A Methodology for Formal Analysis and Verification of EAST-ADL models

Eun-Young Kang[a], Eduard Paul Enoiu[b], Raluca Marinescu[b], Cristina Seceleanu[b], Pierre-Yves Schobbens[a], Paul Pettersson[b]

[a] *PReCISE Research Centre, University of Namur, Belgium*
[b] *Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden*

Abstract

The architectural design of embedded software has a direct impact on the final implementation, with respect to performance and other quality attributes. Therefore, guaranteeing that an architectural model meets the specified requirements is beneficial for detecting software flaws early in the development process. In this paper, we present a formal modeling and verification methodology for safety-critical automotive products that are originally described in the domain-specific architectural language EAST-ADL. We propose a model-based approach that integrates the architectural models with component-aware model checking, and describe its tool support called ViTAL. The functional and timing behavior of each function block in the EAST-ADL model, as well as the interactions between function blocks are formally captured and expressed as Timed Automata models, which have precise semantics and can be formally verified with ViTAL. Furthermore, we show how our approach, supported by ViTAL, can be used to formally prove that the EAST-ADL system model fulfills the specified real-time requirements and behavioral constraints. We demonstrate that the approach improves the modeling and verification capability of EAST-ADL and identifies dependencies, as well as potential conflicts between different automotive functions before implementation. The method is substantiated by verifying an automotive braking system model, with respect to particular functional and timing requirements.

Keywords: EAST-ADL, UPPAAL PORT, Formal Analysis, Model-driven Development, Model transformation

1. Introduction

Automotive embedded systems exhibit the typical behaviors of complex systems that are subjected to timing and resource constraints. For example, a brake system should react timely to avoid catastrophic consequences. Thus, analysis of the automotive system's timing behavior at early stages of development is of utmost importance. Model-Driven Development (MDD) has been proven an effective way of both managing system complexity, as well as providing a solid basis for the systematic design of embedded systems, at various abstraction levels [1]. EAST-ADL [2, 3] is an architecture description language for modeling and development of automotive embedded systems, which focuses on functional specifications [4] with support for structural definition. The language encompasses the specification of requirements, system environment, software and hardware functions, behavior, timing constraints, and other related information [5], and provides an integrated modeling framework that uses concepts from MDD and component-based development [6]. However, the system behavior is only defined in terms of a set of elementary functional blocks and their triggers and interfaces, without formal support for the internal behavioral specification of such function blocks. Instead, the execution of each function is described using external behavioral annexes via

domain-specific tools, e.g., Simulink or UML [5]. This impedes the possibility to construct, transform, and verify EAST-ADL models using formal methods such as symbolic simulation and model-checking.

To alleviate the aforementioned restriction, we propose a methodology that facilitates the verification of system function behaviors in EAST-ADL by using the UPPAAL PORT model-checker [7] and its simulator. Our method is based on integrating architectural description languages with verification techniques and is tailored for EAST-ADL models. In order to provide automated support this method has been implemented in a tool called ViTAL [1](A **V**eri**i**fication **T**ool for EAST-**ADL** Models using UPPAAL PORT) [8]. Our approach allows the specification of the behavior inside each elementary function (block) in Timed Automata (TA) [9] and constructs a complete system behavior model by the parallel composition of local behaviors. In particular, we describe the behavior of each EAST-ADL function block as a UPPAAL PORT TA model. A composition of function behaviors is considered a TA network that enables one to analyze and verify the entire system using the UPPAAL PORT model checker. By employing such an approach, we bridge the gap between architectural modeling and formal verification of automotive embedded systems.

The methodology builds on our previous work [10], however, in this paper we introduce the automatic translation of EAST-ADL models into UPPAAL PORT models, as well as the integrated tool-support for analysis. Moreover, this method is not restricted to a particular component model, rather, it assumes a *read-execute-write* semantics from the latter. We show the proposed methodology at work on an industrial prototype, the Brake-by-Wire System.

