Analyzing Industrial Architectural Models by Simulation and Model-Checking

Raluca Marinescu[1], Henrik Kaijser[2], Marius Mikučionis[3], Cristina Seceleanu[1], Henrik Lönn[2], and Alexandre David[3]

[1] Mälardalen University, Västerås, Sweden
raluca.marinescu@mdh.se, cristina.seceleanu@mdh.se
[2] Volvo Group Trucks Technology, Göteborg, Sweden
henrik.kaijser@volvo.com, henrik.lonn@volvo.com
[3] Aalborg University, Aalborg, Denmark
marius@cs.aau.dk, adavid@cs.aau.dk

Abstract. The software architecture of any automotive system has to be decided well in advance of production, so it is very desirable to assess its quality in order to obtain quick indications of errors at early design phases. In this paper, we present a constellation of analysis techniques for architectural models described in EAST-ADL. The methods are complementary in terms of covering EAST-ADL model analysis against a rich set of requirements, and in terms of the varying degree of confidence in the provided guarantees. Based on the needs of the current model-driven development in a chosen automotive context, we propose three analysis techniques of EAST-ADL architectural models, in an attempt to tackle some of the exposed design needs: simulation of EAST-ADL functions in Simulink, model-checking EAST-ADL models with timed automata semantics, and statistical model-checking in UPPAAL, applied on an automatically generated network of timed automata. An industrial Brake-by-Wire prototype is the case study on which we show the potential of simulating EAST-ADL models in Simulink, model-checking downscale EAST-ADL models, as well statistical model-checking of full model versions, in order to tame verification scalability problems.

1 Introduction

Mechanical and hydraulic systems in current vehicles are being replaced by electrical/electronic systems that can implement highly complex functions like cruise control and automatic braking. In order to deal with this complexity, the automotive industry has moved towards a model-based development process, during which high-level system models are designed and analyzed against requirements. Since many automotive systems are safety-critical, new standards such as ISO26262 place requirements on the quality of software. Consequently, companies that wish to adopt such standards will need to use methods and tools fit for guaranteeing such quality on each level of design abstraction.

Simulink [2], a model-based tool for design, simulation, and code generation of embedded systems, is already a well-established practice in the automotive domain. Simulink is typically used to define and assess system behavior in an early phase, or to create a detailed behavioral behavioral definition of the system in order to automatically generate the corresponding code. Architectural description languages, on the other hand, can be introduced earlier in the development, to provide models that could handle the complex software architecture of automotive systems. Compared to the current state-of-practice, architectural models offer a well-defined and standardized structure that deals with all the related information (e.g. functions, timing, triggering) of safety-critical systems [8]. A candidate for this task is East-adl [7], an architectural description language dedicated to the modeling and development of automotive embedded systems. The use of such modeling notations enables the application of verification techniques early in the industrial development process, in an attempt to gain early-phase indications of possible functional and timing errors.

In this paper, we propose a constellation of complementary verification techniques that can be applied on East-adl models to deliver various types of model correctness assurance. We start by briefly presenting the East-adl architectural language and the tools involved in the verification process (see Section 2), and we discuss the current state-of-practice in the development of automotive systems as used nowadays by the automotive industry (see Section 3). Next, we present our simulation and model-checking methodology (see Section 4), and we show the verification techniques based on the: (i) simulation of East-adl models from a set of predefined verification cases with Simulink (see Section 6), (ii) symbolic simulation and formal verification of East-adl with Uppaal, and (iii) statistical model-checking of the architectural model with Uppaal SMC (see Section 8). In order to enable the verification of architectural models in East-adl, we also contribute with a timed automata (TA) semantics that we propose for the East-adl components (see Section 7). We show how the formal techniques underlying the tools complement each other, by applying the East-adl to Simulink, and East-adl to Uppaal-TA transformations to analyze the Brake-by-Wire (BBW) industrial system (see Section 5). Such an endeavor exposes also the advantages and limitations of each framework, when used on an industrial system model, which can serve as a guiding result especially if safety standards such as ISO26262 are to be adopted. We end this paper by discussing similar related works (see Section 9), and by presenting our conclusions (see Section 10). The actual contribution of this paper consists of introducing two new transformations, one from East-adl models to Simulink models, and one from East-adl models to Uppaal models, together with the application of simulation, model-checking and statistical model-checking on an industrial architectural model.

2 Brief Overview of the EAST-ADL Language

East-adl [7] is an AUTOSAR [4] compatible architectural description language for automotive electronic systems. The functionality of the system is defined at four levels of abstraction, as follows. The *Vehicle Level* is the highest level of abstraction and describes the electronic features as they are perceived externally. Next, the *Analysis Level* allows an abstract functional representation of the architecture without prescribing a specific hardware topology. The *Design Level* presents a detailed functional representation of the architecture, plus the allocation of these elements on to the hardware platform. Last, the *Implementation Level* describes the implementation of the system using AUTOSAR elements. At each abstraction level, the system model relies on the definition of a set of *FunctionTypes* representing components that describe the functional structure of the system. Each of these *FunctionTypes* has: (i) a set of *FlowPorts* that provide and receive data, (ii) a *FunctionTrigger* that can be either time-based or event-based, and (iii) a *FunctionBehavior*. The system is modeled as a set of interconnected *FunctionPrototypes*, where each *FunctionPrototype* is an instantiation of the corresponding *FunctionType*. The execution of each *FunctionPrototype* is based on the "read-execute-write" semantics, which enables semantically sound analysis and behavioral composition, and makes the function execution independent of the notation used, when defining its internal behavior. The *FunctionBehavior* is defined using different notations and tools, e.g., Simulink or UPPAAL PORT timed automata (TA) [13]. At each level of abstraction, the above structural elements of the system can be extended with annotations for orthogonal aspects like requirements, timing properties, generic constraints. etc. EAST-ADL also provides means to describe different validation and verification activities as *VVCases* for different levels of abstraction.

