing). Note that the LA-BLA mappings ensure that all meta-info valid for the LA artifacts do also apply for the corresponding BLA artifacts. The proper extraction of the domain-specific information like bus communication or electrical/physical properties requires two input prerequisites for the E/E-Model Interpreter to be performed by the EEA-ADL on the EEA data model: 1. The *Signal Router* and 2. The *HW Network Synthesis*. These are described briefly in the next sub-section. Afterwards we detail the extraction of these domain- specific meta-info. Finally, note that we intentionally not considered mappings to and information from the System Software Layer, as we are starting from the high-level logical system decomposition and focusing on a behavioral logical simulation, independent on the realization of a function in HW or SW.

Input Prerequisites

In order to be able to extract the network communication information, the signals defined at the LA have to be transformed into physical *Signal Transmissions* (STs) which in turn have to be routed between the hardware components. This is done via a *Signal Router*. It automatically calculates the necessary route of a ST based on the LA-to-HW mappings and a configurable target cost function, inserts

necessary Gateways (GWs) and creates the corresponding *Independent Protocol Data Units* (IPDU) and finally the *Frames* to be transmitted. This is sketched on the right-hand side of Figure 4.

The extraction of electrical and physical properties such as pin types, wire types etc. requires the synthesis of the abstract HW Network layer into the more detailed Electrical Circuit and Wiring Harness layers. This is illustrated by the three stacked layers in Figure 4.

Execution Time

If there is a possibility to specify latency or execution times of artifacts, this can be used as additional non-functional simulation property. The EEA-ADL supports this with an object *Latency Time* comprising three kinds of latencies on its core artifacts including LA artifacts: *minimum, nominal* and *maximum* latency time. Hence, we can annotate and extract this latency time object from our behavioral BLA building blocks to synthesize execution times of actors into our simulation model. Since we allow an n-to-1 mapping of LA logical blocks to one BLA building block, the execution time of a single building block is the sum of the execution times of a specific kind of all mapped LA logical blocks.

Bus Communication Extraction

The Signal Router additionally creates a special kind of mapping, which links the logical provided ports of the source LA logical block and the connected required ports of target LA logical blocks with the created set of STs at the HW layer. This is necessary because the same logical function could be mapped to several instances of a specific ECU, e.g. four wheel ECUs. Then, the signal router has to infer and map four STs, one for each wheel ECU, for the signal specified at the logical provided port. The same holds for logical required ports.

As discussed with Figure 4., assemblies between two logical ports not only represent pure data flow, but can contain bus communications. This is the case if a logical port has a mapping to a ST, which belongs to a bus system. This is true for the example used in Figure 4, where the mapped source and target ECUs are directly connected. Hence, each of the logical port to ST mappings directly contains a ST, which belong to the CAN1 and CAN2 bus system. However, that example is a simplified one. Because it is possible that the source and target ECU are not directly connected, a ST has to pass through one or more Gateway ECUs to reach its target. In this case, the mapped STs are routed by the Gateways towards the target ECU with the help of *Gateway Routing Entries* (GREs) in a Gateway, which define the incoming ST and the corresponding outgoing ST. The latter can belong to the same bus system as the incoming ST, a different bus system or a conventional connection. This is shown in Figure 5. The mapped ST1 of the LF's provided port has to pass the two Gateways GW1 and GW2 until it reaches the ST3, which is mapped to the target required port of LF2.

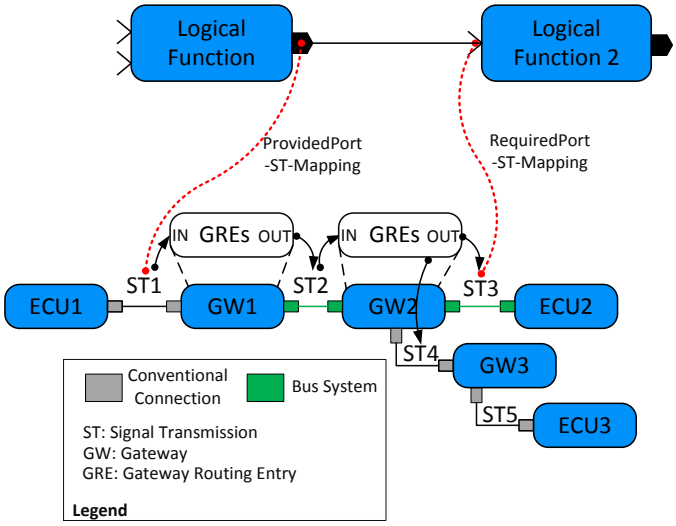

Figure 5. Example ECU network to extract bus communication information between two logical ports.

Hence, to find all possible bus communications, we first have to analyze the directly mapped STs of connected logical ports. Additionally, we need to traverse the ECU network over all possible GWs, more concrete, all GREs, starting from a provided port's ST until one of the target STs are reached. All intermediate STs which are found in between and belong to a bus system are extracted together with its bus system as a bus communication to be simulated between the two logical ports of the mapped BLA building blocks. We use a multi-source breadth first search (BFS) algorithm to extract these intermediate STs with its bus systems for all STs which are not directly mapped to the logical ports. Therewith, the GREs represent the vertices and the STs the edges. The source vertices are the GREs, which contain the directly mapped STs as incoming ST. In Figure 5., these are the GREs of ST1. The target STs serve as abort criterion. The extraction of bus communications between a logical provided port (pPort) and a logical required port (rPort) is shown in Algorithm 1.