ViTAL provides model-checking of EAST-ADL descriptions with respect to timing and functional behavioral requirements. The system designer creates the EAST-ADL models and the execution behavior using the TA framework and checks whether a given requirement is satisfied. For this, we implement an automatic model transformation to UPPAAL PORT, which enables the latter to handle EAST-ADL models as input and provide functional and timing behavior of functional blocks using TA semantics. To increase user friendliness and to align with the implementation of the EAST-ADL profile, our integrated environment is based on Eclipse plug-ins. ViTAL contains the following: (i) an editor for describing the behavior of EAST-ADL function blocks as TA models, (ii) an automated transformation of EAST-ADL models to UPPAAL PORT TA models, (iii) support for mapping external TA variables to external ports, (iv) a simulator that can be used to validate the behavior of the EAST-ADL modeled system, and (v) support for verifying reachability (can a particular goal be reached?) and liveness (can a particular goal be guaranteed to be met eventually?) properties formalized in a subset of Timed Computation Tree Logic (TCTL), the property specification language of UPPAAL PORT.

This work is organized as follows: Section 2 briefly overviews EAST-ADL and UPPAAL PORT. In section 3, we introduce the Brake-by-Wire system running example. In section 4, we describe our proposed EAST-ADL modeling and verification approach. In section 5, the tool environment, ViTAL, is presented. In section 6, we demonstrate the applicability of our method by verifying the Brake-By-Wire system by employing ViTAL. Section 7 presents related work. We discuss future work and conclude in section 8.

2. Preliminaries

2.1. EAST-ADL

EAST-ADL is an architecture description language specified through a meta-model and implemented as a UML2 profile [4], intended to support the development of automotive embedded systems by capturing modeling related engineering information. EAST-ADL handles various types of information including the specification of system environment, related requirements, deployment of software and hardware resources, variability, performance constraints such as behavior and timing constraints, dependability, and V&V (Verification and Validation) related information.

The definition of an EAST-ADL system structure (e.g. components and their connectors), referred to as *System Model*, is given multiple levels of abstraction to deal with complexity control, safety, reliability, cost improvement and efficient development through MBD according to the fundamental characteristics of the system lifecycle. These levels have complete traceability between them, from a higher to a lower abstraction layer, as can be observed in Fig. 1. The support of EAST-ADL ranges from this multi-level system definition to the specification of requirements and related V&V efforts: At `Vehicle Level` the electronic features are described. At `Analysis Level` a complete

[1] ViTAL tool is available at www.vitaltool.org.

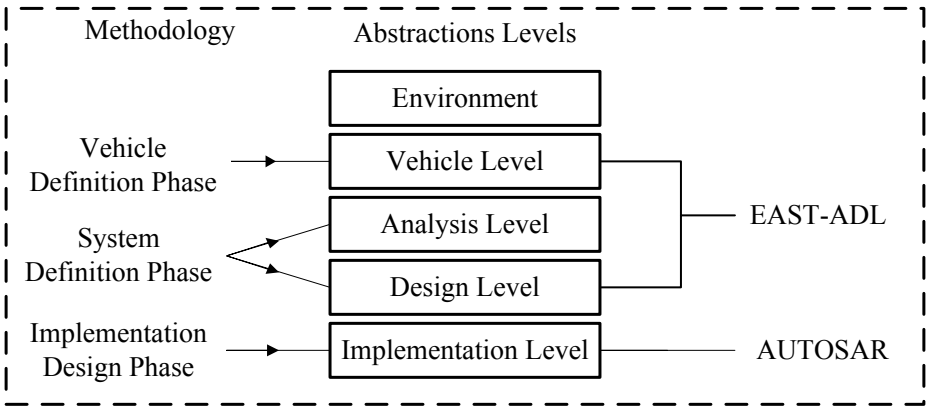

Figure 1. Overview of the EAST-ADL architecture.

representation of the abstract functional definition of features in system context is modeled. This representation is intended to capture the interfaces and behavior of the vehicle subsystems and to allow validation and verification of the integrated system or its subsystems on a high level of abstraction. At `Analysis Level` the features and requirements are refined by iterating the software application together with the other levels of abstraction. At `Design Level` the detailed functional definition of software is introduced according to the realized logical design. The functional abstraction of hardware and middleware are also presented, as well as hardware architecture being captured and function-to-hardware allocation being defined. At `Implementation Level` reusable code and AUTOSAR [2] compliant software and system configuration for hardware deployment are described [4].