In the following section, we present a typical automotive development process and we try to identify different needs and gaps that need to be addressed.

3 The Current Development Process in an Automotive Context

We have identified four main groups of actors who are involved in a typical automotive development process: the *Client*, the *System Engineers*, the *Software Developers*, and the *Verification Engineers*.

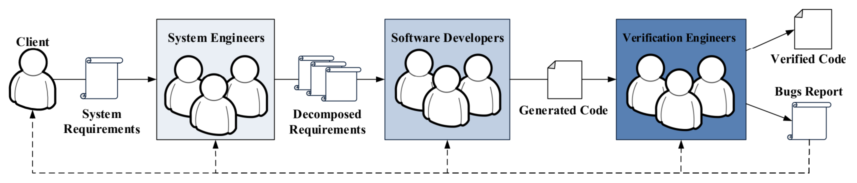


Fig. 1: A typical automotive development process.

As depicted in Figure 1, the *Client* compiles a set of informal, natural language requirements describing the new system that needs to be implemented. The *System Engineers* break down these requirements in incremental steps, passing the current requirement set from one engineer to the other for further decomposition. The *Software Developers* decompose further these requirements while considering implementation elements like the system architecture. This new set of requirements, consisting of one requirement document per system component, is divided among the *Software Developers*, who create a model-based implementation of the components in the system. The components may be modeled using the Simulink tool, and the code is automatically generated based on these models. This code is integrated as the behavior of an AUTOSAR software component and, where necessary, adjusted by the *Software Developers*. In order to ensure correct behavior, model-in-the-loop and software-in-the-loop analysis are used. Once a software component has been implemented, it can be deployed on an electronic control unit (ECU) for component testing. Finally, the *Verification Engineers* perform testing at the system level directly on the platform, using manually written tests. Any bugs discovered in the implementation or any problems in the requirements are reported back to the person responsible for the implementation or requirement, respectively. Since models start to be included in the industrial development process, there is also an increased need of stronger evidence of model correctness with respect to functional or timing requirements.

For the development process described above, different state-of-the-art techniques could facilitate model integration and verification, as follows:

– Introducing architectural languages (like East-adl) will keep track of requirements, features, functions, and hardware topology in an integrated model, making the design decisions consistent and traceable.
– Providing the behavior for architectural components based on formal definitions like TA, together with typical Simulink definitions, will enable alternative representations of the same function, hence providing a more comprehensive assessment of the system.
– Applying formal verification techniques, like model-checking, on the system's formalized structural and behavioral model will provide correctness assurances regarding important properties.

In order to adopt these steps, an integrated system model is needed, such that different verification techniques can be applied consistently, on the same system description, at various levels of abstraction.

4 Our Methodology for Analyzing Architectural Models

In this section, we propose a methodology for simulation and model-checking of East-adl models, which is depicted in Figure 2. Our verification methodology consists of the following steps:

– Create the East-adl model and provide the behavior of each *FunctionPrototype* as a FMU[1] [3] or a Simulink model;

[1] The Functional Mock-up Interface (FMI) is a tool-independent standard to support behavior models using a combination of xml-files and compiled C-code.The standard defines the concept of a Functional Mock-up Unit (FMU), as a software component that implements the FMI standard.

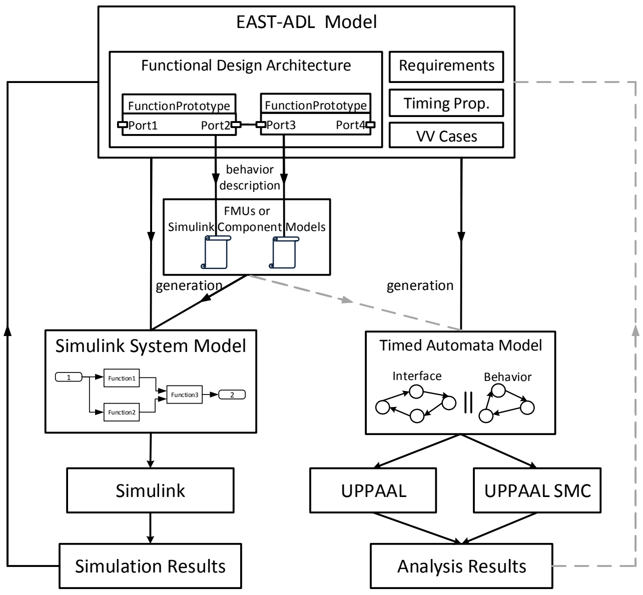


Fig. 2: Our simulation and model-checking methodology.

– Select the verification method:
  1. Simulation: by implementing an automatic transformation from the architectural model to a Simulink model and calling the Simulink tool, we can provide verification through simulation;
  2. Model-checking: by implementing an automatic transformation from the architectural model to a network of TA, we can use the Uppaal or Uppaal SMC model-checker to formally verify the system;
– Return the verification results back to the East-adl model for possible improvements of the design.

There are several differences between the two frameworks. The simulation method requires the East-adl model to be extended with verification and validation elements as *VVCases*, which describe the part of the model to be analyzed, together with the definition of monitor *FunctionTypes*, stimuli data, and the requirements to be verified. The behavioral model of the monitor is provided as an FMU or a Simulink model. The transformation to the network of TA provides formal semantics for the architectural model in terms of timed transition systems [5]. In order to preserve the informal semantics of the architectural language, the transformation produces a network of two synchronized TA for each East-adl *FunctionPrototype*: an $Interface$ TA with the elements provided in the architectural model and a *Behavior* TA.