Lines 1-3 in Algorithm 1. filter the given sets of mapped STs of the given pPort and rPort for STs which belong to bus systems and unifies them as exclusive disjunction in a new set $\boldsymbol{ST}^{\boldsymbol{Bus}}_{\boldsymbol{pPort,rPort}}$. Lines 4-21 represent the extraction of the intermediate STs and their belonging bus systems as described previously. The function call `BFS_GREs(` $\boldsymbol{GREs\_In}$, $\boldsymbol{ST}_{\boldsymbol{RequiredPort}}$ `)` represents the multi-source BFS for all GREs of those pPort's directly mapped STs which are incoming STs. It returns a set $\boldsymbol{ST}^{\boldsymbol{Bus}}_{\boldsymbol{intermediate}}$ of all STs belonging to intermediate bus systems until a target ST is reached as well as the set of corresponding bus systems $\boldsymbol{BusSys}_{\boldsymbol{pPort,rPort}}$.

Algorithm 1. ExtractBusCommunications

Input: pPort: Logical Provided Port
Input: rPort: Logical Required Port
Input: $ST_{RequiredPort}$: Set of mapped STs of rPort
Input: $ST_{ProvPort}$: Set of mapped STs of pPort
Output: $BusSys_{pPort,rPort}$: Set of bus systems between pPort and rPort
Output: $ST^{Bus}_{pPort,rPort}$: Set of STs belonging to bus systems between rPort and pPort

1: $\boldsymbol{ST^{Bus}_{ProvPort}} \leftarrow$ filter $\boldsymbol{ST_{ProvPort}}$ belonging to bus communications
2: $\boldsymbol{ST^{Bus}_{RequiredPort}} \leftarrow$ filter $\boldsymbol{ST_{RequiredPort}}$ belonging to bus communications
3: $\boldsymbol{ST^{Bus}_{pPort,rPort}} \leftarrow \boldsymbol{ST^{Bus}_{RequiredPort}} \oplus \boldsymbol{ST^{Bus}_{ProvPort}}$
4: **for each** $st \in \boldsymbol{ST_{ProvPort}}$
5: $BusSys_{st} \leftarrow$ getBusSystem(st)
6: **if** $BusSys_{st} \neq \emptyset$ and $\nexists\, BusSys_{st}$ in $\boldsymbol{BusSys_{pPort,rPort}}$ **then**
7: Add $BusSys_{st}$ to $\boldsymbol{BusSys_{pPort,rPort}}$
8: **endif**
9: $\boldsymbol{GREs\_In} \leftarrow$ getRoutingEntriesIn(st)
10: $\boldsymbol{ST^{Bus}_{intermediate}} \leftarrow$ **BFS_GREs(**$\boldsymbol{GREs\_In}$, $\boldsymbol{ST_{RequiredPort}}$**)**
11: **for each** $st^{Bus}_{intermediate} \in \boldsymbol{ST^{Bus}_{intermediate}}$
12: **if** $\nexists\, st^{Bus}_{intermediate}$ in $\boldsymbol{ST^{Bus}_{pPort,rPort}}$ **then**
13: Add $st^{Bus}_{intermediate}$ to $\boldsymbol{ST^{Bus}_{pPort,rPort}}$
14: **endif**
15: $BusSys_{st} \leftarrow$ getBusSystem($st^{Bus}_{intermediate}$)
16: **if** $\nexists\, BusSys_{st}$ in $\boldsymbol{BusSys_{pPort,rPort}}$ **then**
17: Add $BusSys_{st}$ to $\boldsymbol{BusSys_{pPort,rPort}}$
18: **endif**
19: **endfor**
20: **endfor**
21: **return** $\boldsymbol{BusSys_{pPort,rPort}}$ and $\boldsymbol{ST^{Bus}_{pPort,rPort}}$

Without loss of generality, we are focusing on extracted CAN bus systems in this work. If there are CAN bus systems found between the two logical ports rPort and pPort, the CAN specific bus properties like baud rate, frame format, transmitting policies etc. are extracted. The corresponding STs in the set $\boldsymbol{ST^{Bus}_{pPort,rPort}}$ are used to extract the contained CAN frames which in turn are used to extract their frame size and priority.

Electrical and Physical Properties

The HW network layer represents only abstract connections between ECUs or within the power distribution network. In combination with the tool-support by synthesizing the electrical and wiring harness layer, more detailed information about the connections can be extracted (see section Background). The mappings to the HW network layer are automatically propagated to the synthesized artifacts. The power distribution network model contains components like *Power Supply*, e.g. battery, *Fuse Relay Boxes* (FRBs), *Ground Points* as well as internal passive components like resistors, capacitors or inductors. The combination of both enables the possibility for analog current and voltage simulations of conventional connections or parts of the wiring harness. Exploiting the mappings to the Geometrical Topology layer, physical properties like the realized wire/cable lengths or cross-section of a specific conventional connection can be determined as well. With this, the current simulation can be even more detailed by taking wiring losses into account. However, we will investigate this in future work and is not further addressed in this work.

Generic Simulation Model Builder

This component serves as a back-end, which uses the behavioral specification as well as the extracted domain-specific and non-functional E/E meta-info to build the unified cross-domain target simulation model, i.e. the PtII model. A single XML file containing the PtII MoML is synthesized. Therewith, the synthesis is twofold:

1. Synthesis of the *behavioral simulation* as specified in the BLA according to the LA-BLA cross-layer links and the mappings of BLA artifacts to PtII artifacts.
2. Synthesis of *aspect-oriented domain-specific simulation* decorating the behavioral actors and/or the data flow between them.

This synthesis flow is depicted in Figure 6. After the selected LA sub-model is interpreted and the E/E meta-info is available, the BLA synthesis of the detailed behavioral simulation model is performed. Each of the top-level building blocks of the BLA including all its children together with their parameters are built according to the mapping rules described in section PtII Actor Library. This stage is a recursive procedure. If a child is another building block, a possible containing director is built and a recursive call is performed on this building block. This is repeated until the bottom of the current building block's hierarchy is reached. After all children at a certain recursive stage are built, they are connected according to the logical assembly connections in the BLA. This purely behavioral synthesis is illustrated by the blue procedure in Figure 6.