EAST-ADL is intended as an integration framework for functionality defined in different notations and tools. The behavioral description relies on the definition of a set of elementary functions that are executed based on the assumption of "read-execute-write" semantics (read inputs from ports, compute, and write outputs on ports). This is chosen to enable analysis and behavioral composition and make the function execution independent of internal behaviors.

2.2. UPPAAL PORT

UPPAAL PORT is an extension of the UPPAAL tool, which supports simulation and model-checking of component-based systems, without the usual conversion or flattening to the model of network TA. This is complemented by the Partial Order Reduction Technique (PORT) [11, 7] that UPPAAL PORT uses, to improve the efficiency of model-checking analysis. This technique has been proposed in order to reduce the state-space explosion caused by interleavings, with the main idea of exploring only a relevant subset of the state-space when model-checking [12]. UPPAAL PORT only explores properties which preserve a subset of the full model based on independence of transitions, instead of examining all possible sequences, which would unnecessarily add to the model-checking complexity. Since the synchronization of global time restricts the independence of transitions in the timed automata framework, UPPAAL PORT uses local time semantics to increase independence, therefore allowing the analysis of components in isolation followed by synchronization to a shared state whenever one writes to another.

A component is defined by its interface and a timed behavior named UPPAAL PORT TA. This special type of TA [7] is defined as a timed automaton $B = (N, l_0, l_f, V_D, V_C, r_o, r_f, E, I)$ where N is a finite set of locations, l_0 is the initial location, l_f is the final location, V_D and V_C are a set of data and clock variables, r_0 and r_f are sets of initial and final reset clocks, E is a set of edges, and I is mapping each location l to its invariant $I(l)$. In the case of $(l, g, e, r, l') \in E$, we write $l \xrightarrow{g,e,r} l'$ to describe an edge from location l to l' with guard g, action update e, and reset clocks r.

In UPPAAL PORT the employed component model has transitions which are independent of actions in other components. The performed experiments [7] suggest the use of UPPAAL PORT when dealing with a "read-execute-write"

[2] AUTOSAR (AUTomotive Open System ARchitecture) is a standardized interface that defines a common software infrastructure for automotive systems. More information can be found on www.autosar.org

component model semantics. The model checker of UPPAAL PORT verifies properties expressed in a subset of timed *computational tree logic* (TCTL). The current input file format for UPPAAL PORT is a component modeling language [13], which describes the architectural framework for modeling real time embedded applications with particular emphasis on automotive domain and safety concerns. In particular, this input language is used to create components and interconnections among them and supports run-to-completion semantics.

3. Running Example: Brake-By-Wire

Our Brake-by-Wire (BBW) case study is provided by VOLVO Technologies within the MBAT project [14]. The vehicle is software-simulated and the functionality of the system can be divided between sensors, actuators, and computations. The brake pedal sensor transforms the position of a pedal into a percentage of a maximum value of the brake force and the wheel sensor reads the actual wheel speed. The wheel actuator computes the brake force and the computation is divided as follows: the brake torque calculator computes the requested brake force, which is used as input in the global brake controller that calculates a basic brake force for each of the wheels. The ABS controls the wheel braking in order to prevent locking of the wheel based on the slip rate value.