The parts represented with a dotted line in Figure 2 have not been implemented in the current version of the transformation. By extending our methodology to include an automatic transformation from the Simulink component model to the corresponding *Behavior* TA, the two models would be consistent and the verification results of the both frameworks would truly complement each other. However, information would be lost in such a transformation and the TA model would require manual refinements, such that the TA could represent the key behavior of the component that is largely consistent with the corresponding Simulink model.

5 An Example from Industry: Brake-by-Wire Case Study

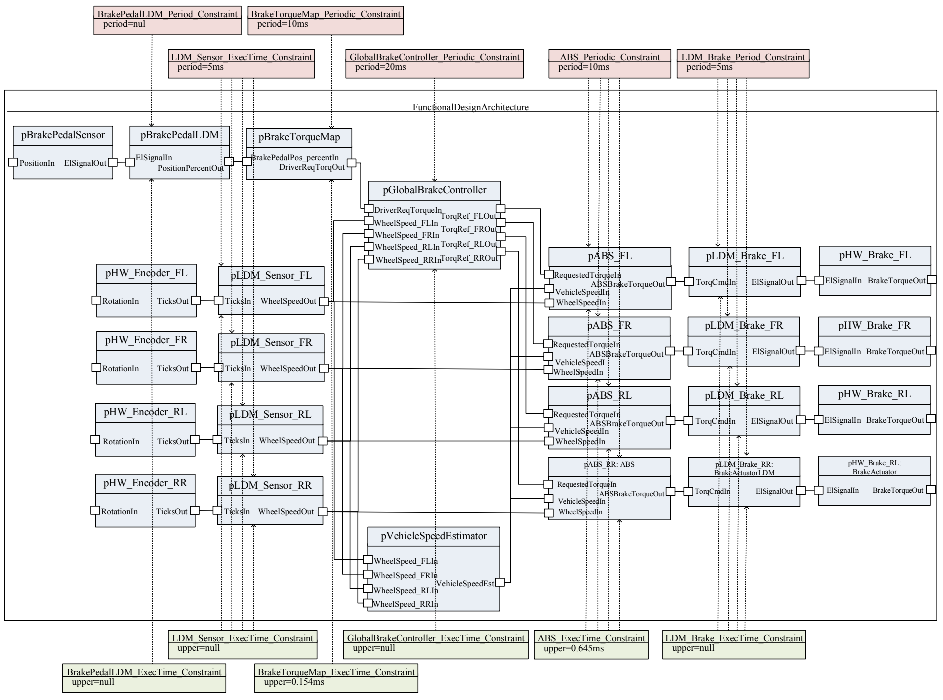


Fig. 3: The EAST-ADL model of the BBW system at Design Level.

In this section, we introduce the Brake-by-Wire (BBW) system that will be used through the paper as the running example to illustrate our techniques. The BBW system is a braking system equipped with an ABS function, and without any mechanical connectors between the brake pedal and the brake actuators. A sensor attached to the brake pedal reads its position, which is used to compute the desired global brake torque. For vehicles with stability control, the torque is influenced by the wheel speed and the desired torque for each wheel is calculated based on the following equation:

$$torque = (pos/100) \times maxBrakeTorque \times distribution \tag{1}$$

where *pos* is the pedal position with values $\in$ [0,100], *maxBrakeTorque* is the maximum global brake torque, and *distribution* is the static distribution factor. The ABS algorithm computes the slip rate s based on the following equation:

$$s = (v - w \times R)/v \tag{2}$$

where v is the speed of the vehicle, w is the speed of the wheel, and R is the radius of the wheel. The friction coefficient has a nonlinear relationship with the slip rate: when s starts increasing, the friction coefficient also increases, and its value reaches the peak when s is around 0.2. After that, further increase in s reduces the friction coefficient of the wheel. For this reason, if s is greater than 0.2 the brake actuator is released and no brake is applied, otherwise the requested brake torque is used.

Figure 3 presents the EAST-ADL model of the BBW system at the Design Level, and a set of requirements has been provided (to describe the functionality of this system at this level), as follows:

$\mathbf{D}_1$ The torque on the wheel shall be defined as: (pos/100) × maxBrakeTorque × distribution.
$\mathbf{D}_2$ If VehicleSpeedIn>ABSVehicleSpeedThreshhold **and** s>ABSSlipRateThreshhold, then ABSBrakeTorqueOut shall be set to 0Nm.
$\mathbf{D}_3$ If s<=ABSSlipRateThreshhold **or** VehicleSpeedIn<=ABSVehicleSpeedThreshhold, then ABSBrakeTorqueOut shall be set to RequestedTorqueIn.
$\mathbf{D}_4$ Investigate the latency between the wheel sensor and the brake pedal actuator.

The goal of this work is to show how one can verify the above requirements on the EAST-ADL description, using various verification techniques that we present in the following.

6 Simulation of EAST-ADL Functional Architecture in Simulink

In this section we describe the simulation method proposed in Section 4, which has been implemented as an EATOP [1] plug-in called FMUSim that synthesizes a Simulink model and configures it according to the properties in the EAST-ADL model. The model transformation preserves the compositional hierarchy of the EAST-ADL model in EATOP, and is implemented as a one-to-one mapping between EAST-ADL elements and Simulink elements, as depicted in Table 1.