The synthesis of the domain-specific and non-functional simulation sub-models are based on a unique feature of PtII. Actors and/or ports can be decorated by so called *aspects* [14], which are based on *quantity managers* introduced in [28]. These are components in a model serving as a mediator to another model in order to refine the original one with a specific aspect, e.g. with a communication aspect. This decoration of actors with aspects is covered by the *Decorate BLA Top-Level Artifacts* process depicted in Figure 6. Note that these aspects are only necessary for the top-level BLA building blocks, i.e. composite actors. The reason is that the domain-specific and non-functional properties are valid for the LA logical blocks and therefore for the entire mapped BLA building block.

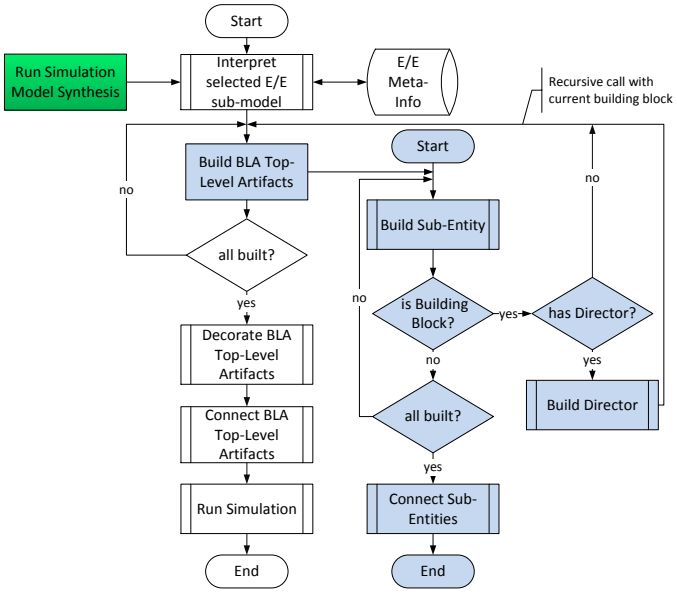

Figure 6. Synthesis flow of the unified PtII MoML simulation model.

PtII provides capabilities for communication and execution aspects, which can be, like ordinary actors, atomic or composite. We leverage composite aspects in order to encapsulate the domain-specific or non-functional aspects. This allows an arbitrary refinement of the composite aspect with atomic aspects, complete sub-models or a combination of both. These aspects typically represent timed simulations in order to respect execution and communication delays.

Concerning the communication aspect, i.e. in our case the CAN communication, we leverage an abstract CAN atomic aspect provided by the PtII actor library. We encapsulate it in a composite communication aspect actor which is built for each CAN bus system extracted by the E/E-Model Interpreter. The individual bus system properties like baud rate are set as parameters on the atomic CAN aspect. A DE director is used within the composite aspects, as a timed simulation is required. The identified logical connections which belong to the bus systems, more concrete, the receiving ports, are decorated with the appropriate CAN aspect. The receiving data at the appropriate input port is mediated to the CAN composite aspects. Therewith, the ports are decorated with the frame priority (ID) and frame size of the CAN frame extracted from the EEA data model. We extended the base composite communication aspect of the actor library by a *composite CAN communication aspect*, which adds the frame ID and size as parameters to the mediated data in order to use them in the composite aspect. So far, the CAN atomic aspect of PtII only supports frames with fixed frame sizes of 108 *bits* and 128 *bits* according to the standard and extended frame format, respectively. However, a variable frame size is possible to extend in the aspect implementation of the PtII actor library.

Regarding the execution aspects, the execution of an actor which has an annotated latency time greater than zero is mediated to a composite execution aspect. The latter delays the received data sent by the source actor according to the maximum latency time annotated to that source before the functional behavior simulation proceeds.

In addition, concerning the possible current and voltage simulations of the wiring harness, we introduce the notion of a *physical composite aspect*, which, analogues to the other aspects, encapsulates these continuous simulations. Therefore, this can be extended in a modular way without touching the functional behavior model. We will investigate this aspect in future work.

Finally, after all top-level building blocks are built and decorated by the aspects, they are automatically connected by means of the introduced automatic approach described in section Transition between LA and BLA. After that, the simulation model is fully synthesized and ready to run.

Case Study

The performed case study presented in this section serves as a proof-of-concept of our methodology. We implemented our approach and the case study in terms of a simplified ACC application within PREEvision. Two goals are pursued within the case study: 1. Demonstration of the correct synthesis of the PtII MoML containing the modeled ACC application as well as the extracted E/E meta-info across the LA, BLA and HW network layers. This especially includes the automatic connection of the top-level composite actors as well as the CAN communication and execution aspects; 2. Demonstration of the benefits of the separate BLA layer by exploring realization alternatives of the ACC application with low effort.

Implementation

The E/E-Model Interpreter and the Simulation Model Builder each are implemented as an additional Eclipse plug-in within the Metric Framework of PREEvision. The main metric used artifacts are *Model Query* and *Calculation* blocks. The Model Query block has access to all artifacts in the EEA data model and is used to fetch the LA model of interest to be synthesized. They have output ports to transfer the artifacts to other blocks using data flow semantics. The benefit is a selective choice of the LA (sub-) model of interest. The plug-in containing the E/E-Model Interpreter and the Simulation Model Builder is realized as a customized metric calculation block. The latter contains Java code implementing its behavior and can provide I/O ports as well. It receives the selected LA artifacts of the model query and sends the synthesized MoML to the simulation executer. The execution of the simulation model is realized in an additional customized calculation block plug-in, which receives the synthesized MoML file, executes the simulation and opens a PtII simulation view. The latter contains possible visualizations, if the corresponding sink actors are modeled, as well as a run control panel.