The system consists of five Electronic Control Units (ECUs) connected by a network bus: a central ECU connected to the brake pedal and four ECUs connected to the wheels. The central ECU has three components: a brake pedal sensor, a component that calculates the global brake torque from the brake pedal position, and a component that distributes the global torque to the four wheels. Each wheel ECU also has three components: a sensor that measures the wheel speed, a component for the brake actuator, and an ABS controller. The ABS controller is based on a simple logic: if the slip rate of the wheel is larger than 0.2, then the brake actuator is released and no brake is applied. Otherwise, the requested brake torque decided by the central ECU is used.

The BBW system, modeled in EAST-ADL, needs thorough quality assurance up to formal verification and as such this system has to obey strict timing requirements. The system should react timely to prevent serious accidents, for example. Timing behavior and its analysis play a crucial role in the early development of embedded and critical systems in order to avoid late detection of design flaws which negatively impacts cost: Verifying the end-to-end timing requirements, from a sensor to an actuator, must take into account the timing properties of all the components. Further details will be given in the following sections.

4. Methodology and Proposed Solutions

In this section, we describe how our approach enables formal analysis of EAST-ADL descriptions, by model-checking their behavior with UPPAAL PORT, independent of platform characteristics and constraints. The verification is tailored for the Analysis Level of EAST-ADL, and is composed of the following steps, mirrored in Fig. 2: (i) Architectural Mapping, (ii) Integrating the Behavioral Formal Model, and (iii) Simulation and model checking. We discuss these steps in further detail in the following sections.

4.1. Architectural Mapping: EAST-ADL *to* INTERMEDIATE COMPONENT MODEL

This architectural mapping step, M2M in our methodology roadmap (Fig. 2), is an architecture-to-architecture transformation, from EAST-ADL to an Intermediate Component Model (ICM) that can be further integrated with the formal behavioral model of TA. We work at the Analysis Level in EAST-ADL, such that we need to map essentially all the interface elements alongside the existing timing annotations.

In order to integrate the formal model of UPPAAL PORT TA with EAST-ADL, we need to first perform a semantic anchoring of the latter to a component model that obeys the *read-execute-write* semantics, hence preserving the informal semantics of EAST-ADL without altering its structure. Each elementary *Function Prototype* (f_p) has its own logical execution and no internal concurrency, therefore it can be straightforwardly mapped to a component in the intermediate component model. We assign one component per f_p element of the EAST-ADL model (e.g., BrakeController, ABS, in our running BBW system).

Assume an EAST-ADL composite EAST-ADL *Function Type* defined as the following tuple:

$$FunctionType \triangleq \langle F_P, DP, Con, TC \rangle,$$

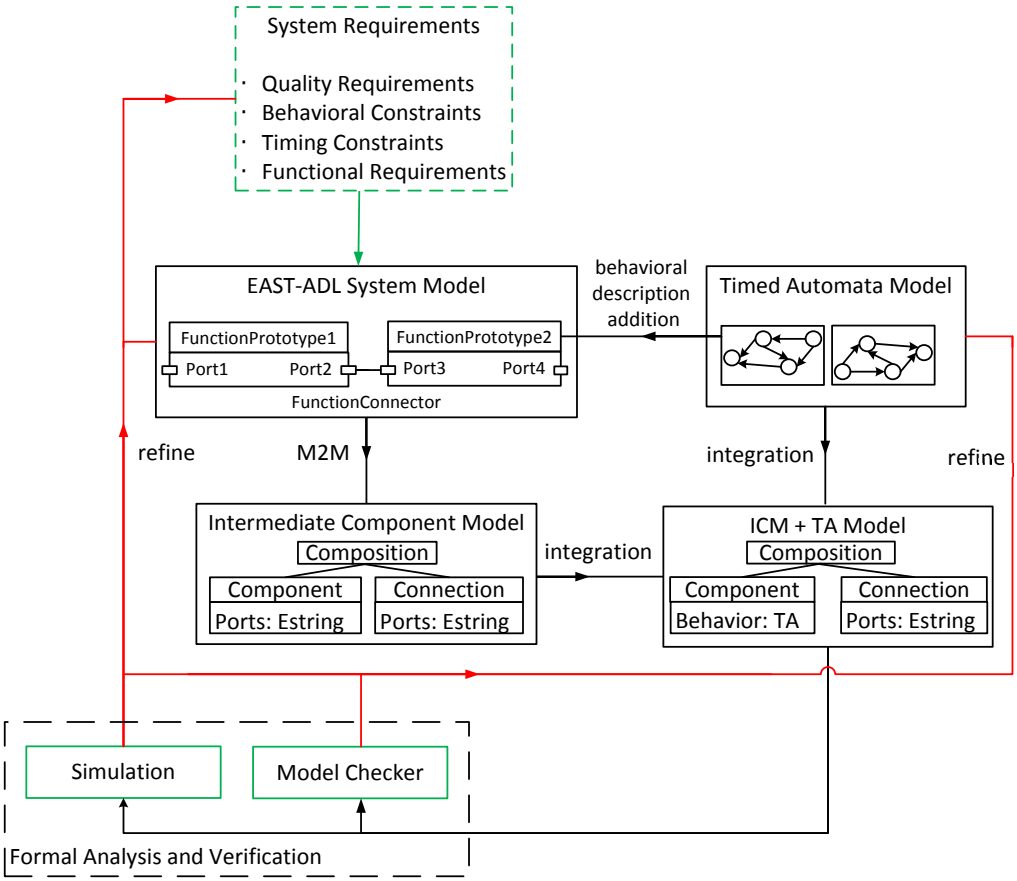

Figure 2. Methodology Roadmap

where F_P is the set of Function Prototypes (f_p) of a Function Type, DP is the set of data ports, defined as the reunion of input ports and output ports, Con is the set of connectors, and TC is the set of the model's timing constraints. In the case of $F_P = \emptyset$ and $Con = \emptyset$, we obtain a primitive EAST-ADL component. In our approach pictured in Fig. 2, each Function Prototype (f_p) is associated with a formal behavior described in UPPAAL PORT TA that is recalled in section 2.2.

We also assume our Intermediate Component Model (ICM) is defined as a tuple as follows:

$$ICM \triangleq \langle Comp, P_{in}, P_{out}, trig, Connections, TC_{ICM} \rangle,$$

where $Comp$ is the set of components that ICM contains, P_{in}, and P_{out} are the input and output dataflow ports, respectively, $trig$ is the trigger variable that is set to TRUE by a clock component, and TC_{ICM} is the ICM's set of timing constraints. If $Comp = \emptyset$ and $Connections = \emptyset$, then ICM is a primitive component.

The architectural mapping between an EAST-ADL $FunctionType$ and an ICM, defined as above, is a function $\pi : FunctionType \rightarrow ICM$, which maps each function prototype f_p to an ICM component, input ports to the ICM's component dataflow input ports, output ports to the ICM's component dataflow output ports, connectors to the ICM's component connections, and the Function Type's timing constraints TC to ICM's timing constraints, TC_{ICM}.

One or more ports and elements of EAST-ADL models may be realized by one port and one component of ICM, as these may have several signals or data elements per interface, which are simplified in one port and one component in our ICM representation. Function interactions within a $FunctionType$ can be either $FlowPort$ interactions, in which a function performs computations on provided data, or $ClientServer$ interactions, where the execution of a service is called upon by another function. However, in this paper we consider and implement only $FlowPort$ interactions. The triggering of each $FunctionType$ is defined either as time-driven or event-driven on one of the input ports. There

are two types of function entities, time-discrete functions and time-continuous functions. A time-discrete function is performed after a computational delay, whereas a time-continuous function defines the transfer function from input to output, and the computation rate is infinite. However, on our mapping we do not deal with time-continuous functions, but only with time-discrete ones. The latter are invoked either in a time-triggered fashion, in which case time alone causes execution to start, or in an event-triggered manner, which is caused by data arrival or calls on the input ports. The trigger conditions are matched by those of a clock component and data type ports in *ICM*, respectively.