In order to simulate a time-trigged EAST-ADL function, the FMU block needs to be sampled once per period. However, the FMU blocks provided by the FMI Toolbox are continuous and cannot be sampled directly. As depicted in Figure 4, the solution chosen in this implementation is to add a pulse generator and a subsystem InputData that is acting as a flip-flop clocked on the positive flank of the pulse. Since the execution of a Simulink block is instantaneous, another flip-flop OutputData is added, which is clocked on the negative flank of the pulse, such that the execution time of the FMU becomes equal to the pulse width. Similarly, in order to simulate an event-trigged EAST-ADL function, we reuse the negative flank of the trigger pulse from another time-triggered function that acts as the event source. The negative flank of EventTriggerIn is used to clock a flip-flop InputData to control execution start, as depicted in Figure 5. The execution period of the function is then simulated by adding a flip-flop OutputData, which is clocked on a step down that is generated at a time equal to the worst-case

Table 1: Mapping rules for the EAST-ADL to Simulink transformation.

EAST-ADL element	**Simulink element(s)**
composed FunctionType	Subsystem
Function Connector	Line
non-top-level Function Flow Port In	Inport
non-top-level Function Flow Port Out	Outport
top-level Function Flow Port In	Repeating Sequence Interpolated
top-level Function Flow Port Out	Scope
time-trigged leaf Function Type with FMU behavior	Pattern with several elements
event-trigged leaf Function Type with FMU behavior	Pattern with several elements
Continuous leaf Function Type with FMU behavior	FMU Block
leaf Function Type with Simulink behavior	Same pattern as in the FMU cases above, but a copy of the behavior model is inserted instead of the FMU block

execution time (WCET) after the function starts executing. The clock signal is exported as EventTriggerOut for the pattern to be repeatable. This means that it is possible to simulate a chain of event-trigged functions with the pattern.

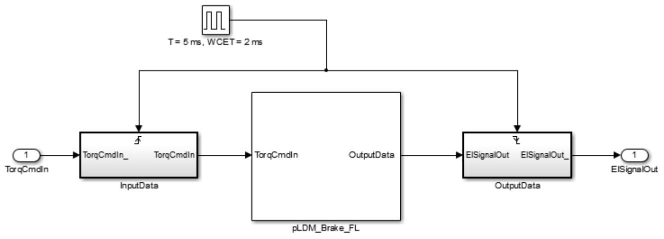


Fig. 4: Simulink pattern for modeling time-trigged execution of an EAST-ADL function with execution time. The block pLDM_Brake_FL represents the FMU.

In this transformation, we have not addressed the nondeterminism or the possible interleavings of the *FunctionPrototypes*'s execution. Since we are performing simulations on the transformed model, the current execution pattern is one of infinitely many interleavings and event sequences, which means that some errors may be overlooked. To represent deviating clock speeds and arbitrary start-up time, an arbitrary component could be added by the transformation to the offset and period times, and a deterministic yet random sequence would secure repeatability of the simulation runs. Multiple runs with randomized parametrization would increase confidence through the extended state space

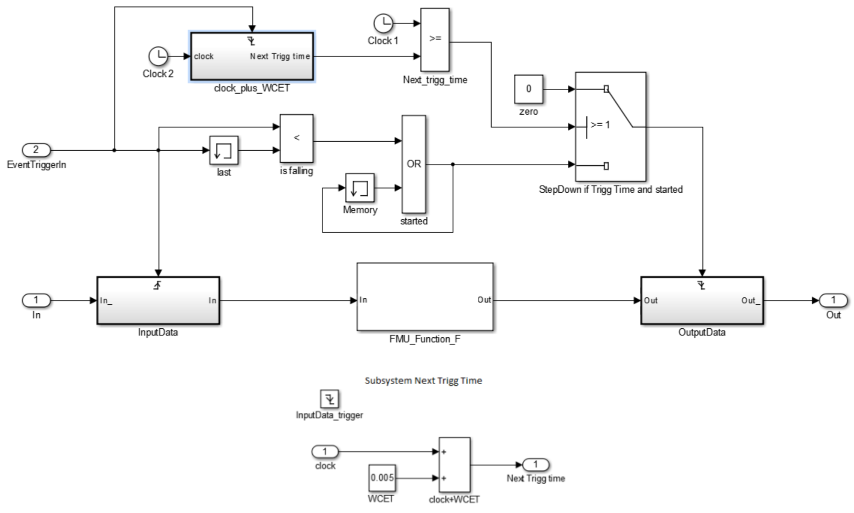


Fig. 5: Simulink pattern for modeling event-trigged execution of an EAST-ADL function with execution time. The block FMU_function_F represents the FMU.

covered. However, these extensions to the method are not in the scope of this paper.

Application on the BBW case study. We have applied the transformation described above on the BBW case study. The resulting model contains one FMU for each leaf EAST-ADL *FunctionPrototype*, plus the required monitors for the *VVCase* specified in the EAST-ADL model.

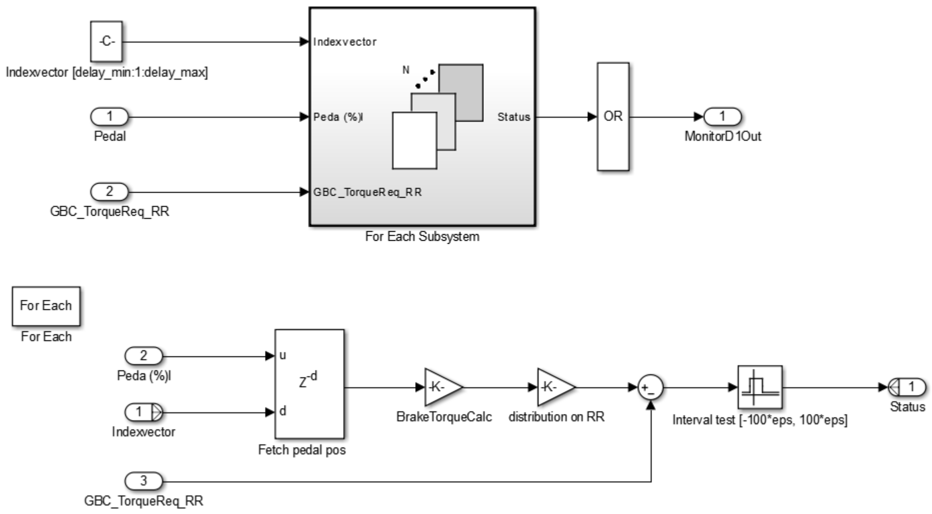


Fig. 6: Implementation of the pBrakeTorqueRRMonitor. The lower half of the figure shows the contents of the block named for each subsystem in the upper half.