EEA Model Synthesis

The ACC use case is modeled at the LA, BLA and HW network layer. The setup is depicted in Figure 7. Thereby, the LA contains the abstract ACC system behavior made up of four blocks: the two sense functions *GetRadarSpeed* and *GetWheelSpeed* provide the velocities of the leading vehicle and the measured speed of the modeled vehicle,

respectively, to the *ACC* controller function. The latter calculates the necessary acceleration and provides it to the *setWheelSpeed* actuation function which drives the input acceleration to the appropriate speed. GetWheelSpeed and setWheelSpeed have additionally set the *latency time* attribute, each equals to $0.05s$.

The Sense, Function and Actuation blocks of the LA are mapped to the corresponding Sensor, ECU and Actuator HW components as shown in Figure 7. Note that Sensor and Actuator blocks can contain both hardware components and processing units. It shows, that the sensor blocks *WheelSpeed* and *RadarSpeed* have to communicate over the two bus systems *HS-CAN* (High-Speed-CAN) at $500kbps$ and *LS-CAN* (Low-Speed-CAN) at $125kbps$ to reach the target ECU *ACC*. These bus communications together with its appropriate frames *RadarSpeed* and *WheelSpeed* are synthesized by the Signal Router.

The BLA layer contains a logical function package called ACC which contains the detailed functional behavior refinement of the individual LA blocks. The *LeadingPlatoon* building block simulates a single leading vehicle or platoon with a CT director as domain attribute. It provides three outputs, its current speed, position and acceleration. Therefore it is mapped to the LA block GetRadarSpeed, which provides the measured radar speed of the leading vehicle. Here especially, the introduced port mapping is important, as the target BLA block provides more ports than the one of the LA block. Hence, the only port of GetRadarSpeed is mapped to the appropriate *speedLeader* port of the building block (cp. Figure 7.). The *WheelControllerDE* block implements both driving the received acceleration to the appropriate speed and providing the measured speed. Thus, the GetWheelSpeed and setWheelSpeed blocks are mapped to the building block in an n-to-1 mapping fashion providing the ports of both former blocks, which in turn are mapped accordingly to the LA ports (cp. Figure 7.). The ACC function is implemented as a building block of type *AccController*. Here we exploit the possibility to provide alternative realizations of the same building block type, but with the same interface, namely *P_Controller* and *IDM_Controller*. They require the speed of the leading vehicle as *desiredSpeed* (from GetRadarSpeed) and the own *measuredSpeed* (from GetWheelSpeed) to calculate the provided *acceleration*.

Since the synthesized MoML is a purely textual XML description, we visualized the top-level MoML by means of the PtII GUI Vergil in order to illustrate and verify the synthesis result. It is depicted in Figure 8. Because of the port mappings and the connections between the LA blocks it follows that WheelControllerDE and AccController will build a closed feedback loop, which is synthesized and automatically connected. The *speedLeader* port is connected to the *desiredSpeed* port. From the HW network mappings it follows, that the communication of both the LeadingPlatoon and the WheelControllerDE between the P_Controller has to be mediated by a communication aspect for the HS- and LS-CAN bus. This information is extracted by means of Algorithm 1. The synthesized result is represented by the LS-CAN and HS-CAN composite aspects at the top of Figure 8. They each contain the CAN atomic aspect with a baud rate parameter set to $125kbps$ and $500kbps$ respectively. The communication mediation is highlighted by the orange input ports decorated with the frame ID and size of the corresponding frames of each CAN bus. In addition, the WheelControllerDE is decorated with an execution aspect delay of $0.1s$, because each of the mapped LA blocks specifies a latency time of $0.05s$. The green shape highlights the enabled execution aspect *WheelControllerDE_ExecutionAspect* at the top of Figure 8. Finally, the top-level DE director with a simulation stopTime parameter as well as parameters for max. and min. acceleration limiting the P_Controller are shown. They were set properly as specified with the generic attributes at the BLA layer.

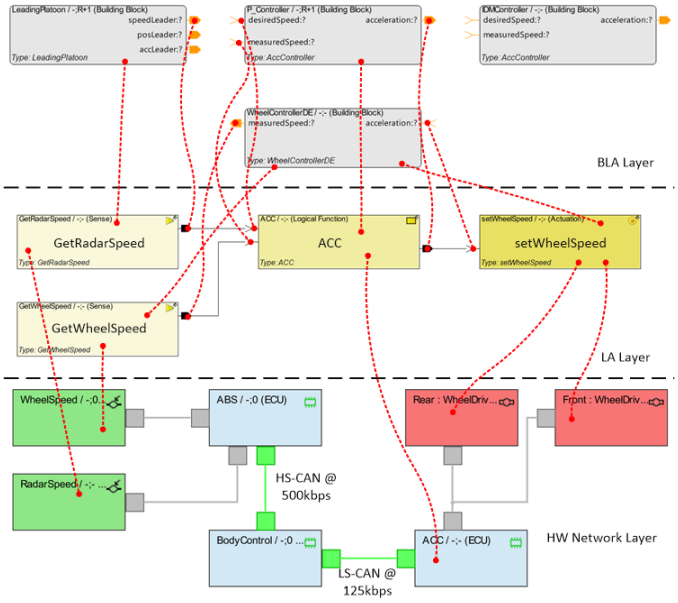

Figure 7. ACC case study setup showing the abstract LA model (middle), the BLA model (top) and the HW network model (bottom) as well as the cross-layer mappings (dotted red lines).