4.2. Integrating the Behavioral Formal Model: Mapping ICM *to* TA

In the previous sections, we have shown a straightforward mapping from the informal semantics of EAST-ADL to the intermediate component model, *ICM*. Since EAST-ADL allows the use of various behavioral notations, we exploit this feature to our advantage, and specify the function behavior by assigning an UPPAAL PORT TA model to each *ICM* component mapped from its corresponding $FunctionPrototype(f_p)$; in this procedure, we comply to the triggering definition, as well as translate the execution time of each f_p, and its requirements. The TA model encapsulates the "internal behavior" of a $FunctionPrototype$ and is used as input to the UPPAAL PORT model-checker that we use to carry out the formal analysis and verification of functional and timing EAST-ADL properties.

Let us consider the Intermediate Component Model (ICM) as defined in section 4.1:

$$ICM \triangleq \langle Comp, P_{in}, P_{out}, trig, Connections, TC_{ICM} \rangle.$$

The behavior of a $FunctionPrototype$ mapped onto a component within the set $Comp$ of *ICM* is modeled as an UPPAAL PORT TA, as in section 2:

$$TA \triangleq \langle L \cup \{l_\perp\}, l_0, l_f, V_C, V_D, r_0, r_f, E, I \rangle,$$

where L is the set of TA locations, extended with the idle location $l_\perp$, $l_0 \in L$ is the initial location, $l_f \in L$ is the final location (used to model the termination of a function execution), such that no edges in E are leading out from l_f. The rest of the tuple elements have the same meaning as described in section 2. We also define the action $Read(P_{in})$ for reading variables from P_{in} and $Write(P_{out})$ for writing variables onto output ports.

The execution of the *ICM* behavior is determined in terms of the triggering value, which changes according to a periodic clock. When $trig$ holds, the TA data variables in V_D are updated by READ(P_{in}), and WRITE(P_{out}), respectively, which are atomic and urgent (in the sense that time is not allowed to pass when a component reads or writes). A component is initially *idle*, and after performing the read action it starts executing until its internal computation is done. After completing the write action, which forwards data from the output ports via connections, the component becomes *idle* again. That being said, we define the mapping between *ICM* and *TA* as follows:

Mapping ICM **to** TA**.** Given *ICM* and *TA* as above, the mapping rules for integrating them are:

- $l_\perp \xrightarrow{g,read,r_0} l_0$ The read action is READ(P_{in}) that updates V_D with input values from P_{in}. The clock r_0 is the initial clock reset;
- $l_f \xrightarrow{g,write,r_f} l_\perp$ The write action is WRITE(P_{out}) that writes on output ports, whereas r_f is the final clock reset;
- $I_\perp(l_\perp) = \text{true}$, $I_\perp = I(l)$ for $l \neq l_\perp$.

The TA of a composition C, TA(C), is defined as a network of local TA. For f_{p_i} and its corresponding component $ICM_i \in C$, the write action in TA(ICM_i) is extended to update the input ports (noted $P_{in.j}$) of a target component $ICM_j \in C$, according to the connections from the outports of ICM_i (noted $P_{out.i}$). A connection connects a source port $p \in P_{out.i}$ to a target port $p' \in P_{in.j}$. The edges e of TA(ICM_i) are explained with *extended write actions* as follows.

Extended Write Actions. Assume $TA(ICM_i)$ is defined as $\langle L, l_0, l_f, r_0, r_f, V_D, V_C, \{XWRITE_i(e) \mid e \in E\}, I \rangle$, such that the following hold:

- $XWRITE_i\ (l \xrightarrow{g,a,r} l') \triangleq (l \xrightarrow{g,w,u} l'\ ; WRITE(P_{out.i}))$, if $a = write$. Note that ";" represents the sequential execution;

- $XWRITE_i\ (l \xrightarrow{g,a,r} l') \triangleq l \xrightarrow{g,a,u} l'$, if $a \neq write$.

The automata TA(C) is then the network of TA(C_i), for $C_i \in C$.