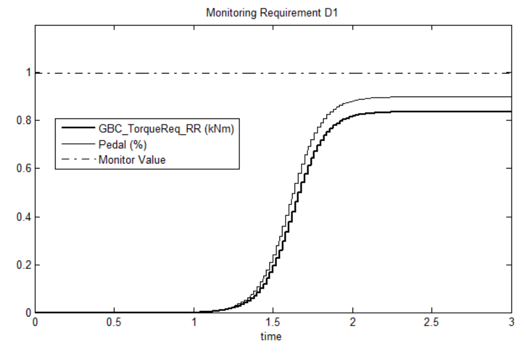


Fig. 7: Simulation results provided by the pBrakeTorqueRRMonitor.

As depicted in Figure 6, pBrakeTorqueRRMonitor is a complex monitor despite the fact that it verifies a simple linear function like requirement D_1 for the rear right wheel. The time until a new pedal position has propagated through the system and has given rise to a new torque value GBC_TorqueReq_RR varies between delay_min and delay_max [ms]. As shown in Figure 7, the torque requested by the brake controller on the rear right wheel is a linear scaling of the pedal position delayed by the propagation time. The boolean monitor function "looks back" in time according to the delay interval, and is able to find a pedal position corresponding to the requested torque at all evaluated time points. The result shows that requirement D_1 is satisfied to the extent guaranteed by the simulation technique.

7 Formal Semantics of EAST-ADL as a network of Timed Automata

To formally verify that the architectural model meets its requirements, we need to exhaustively explore all the function blocks in the model. In this context, we need to represent the execution semantics of the EAST-ADL function blocks using a network of TA (see Figure 2), which has a well-defined formal semantics in terms of timed transition systems [5]. We have developed an automatic transformation, considering a subset of the EAST-ADL elements, which we define as a tuple:

$$EAST-ADL_{DesignLevel} \triangleq \langle F_P, Con, DP, Trigg, TC \rangle,$$

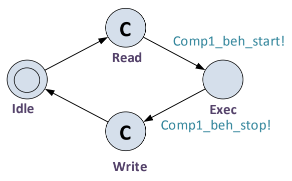
where F_P is the set of $FunctionPrototypes$, Con is the set of connectors between the F_P, DP is the set of data ports, defined as the union of input ports and output ports, $Trigg$ is the set of triggering elements, defined as the union of events and periodic triggers, and TC the set of the model's timing constraints. In a similar manner, the TA is defined as a tuple:

$$TA \triangleq \langle L, l_0, C, A, E, I \rangle,$$

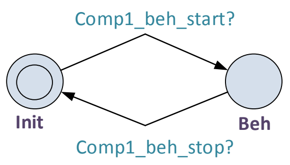
where L is a finite set of locations, $l_0 \in L$ is the initial location, C is a set of clocks, A is a set of possible actions, E is a set of edges between two locations, and I is a set of invariants attached to the locations.

The transformation is a one-to-one function π : EAST-ADL$_{DesignLevel} \rightarrow$ TA, which maps each element in the EAST-ADL$_{DesignLevel}$ to a TA element. The mapping rules are:

– Each function F_P is defined in terms of a network of two TA, as shown in Figure 8. To preserve the "read-execute-write" semantics of East-adl, the $Interface$ TA (see Figure 8a) has four locations: (i) $Idle$, (ii) a *Read* location that allows the update of the variables according to the values on the input ports, independent of other computations, (iii) an *Exec* location that triggers the *Behavior* TA (see Figure 8b) that models the desired behavior of F_P, and (iv) a $Write$ location that allows the update of the output ports according to the values of the computed internal variables, respectively, independent of other computations.
– Each input and output port DP is mapped to a global variable in the TA network, respectively.
– Each connector Con from output port $Port_{out1}$ of F_{P1} to input port $Port_{in2}$ of F_{P2} is transformed into an assignment $Port_{in2} := Port_{out1}$, along the edge from $Idle$ to *Read*;
– The triggering of each interface TA is based on the triggering $Trigg$ associated to the East-adl F_P. Concretely, this creates two possible instantiations of the $Interface$ TA: (i) for timed-triggered F_P the transformation produces a local clock, plus invariants and guards on TA (see Figure 9a), and (ii) for event-triggered F_P the transformation produces a set of dedicated variables that need to be constantly updated and reset, respectively (see Figure 10a).
– Other timing annotations TC, e.g., the execution time, can be included in the timing behavior of the TA model.



(a) Interface template.



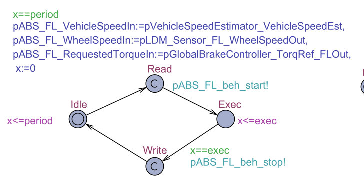
(b) Behavior template.

Fig. 8: The generic TA semantics of an East-adl F_P.

Once we obtain the network of TA corresponding to the East-adl model, one manually edits the *Behavior* TA to match the desired behavior of the corresponding *FunctionPrototype*. Formal analysis techniques like model-checking and statistical model-checking are then applied to verify the resulting model. In the next section we apply such transformation on the BBW East-adl model, to enable the latter's verification.