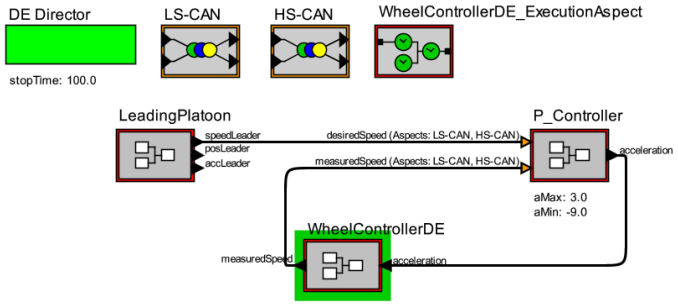

Figure 8 Synthesized PtII MoML of the BLA containing CAN communication, execution aspects and parameters. Visualized in the PtII GUI Vergil.

In the next sections, we simulate the ACC behavior with both the P_Controller and the IDM_Controller by simply changing the mappings. The influence of the communication and execution aspects on the different realizations is also analyzed.

ACC Simulation: Realization I – P_Controller

In the first case, we simulate the ACC with the P_Controller, a simple proportional controller with a loop gain of 10, based on a tested PtII Car Tracking demo model. The main parameters and attributes

relevant for the following conducted ACC simulations (also for the IDM_Controller) are summarized in Table 2. Note that in contrast to the synthesized MoML in Figure 8. we only simulate the CAN aspects in this first simulation scenario and set the latency times to zero. In the implementation of the P_Controller we modeled *TimedPlotter* and *TimedDisplay* actors in order to visually verify and monitor the receiving speed values of the leading vehicle and the measured speed resulting from the calculated acceleration value of the P_Controller. Additionally, the distance is calculated by means of the received speed values, but does not has an impact on the acceleration calculation. We set an initial distance of $100m$. The speed values (excerpt for illustration reasons) and the distance are plotted in Figure 9. and Figure 10., respectively. Despite the two LS- and HS-CAN communications of both the desired speed of the leading vehicle and the measured speed in the feedback path to the P_Controller, the speed values are calculated correctly. This is clearly shown in Figure 9., where the measured speed inside the P_Controller is closely following the one of the leading vehicle. The reason is that only a small delay caused by the CAN communication impacts the controller, which is compensated after an initial swinging. We monitored a first event of the desired speed of the leading vehicle at $1.08ms$, which exactly matches the delay of the 108 $bits$ sized frame RadarSpeed over both CAN busses:

$$\Delta t_{RadarSpeed} = 108\ bit * \left({}^{1}/_{500kbps} + {}^{1}/_{125kbps}\right) = 1.08ms \qquad (1)$$

The first event of the measured speed in the feedback path raised at $1.944ms$. This delay is greater, because frame collisions occurred on both busses. This is the case, since the frame IDs of both frames are identical (cp. Table 2) and both the RadarSpeed and WheelSpeed frames requested a CAN communication on both busses at simulation time $0.0s$. However, because the LeadingPlatoon actor is executed before the WheelControllerDE in the execution order (determined by a topological sort of a directed acyclic graph of the actors [14]), the RadarFrame is served first. The distance depicted in Figure 10. closely alternates around the initial distance of $100m$ between about $-0.5m$ and $+2.6m$, because the P_Controller simply tries to reach the desired speed without taking the distance into account.

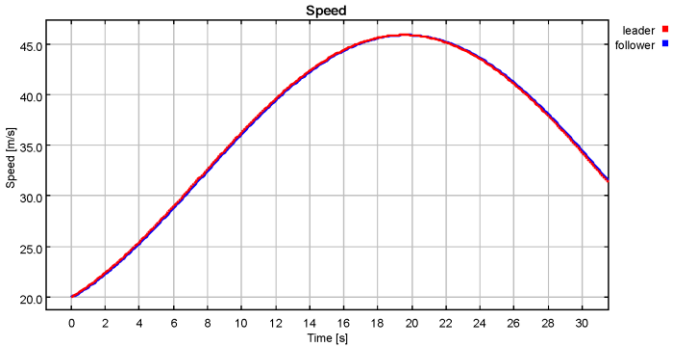

Figure 9. Plotted speed of the simulated leading vehicle (red) and the follower (blue) calculated by the P_Controller and WheelControllerDE (excerpt).

ACC Simulation: Realization II – IDM_Controller

In this simulation scenario, we exchanged the P_Controller with the IDM_Controller by simply changing the mapping and running the simulation synthesis once again. It realizes the *Intelligent Driver Model* (IDM) [29] car following model, which additionally takes the gap to the leading vehicle by means of coupled ordinary differential equations into account. The used parameters are shown in Table 2. Note that the parameter v_0 is the desired speed of the IDM on a free road and not the input port to the IDM_Controller. In this scenario, we simulate the impact of the execution time delay of the WheelControllerDE, and compare the results with an additional run of the P_Controller considering the execution time, too.

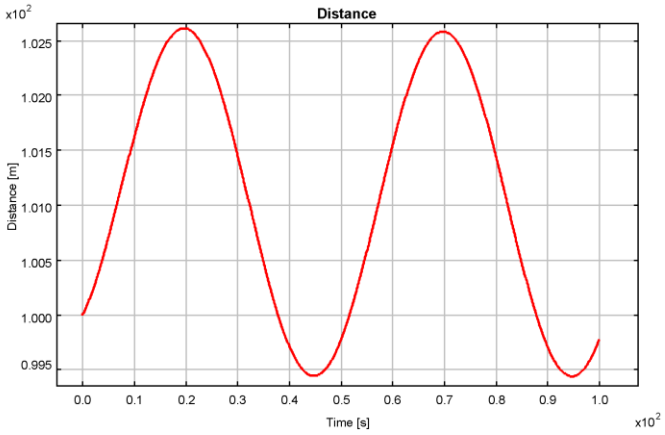

Figure 10. Plotted distance between the simulated leading vehicle and the follower.