An environment is modeled as TA_{Env} in a similar way. The resulting composition is thus defined as the network $TA(C) \times TA_{Env}$, where any edge in TA_{Env} updating ports P_{in} of C, is extended with an update $WRITE(P_{out.Env})$. This is similar to the adaptation of the $XWRITE$ action that is used to build $TA(C_i)$ in Definition 4.2.

4.3. Simulation and Model Checking

The execution of each *FunctionType* in EAST-ADL is specified by UPPAAL PORT TA and integrated within *ICM*, as described in section 4.2. The entire system (network of UPPAAL PORT TA) has its semantics given in terms of timed transition systems [15], and this whole model is verified by the UPPAAL PORT model-checker. Functional requirements, timing, as well as other behavioral constraints (see Fig. 2) are formalized as safety properties in Timed Computation Tree Logic (TCTL) used by UPPAAL PORT.

All requirements are derived from given system textual descriptions. On the one hand, we can verify certain delay, reaction and synchronization constraints (that is, overall behavioral constraints of a system). On the other hand, we may also check functional requirements, like particular triggering properties associated to an EAST-ADL function block. The UPPAAL PORT model checker verifies such functional requirements and behavioral constraints, which are formalized in TCTL as safety properties. If UPPAAL PORT reports a counter-example to a particular model, meaning that the property under investigation does not hold for all possible system behaviors, the resulting trace could be used to refine the behavioral constraints of the system model, or modify the original EAST-ADL model, or the TA behavioral description.

5. Tool-supported Modeling and Analysis

The architecture of the tool is separated into two distinct layers. The bottom layer is composed of standard editors offered by the Eclipse platform. These editors are used to design and transform the models used within the tool. Our contribution lies in the top layer, in which the ViTAL modeling and analysis approach is implemented. This section introduces the ViTAL modeling approach for EAST-ADL system models. In order to align UPPAAL PORT with the implementation of the EAST-ADL profile, we propose an integrated tool based on Eclipse plug-ins, as observed in Fig. 3. Our modeling and verification tool contains the following: an editor for TA description of the functional and timing behavior of EAST-ADL functional blocks, automated transformation of EAST-ADL models to UPPAAL PORT input model, support for mapping internal TA variables to external ports, a simulator that can be used to validate the behavior of an EAST-ADL modeled system, and support for verifying reachability and liveness properties formalized in a subset of TCTL.

5.1. Modeling Approach

In most cases an automotive system is composed of a large number of software and hardware vehicle functions. EAST-ADL depicts the whole automotive system from requirements specification until design and implementation. We employ the EAST-ADL functional model on analysis and design level as the modeling language to specify the structure of a system. The challenge of EAST-ADL system modeling on analysis and design level is to target different system components for concepts like allocation of requirements, analyzing, and verifying functional requirements before implementation. On the analysis and design levels, the system is described by a Functional Analysis Architecture (FAA) and Functional Design Architecture (FDA), respectively [4]. FAA can be defined separately from the other levels and we can assume that, from an FAA-level point of view, the description is complete with respect to the dependencies between different vehicle functionalities. The system on the FAA is composed of a number of interconnected f_p, where each prototype is an instantiation of f_t [2]. Inside an f_p the composing sub-functions have synchronous execution semantics. These functional blocks are *time-triggered*, or triggered based on data arrival on their flow ports. An f_p follows the *"read-execute-write"* semantics, which specifies that once a function is triggered, it reads the input flow ports, executes the computation, and then writes to its output flow ports, all without interruption.