8 Analysis of EAST-ADL Models Using Model-Checking and Statistical Model Checking

We have applied our method on the BBW architecture, and generated a network of 50 TA, by transforming each of the 25 F_P of Figure 3 into a network of two synchronized TA, respectively. In Figures 9 and 10, we exemplify the transformation of two F_P as follows: Figure 9a presents the interface of the time-triggered pABS_FL F_P, automatically generated from the EAST-ADL model, Figure 9b presents the behavior of the pABS_FL F_P obtained after manually editing the dedicated TA template (see Figure 8b); Figure 10a shows the interface of the event-triggered pVehicleSpeedEstimator F_P, whereas Figure 10b shows the behavior of the pVehicleSpeedEstimator F_P, after manually editing the dedicated TA template.


x==period
pABS_FL_VehicleSpeedIn:=pVehicleSpeedEstimator_VehicleSpeedEst,
pABS_FL_WheelSpeedIn:=pLDM_Sensor_FL_WheelSpeedOut,
pABS_FL_RequestedTorqueIn:=pGlobalBrakeController_TorqRef_FLOut,
x:=0
Read
pABS_FL_beh_start!
Idle
Exec
x<=period
x<=exec
Write
x==exec
pABS_FL_beh_stop!
(a) Interface TA

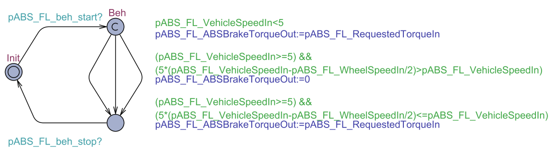
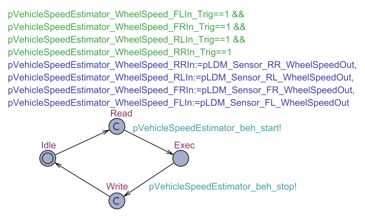

pABS_FL_beh_start?
Beh
Init
pABS_FL_VehicleSpeedIn<5
pABS_FL_ABSBrakeTorqueOut:=pABS_FL_RequestedTorqueIn
(pABS_FL_VehicleSpeedIn>=5) &&
(5*(pABS_FL_VehicleSpeedIn-pABS_FL_WheelSpeedIn/2)>pABS_FL_VehicleSpeedIn)
pABS_FL_ABSBrakeTorqueOut:=0
(pABS_FL_VehicleSpeedIn>=5) &&
(5*(pABS_FL_VehicleSpeedIn-pABS_FL_WheelSpeedIn/2)<=pABS_FL_VehicleSpeedIn)
pABS_FL_ABSBrakeTorqueOut:=pABS_FL_RequestedTorqueIn
pABS_FL_beh_stop?
(b) Behavior TA

Fig. 9: The TA model for the pABS_FL EAST-ADL F_P.


pVehicleSpeedEstimator_WheelSpeed_FLIn_Trig==1 &&
pVehicleSpeedEstimator_WheelSpeed_FRIn_Trig==1 &&
pVehicleSpeedEstimator_WheelSpeed_RLIn_Trig==1 &&
pVehicleSpeedEstimator_WheelSpeed_RRIn_Trig==1
pVehicleSpeedEstimator_WheelSpeed_RRIn:=pLDM_Sensor_RR_WheelSpeedOut,
pVehicleSpeedEstimator_WheelSpeed_RLIn:=pLDM_Sensor_RL_WheelSpeedOut,
pVehicleSpeedEstimator_WheelSpeed_FRIn:=pLDM_Sensor_FR_WheelSpeedOut,
pVehicleSpeedEstimator_WheelSpeed_FLIn:=pLDM_Sensor_FL_WheelSpeedOut
Read
pVehicleSpeedEstimator_beh_start!
Idle
Exec
Write
pVehicleSpeedEstimator_beh_stop!
(a) Interface TA

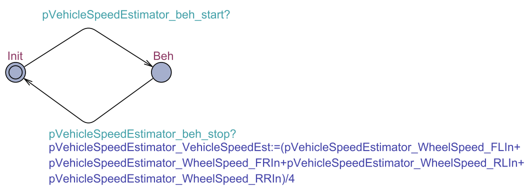

pVehicleSpeedEstimator_beh_start?
Init
Beh
pVehicleSpeedEstimator_beh_stop?
pVehicleSpeedEstimator_VehicleSpeedEst:=(pVehicleSpeedEstimator_WheelSpeed_FLIn+
pVehicleSpeedEstimator_WheelSpeed_FRIn+pVehicleSpeedEstimator_WheelSpeed_RLIn+
pVehicleSpeedEstimator_WheelSpeed_RRIn)/4
(b) Behavior TA

Fig. 10: The TA model for the pVehicleSpeedEstimator EAST-ADL F_P.

On this formal model, we have applied model-checking and statistical model-checking techniques to validate the original EAST-ADL model against the requirements introduced in Section 5.

Model-Checking with UPPAAL. With UPPAAL, we have simulated and we have attempted to verify the previously described network of TA. However, the size of the model has lead to a state space explosion. On a computer with 1.8 Ghz Intel processor and 8GB memory, the verifier could explore only 10 962 377 states before it had run out of memory. This is not surprising, since the BBW system is subject to an enormous state-space explosion due to large number to TA in the network, each with its clock and its set of variable created based o the ports of the corresponding *FunctionPrototype*.

Consequently, we have used Uppaal to verify a simplified version of the BBW system with one wheel only. Properties $\mathbf{D}_2$ and $\mathbf{D}_3$ are formalized as TCTL properties [5], as follows:

$\mathbf{D}_2$ A[] pABS_FL_VehicleSpeedIn>speed_thrshld **and** pABS_FL_s==true **imply** pABS_FL_ABSBrakeTorqueOut==0.