Table 2. Artifact attributes used in the case study.

Artifact	Attribute	Value
LA		
GetRadarSpeed	Latency Time	0.05s
setWheelSpeed		0.05s
HW Network Layer		
HS-CAN Bus	Baud Rate	500kbps
LS-CAN Bus		125kbps
CAN-Frame RadarSpeed	Frame ID	0x123
CAN-Frame WheelSpeed		0x123
BLA		
P_Controller	aMax	3.0m/s²
IDM_Controller		3.0m/s²
P_Controller	aMin	-9.0m/s²
IDM_Controller		-9.0m/s²
P_Controller	Domain	CT Director
IDM_Controller		DE Director
WheelControllerDE		DE Director
LeadingPlatoon		CT Director
IDM_Controller	Acceleration acc	3.0m/s²
	Deceleration dec	4.0m/s²
	Headway Time T	1.5s
	Initial Distance d	100m
	Minimum Gap s_0	2m
	Desired Velocity v_0	50m/s
	Acceleration Exponent δ	4

In Figure 11., an excerpt of the accelerations calculated by the P_Controller and IDM_Controller are plotted. It clearly shows the impact of the additional execution delay of $0.1s$ in the feedback path of the P_Controller, which is heavily swinging. In contrast, the IDM is much more robust against the additional delay and calculates the proper acceleration values to smoothly follow the leading vehicle. This is underlined by the speed values depicted on the left-hand side of Figure 12. On the right-hand side the distance is illustrated. It shows, that the IDM tries to approach the leading vehicle starting from the initial distance of $100m$ until a certain distance (dependent on the headway time parameter T), while the P_Controller simply tries to match the input speed of the leading vehicle.

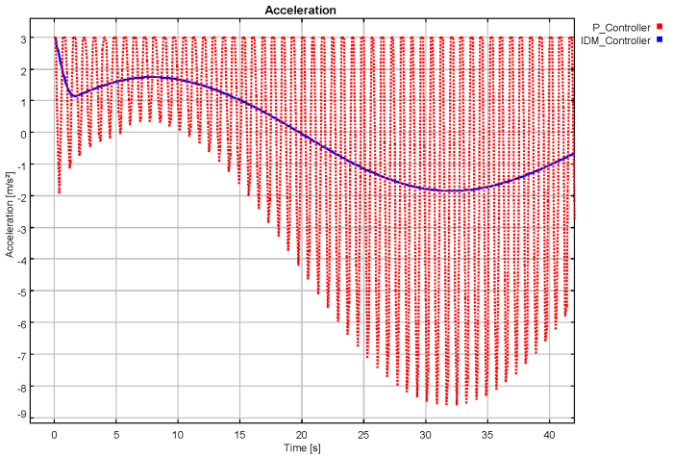

Figure 11. Comparison of the accelerations (excerpt) calculated by the P_Controller (dotted red) and IDM_Controller (blue) with impact of CAN and execution time delay.

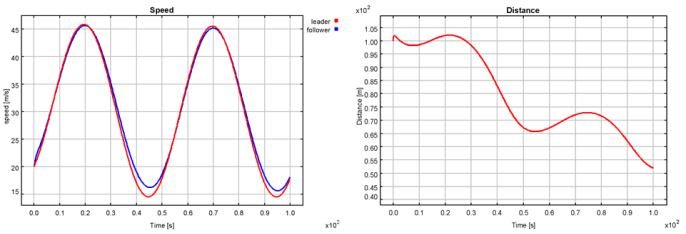

Figure 12. Left: Speed of the leading vehicle (red) and the IDM controlled follower (blue). Right: Distance between the leading vehicle and the follower using the IDM_Controller.

Discussion

To verify the synthesized MoML it was visualized in the PtII GUI Vergil and compared across all hierarchy levels to the modeled EEA artifacts, attributes and the information derived from the cross-layer mappings, e.g. CAN communication and execution times. For space reasons, only the synthesized top-level model is depicted in Figure 8. The presented case studies showed that the EEA model is synthesized correctly containing all information specified across the layers LA, BLA and HW network including the automatic connection of the top-level BLA composite actors as well as the composite aspects. To verify the ACC simulation results of the synthesized MoML we used equivalent models manually created within Vergil which represent the reference models of the expected synthesis outcome derived from the EEA model as described in the EEA Model Synthesis sub-section. We compared the produced plot data and especially the CAN and execution time delays. Therewith, no deviations could be observed and thus verified the simulation results of the synthesized MoML. Hence, an early but holistic view on the system dynamics is provided by performing the cross-domain simulation. The case studies also showed the flexibility of our approach in terms of design space exploration. By simply changing the mapping of the ACC logical function between the P_Controller and IDM_Controller an early analysis of the different delay impacts showed, that the IDM controller is the more reliable and suitable solution. Note, that although we modeled a simple WheelController without looking at detailed dynamics of a wheel drive model, the BLA leverages the heterogeneous model composition capabilities of PtII to capture more elaborate hybrid system models including plant models. Beyond that, model changes inferred by the underlying EEA model (e.g. CAN parameters such as baud rate, frame IDs or ECU mappings yielding in possibly different used bus systems) are automatically synthesized and do not need to be adjusted manually in the target simulation model. This strongly decreases model maintenance efforts compared to separate running modelling and simulation processes.

Furthermore, despite our methodology is presented using the example of the EEA-ADL, it is more generic and transferrable to other EEA ADLs, for two reasons: 1. the reuse of the same artifacts at the BLA layer to refine those from the more abstract higher logical layer avoids changes of the underlying EEA meta-model. 2. the meta-model artifacts from the source EEA model are widely abstracted via templates in the E/E-Model Interpreter front-end. Thus, it eases the portability to other meta-model classes of ADL layers comparable to the LA layer, e.g. the Functional Analysis Architecture of the EAST-ADL. The transfer to other domains like avionics is also possible. A necessary prerequisite is that the source meta-model supports the notion of actor-oriented design made up of hierarchical blocks communicating via ports. Similarly, in case of the simulation model builder, different back-ends for the target model can be used, since we are providing a reference to a generic model builder object in the interpreter, which implements the appropriate back-end. Thus, target models different to the MoML, e.g. SystemC, could be implemented. Moreover, the encapsulation of aspects in composite aspects allows an arbitrary refinement of the composites down to possible co-simulation of the abstract model with detailed domain-specific tools. This is done by exploiting *HlaComposite*s in PtII developed and used within our previous works [30] [31] [32]. The used IDM model in the case study, for instance, was verified within [30] by means of a co-simulation of PtII with a traffic simulator and an implementation of the IDM running on a SystemC multi-core model.

One drawback currently is the necessity of the manual creation of mappings and thus performing the iterative optimization (cp. Figure 1.). This can be addressed by model metrics which automatically find proper (re-) mappings and perform the model operations. The PtII actors used in the BLA currently are created manually in the library. To further increase the development productivity, we will implement an import/export functionality of PtII actors. Additionally, we will add the support for modal models as described in the PtII Actor Library

section. The latter in combination with aspect actors will both enable the creation of and reaction to event based situations and greatly help handling their complexity in a modular way. E.g. the detection and reaction to a factually flawed ACC information can be encapsulated in composite aspect containing a modal model not touching the structure of the original behavioral model. Another drawback is the dependability on the signal router and frame synthesis as input for the bus communication extraction. This can be addressed by a more generic signal tracing and frame synthesis algorithm directly integrated in the E/E-Model Interpreter. Thus the approach gets more independent on the underlying EEA data model, since the routing is no input to the front-end anymore.

Conclusion and Future Work

Current domain-specific ADLs for modelling EEA lack for the possibility to explicitly specify an executable behavior in an integrated manner. Additionally, the modelling and simulation of EEA are often separate running processes. Within this work we presented a novel integrated approach to synthesize an executable high-level simulation model starting with an abstract logical function architecture of a single source EEA model. We introduced a new abstraction layer called BLA which refines the LA and specifies the functional behavior, control and plant models by means of an extensible PtII actor library. The BLA layer is seamlessly integrated within the development process of the state-of-the-art EEA-ADL. We extended existing cross-layer mappings and leveraged the latter as well as a signal router and HW network synthesis of the EEA-ADL to link the BLA model with lower layer implementation details like bus communication, execution time delays and electrical properties. This enables the synthesis and integrated conduction of a unified cross-domain simulation model. An ACC case study proved the concept of synthesizing the unified simulation model with CAN bus communications and execution times. Early but holistic analysis by means of integrated high-level simulations of different ACC controllers which are influenced by CAN and execution time delays allowed an early decision on the most suitable realization by simply changing the cross-layer mappings. The latter additionally enable the seamless traceability e.g. of the BLA behavioral artifacts to the executive ECUs or the respective requirements. Finally, the modular approach enables the transfer to other EEA ADLs and application domains.

We will further develop our approach by considering the synthesis of electrical and physical properties and performing electrical simulations of the wiring harness. Suitable metrics using the simulation results for iterative optimizations can be addressed. The support and integration of domain-specific models specified in external expert tools will also be investigated by means of the PtII co-simulation capabilities developed in our previous works in order to further increase reuse and productivity. We will also extend our case studies, e.g. by integrating more elaborate wheel drive plant models incorporating with the WheelController or considering event based situations with the help of modal models.

References

1. C. Buckl et al., "The software car: Building ICT architectures for future electric vehicles," in *Electric Vehicle Conference (IEVC), 2012 IEEE International*, March 2012, pp. 1-8.
2. F. K. Bapp et al., "Adapting Commercial Off-The-Shelf Multicore Processors for Safety-Related Automotive Systems Using Online Monitoring," in *SAE Technical Paper*, Apr. 2015. [Online]. http://dx.doi.org/10.4271/2015-01-0280
3. K.I. Voigt, *Industrielles Management - Industriebetriebslehre aus prozessorientierter Sicht*. Berlin Heidelberg, Germany: Springer Verlag, 2008.
4. OMG. Systems Modeling Language SysML®, formal/2015-06-03. [Online]. http://www.omg.org/spec/SysML/1.4/PDF
5. OMG. UML Profile for MARTE: Modeling and Analysis of Real-time and Embedded Systems, formal/2011-06-02. [Online]. http://www.omg.org/spec/MARTE/1.1/PDF
6. J. Matheis, "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.
7. EAST-ADL Association. (2016) EAST-ADL Association. [Online]. http://www.east-adl.info/Specification/V2.1.12/EAST-ADL-Specification_V2.1.12.pdf
8. AUTOSAR. AUTOSAR 4.2 (Automotive Open System Architecture) Specifications. [Online]. http://www.autosar.org
9. The Mathworks, Inc. Simulink 2016. [Online]. http://www.mathworks.com/products/simulink/
10. Modelica Association. Modelica Standard Library. [Online]. https://modelica.org/libraries/Modelica/
11. J. Eker et al., "Taming Heterogeneity—The Ptolemy Approach," *Proceedings of the IEEE, Vol. 91, No. 1*, pp. 127-144, 2003.
12. P. Derler, E. A. Lee, and A. Sangiovanni-Vincentelli, "Modeling Cyber-Physical Systems," *Proceedings of the IEEE (special issue on CPS)*, vol. 100, no. 1, pp. 13-28, January 2012. [Online]. https://doi.org/10.1109/JPROC.2011.2160929
13. E.A. Lee and A.L. Sangiovanni-Vincentelli, "Component-based design for the future," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, March 2011, pp. 1-5.
14. C. Ptolemaeus, *System Design, Modeling, and Simulation using Ptolemy II*.: Ptolemy.org, 2014. [Online]. http://ptolemy.org/books/Systems
15. Vector Informatik GmbH, "PREEvision Version 7.5 Manual," 2016.
16. S. Halbach et al., "Model Architecture, Methods, and Interfaces for Efficient Math-Based Design and Simulation of Automotive Control Systems," in *SAE Technical Paper*, Apr. 2010. [Online]. http://dx.doi.org/10.4271/2010-01-0241
17. G. Weiss, M. Zeller, D. Eilers, and R. Knorr, "Approach for Iterative Validation of Automotive Embedded Systems," in