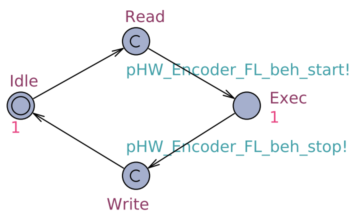
$\mathbf{D}_3$ A[] pABS_FL_VehicleSpeedIn<=speed_thrshld **or** pABS_FL_s==false **imply** pABS_FL_ABSBrakeTorqueOut==pABS_FL_RequestedTorqueIn

Both properties have been verified and hold on the model. For property $\mathbf{D}_2$ the verification took 13,7 seconds and used 26 900KB of memory. For property $\mathbf{D}_3$ the verification took 9,1 seconds and used 26 916KB of memory.

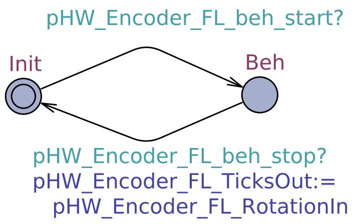
Statistical model-checking with UPPAAL SMC. TA is a suitable formalism for analyzing architectural models like East-adl, and enables symbolic model-checking techniques to provide a rigorous proof of verifying or refuting a TCTL property. However, such techniques suffer from state-space explosion in terms of number of parallel components in the model, which is the case with complex, industrial systems. One possible solution is the use of a statistical model-checking engine to generate stochastic simulations and employ statistical methods to estimate probabilities and probability distributions over time with given confidence levels. The Uppaal modeling language has been extended with probabilistic and dynamical constructs, given a stochastic semantics of timed automata networks [9], and the tool has been equipped with statistical model-checking (SMC) algorithms [10] to decide qualitative properties in terms of probabilities and cost. The symbolic and statistical techniques complement each other: SMC can show results only up to a specified level of confidence and never for certain like symbolic techniques, but it is a cheap way to generate and confirm safety counter-examples where symbolic techniques may employ expensive over-approximation [11]. Here, we attempt to analyze requirement $\mathbf{D}_4$.

Since Uppaal SMC works on stochastic models, we have manually added probabilistic extensions to the four-wheels BBW model that contains the timed behavior. Figures 11a and 11b show exponential rates added to locations Idle and Exec of one Encoder component of Figure 3. The rate of 1 means that the component may potentially stay in the location forever, but it will stay there for 1 time unit on average which is consistent with the timed behavior. Further, we are interested in latency between pressing the pedal and applying the brakes, hence we added a monitoring stop-watch automaton shown in Figure 11c. The monitoring automaton has a stop-watch L that is stopped originally in location Wait by specifying that the derivative is zero: L'==0. The stop-watch is started when synchronization pBrakePedalSensor_beh_start? is received (the derivative L'==1 is implicit in timed automata). The stop-watch is stopped again when any of the wheels receive braking signal by synchronization pHW_Brake_FL_beh_start?, pHW_Brake_FR_beh_start?, pHW_Brake_RL_beh_start? or pHW_Brake_RR_beh_start? (the synchronizations are then on different edges that are drawn on top of each other to minimize cluttering). The latency can be estimated by the following query: Pr[bm.L<=1000](<>bm.Done) that asks what is the probability that the brake monitor process bm will end up in location Done in terms of the stop-watch L value.

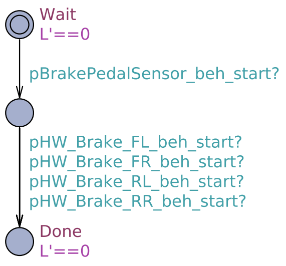
The result is shown in Figure 11d. The average latency is 5 time units but it tends to be high even though our added stochastic delay assumptions are decreasing towards infinity, which is a worrying behavior. The good news is that it seems to be strictly limited by 6 time units and no simulation has been observed greater or equal than 6 time units, which is on the other hand surprising, as the model contains components with unlimited delays.



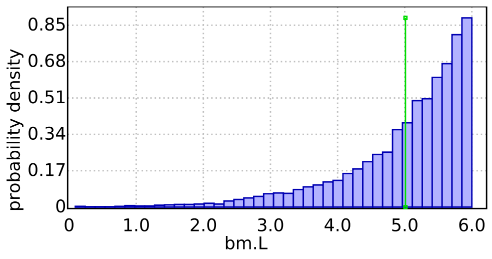
(a) pHW_Encoder_FL_CTRL component.


pHW_Encoder_FL_beh_start?
Init
Beh
pHW_Encoder_FL_beh_stop?
pHW_Encoder_FL_TicksOut:=
pHW_Encoder_FL_RotationIn

(b) pHW_Encoder_FL_BEH component.



(c) Stop-watch automaton bm measuring latency.



(d) Estimated latency of 5.01 ± 0.05 time units (99.9% CI).

Fig. 11: The components decorated with stochastic extensions and estimated latency between pressing the pedal and applying brakes.

9 Related Work

Several researchers have looked into the formal analysis and verification of East-ADL models. Kang et al. [13] propose a component-based analysis framework for the East-ADL models extended with TA semantics based on the Uppaal Port model-checker. Mallet et al. [14] describe the use of UML MARTE profile for the timing analysis of East-ADL. In addition, Feng et al. [12] propose a translation of East-ADL activity diagrams into the input language of SPIN for formal verification. More recently, Qureshi et al. [15] describe a model-to-model transformation from East-ADL to timed automata towards formal verification based on timing constraints using Uppaal. Closely related to our work, in the context of model-driven development, Biehl et al. [6] propose a modular approach for data integration, together with their experiences from applying this approach for the verification of East-ADL models. The latter is focused on introducing a systematic solution for model-based tool integration, whereas our work is focused on the analysis of industrial systems through complementary methodologies that provide various degrees of assurance.