Models 2010 ACES-MB Workshop Proceedings, 2010, pp. 69–83.

18. R. Marinescu 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. [Online]. http://dx.doi.org/10.1007/978-3-319-17581-2_13
19. R. Weissnegger et al., "Simulation-based Verification of Automotive Safety-critical Systems Based on EAST-ADL," *Procedia Computer Science*, vol. 83, pp. 245-252, 2016. [Online]. http://dx.doi.org/10.1016/j.procs.2016.04.122
20. IEEE, "IEEE Standard for Standard SystemC Language Reference Manual," *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005*, pp. 1-638, Jan. 2012. [Online]. http://standards.ieee.org/getieee/1666/download/1666-2011.pdf
21. IEEE, "IEEE Standard for Standard SystemC-Analog/Mixed-Signal Extensions Language Reference Manual," *IEEE Std 1666.1-2016*, 2016. [Online]. http://standards.ieee.org/getieee/1666_1/download/1666_1-2016.pdf
22. J. Wan, A. Canedo, and M. A.A. Faruque, "Functional Model-Based Design Methodology for Automotive Cyber-Physical Systems," *IEEE Systems Journal*, vol. PP, no. 99, pp. 1-12, 2015.
23. LMS Imagine.Lab. AMESim 2016. [Online]. http://www.plm.automation.siemens.com/en_us/products/lms/imagine-lab/amesim
24. D. Gebauer, J. Matheis, M. Kühl, and K. D. Müller-Glaser, "Integrierter, graphisch notierter Ansatz zur Bewertung von Elektrik/Elektronik- Architekturen im Fahrzeug," in *Moderne Elektronik im Kraftfahrzeug, IV*.: HDT (Haus der Technik), 2009, ch. 2.1, pp. 49-62.
25. E. A. Lee, S. Neuendorffer, and M. J. Wirthlin, "Actor-Oriented Design Of Embedded Hardware And Software Systems," *Journal of Circuits, Systems, and Computers*, vol. 12, pp. 231-260, 2003.
26. E. A. Lee and S. Tripakis, "Modal Models in Ptolemy," in *Proceedings of 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT 2010)*, October 2010, pp. 11-22.
27. X. Liu, Y. Xiong, and E. A. Lee, "The Ptolemy II Framework for Visual Languages," in *Proceedings of the IEEE 2001 Symposia on Human Centric Computing Languages and Environments (HCC'01)*, Washington, DC, USA, 2001, p. 50.
28. A. Davare et al., "A Next-Generation Design Framework for Platform-based Design," in *DVCon 2007*, February 2007. [Online]. http://chess.eecs.berkeley.edu/pubs/228.html
29. M. Treiber, A. Hennecke, and D. Helbing, "Congested Traffic States in Empirical Observations and Microscopic Simulations," *Physical Rev. E*, vol. 62, no. 2, pp. 1805-1824, 2000. [Online]. https://doi.org/10.1103/PhysRevE.62.1805
30. C. Roth et al., "A Simulation Tool Chain for Investigating Future V2X-based Automotive E/E-Architectures," in *7th European Congress on Embedded Real-Time Software and Systems (ERTS²)*, Toulouse, France, 2014, pp. 1739-1748.
31. H. Bucher, A. Klimm, O. Sander, and J. Becker, "Power Estimation of an ECDSA Core Applied in V2X Scenarios Using Heterogeneous Distributed Simulation," in *2015 IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, Chengdu, China, October 2015, pp. 187-194. [Online]. https://doi.org/10.1109/DS-RT.2015.35
32. H. Bucher et al., "A V2X Message Evaluation Methodology and Cross-Domain Modelling of Safety Applications in V2X-enabled E/E-Architectures," *EAI Endorsed Transactions on Security and Safety*, vol. 16, no. 8, 2016. [Online]. http://dx.doi.org/10.4108/eai.24-8-2015.2261038

Contact Information

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

Abbreviations

ACC	Adaptive Cruise Control
ADL	Architecture Description Language
BB	Building Block
BFS	Breadth First Search
BLA	Behavioral Logical Architecture
CAN	Controller Area Network
CT	Continuous Time
DE	Discrete Event
EAST-ADL	Electronics Architecture and Software Technology-ADL
ECU	Electronic Control Unit
EEA	Electric/Electronic Architecture
EEA-ADL	Electric/Electronic Architecture - Analysis Design Language
FRB	Fuse Relay Box
FSM	Finite State Machine
GRE	Gateway Routing Entry
GUI	Graphical User Interface
GW	Gateway
IDM	Intelligent Driver Model
LA	Logical Architecture
LF	Logical Function
MoC	Model of Computation
MoML	Modelling Mark-up Language
PtII	Ptolemy II
ST	Signal Transmission
XML	eXtensible Mark-up Language