10 Conclusions and Discussion

In this paper, we have presented a set of analysis techniques dedicated to the simulation and verification of automotive embedded systems specified in the EAST-ADL architectural language. In order to provide different correctness guarantees, we present three techniques that enable the transformation in, and analysis of EAST-ADL models with: (i) Simulink, a design and simulation tool used extensively in industry, (ii) UPPAAL for model-checking purposes, and (iii) UPPAAL SMC, a new extension of UPPAAL with statistical model-checking capabilities. We report our analysis results by applying all these frameworks on the industrial BBW case study. As future work, we intend to investigate the possible integration and application of these frameworks into the large-vehicle industrial development process.

Limitations. Our current transformation to Simulink does not support jittering of the execution start time and period times. The coverage of the state space in terms of different function execution orders and phasings is thus very low, but sufficient to detect the fundamental problems.

The model transformation from EAST-ADL to the network of TA and to the Simulink model rely on the execution semantics of EAST-ADL. However, the TA used to define *FunctionBehavior* is difficult to make fully consistent with the richer representation of the Simulink model or the FMU that is used by the FMUSim tool. The verifications are thus complementary, and will not in general verify the same properties.

Lessons Learned. Both transformations presented in the paper are conceptually simple, making them easy to implement and fast to execute. The two model transformations preserve the structure of the architecture, which simplifies the understanding and the debugging of the model. In our transformation to Simulink, it is possible to define useful transformation patterns for time and event triggered functions based on the FMI Toolbox and legacy Simulink blocks only, so additional commercial toolboxes are not required. The EAST-ADL models with feedback loops require that the loops are broken before they can be simulated in Simulink. This can be achieved either by adding a memory block somewhere in each loop or latching the subsystem ports of at least one subsystem in each loop. Moreover, the network of TA can be easily used for statistical model-checking with UPPAAL SMC, ensuring formal verification of the model even if the analysis with UPPAAL leads to a state-space explosion.

Acknowledgment: The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement number 269335, and from VINNOVA, the Swedish Governmental Agency for Innovation Systems, within the MBAT project.

References

1. Eclipse. The EAST-ADL Tool Platform (EATOP) Editor Tool. Available from http://www.eclipse.org/proposals/modeling.eatop/, (2014).
2. Matworks. The MATLAB Simulink Design Tool. Available from http://www.mathworks.se/products/simulink/, (2014).
3. Modelica Association Project. The Functional Mock-up Interface (FMI) Standard. Available from http://www.fmi-standard.org/, (2014).
4. The AUTomotive Open System ARchitecture (AUTOSAR). Available from http://www.autosar.org/, (2014).
5. Rajeev Alur. Timed Automata. In *Computer Aided Verification*, pages 8–22. Springer, (1999).
6. Matthias Biehl, Carl-Johan Sjöstedt, and Martin Törngren. A Modular Tool Integration Approach- Experiences From Two Case Studies. In *3rd Workshop on Model-Driven Tool & Process Integration at the European Conference on Modelling Foundations and Applications*, (2010).
7. Hans Blom, Henrik Lönn, Frank Hagl, Yiannis Papadopoulos, Mark-Oliver Reiser, Carl-Johan Sjöstedt, De-Jiu Chen, Fulvio Tagliabò, Sandra Torchiaro, and Sara Tucci. EAST-ADL: An Architecture Description Language for Automotive Software-Intensive Systems. *EAST-ADL WhitePaper*, Volume 1, (2013).
8. Philippe Cuenot, DeJiu Chen, Sebastien Gerard, Henrik Lonn, M-O Reiser, David Servat, C-J Sjostedt, Ramin Tavakoli Kolagari, Martin Torngren, and Matthias Weber. Managing Complexity of Automotive Electronics using the EAST-ADL. In *12th IEEE International Conference on Engineering Complex Computer Systems*, pages 353–358. IEEE, (2007).
9. Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, Danny Bøgsted Poulsen, Jonas van Vliet, and Zheng Wang. Statistical Model Checking for Networks of Priced Timed Automata. In *International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, pages 80–96, (2011).
10. Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, and Zheng Wang. Time for Statistical Model Checking of Real-Time Systems. In *Computer-Aided Verification*, pages 349–355, (2011).
11. Alexandre David, Kim Guldstrand Larsen, Axel Legay, and Marius Mikucionis. Schedulability of Herschel-Planck Revisited Using Statistical Model Checking. In *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, pages 293–307, (2012).
12. Lei Feng, DeJiu Chen, Henrik Lönn, and Martin Torngren. Verifying System Behaviors in EAST-ADL2 with the SPIN Model Checker. In *International Conference on Mechatronics and Automation*, pages 144 –149, (2010).
13. Eun-Young Kang, Eduard Paul Enoiu, Raluca Marinescu, Cristina Seceleanu, Pierre-Yves Schobbens, and Paul Pettersson. A Methodology for Formal Analysis and Verification of EAST-ADL Models. *Reliability Engineering & System Safety International Journal*, (2013).
14. Frédéric Mallet, Marie-Agnès Peraldi-Frati, and Charles André. Marte CCSL to Execute EAST-ADL Timing Requirements. In *International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 249 –253. IEEE, (2009).
15. Tahir Naseer Qureshi, De-Jiu Chen, Magnus Persson, and Martin Trngren. On Integrating EAST-ADL and UPPAAL for Embedded System Architecture Verification. In *Embedded Systems Development*, volume Volume 20, pages 85–99. Springer, (2014).