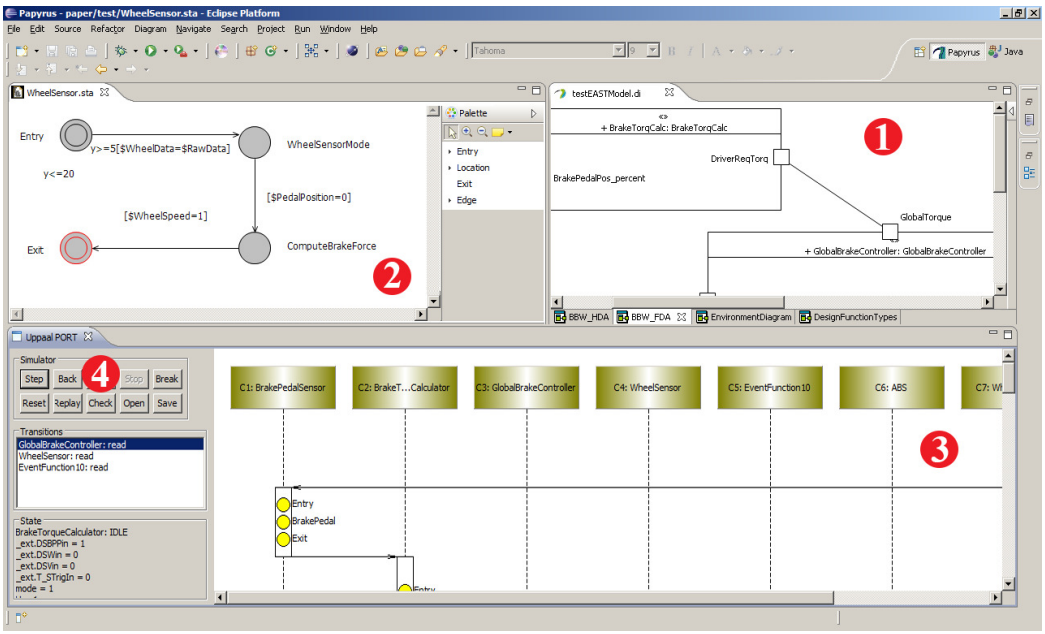

Figure 3. Integrated EAST-ADL Platform Editor (1), Timed Automata Editor (2), UPPAAL PORT Simulator (3), and the UPPAAL PORT Verifier (4).

5.1.1. *The* EAST-ADL *System Model*

Using the EAST-ADL system and behavioral model, we describe how functional blocks are related to each other in capturing behavior and algorithms of the system as well as the environment. In EAST-ADL system modeling, functional blocks in FAA are built from interconnected f_ps with well-defined interfaces consisting of a set of input- and output data ports. In the current work, the architectural specifications are used from structure, behavioral, and timing EAST-ADL packages [4, 16] in order to simplify the definition of the semantics and the integration towards UPPAAL PORT. The core ICM consists of the following modeling elements: composite `Analysis Function Type`, basic `Analysis Function Prototype`, connectors `Function Connector`, `Flow Ports`, `Event Function`, `Event Function Flow Port` associated to a set of ports, `Periodic Constraint` associated to an f_p , `Delay Constraint`, and a behavioral description named `Behavioral Model`. All the elements, except the last, are translated from the EAST-ADL structure package. The behavioral description complemented with timing formalisms complies with the component-based approach that enables early formal analysis of relevant concerns.

5.1.2. *Timed Behavior*

The timed behavior of a functional block is defined as a TA, extended with data variables and a final location. An f_p is described by its interface in terms of ports and a timed behavior [7]. The TA model is an abstraction of the f_p behavior and assumed properties. For analysis and verification of EAST-ADL FAA models in UPPAAL PORT, each f_p is associated with a behavioral model consisting of TA, and a mapping between ports and the TA variables. We provide a mapping scheme between flow ports and TA variables as additional model parameters.

5.2. *Model transformation to* UPPAAL PORT

In this work, no f_t can have instances of other f_t with the exception of FAA. We have developed the model transformation shown in Fig. 4. We specify the input models for UPPAAL PORT as described in the *ICM*. In order to simplify the semantic definition of *ICM*, we focus on the EAST-ADL language constructs relevant for functional and timing verification. For this reason, other modeling constructs, such as requirements models and variability models, are not addressed in this work.

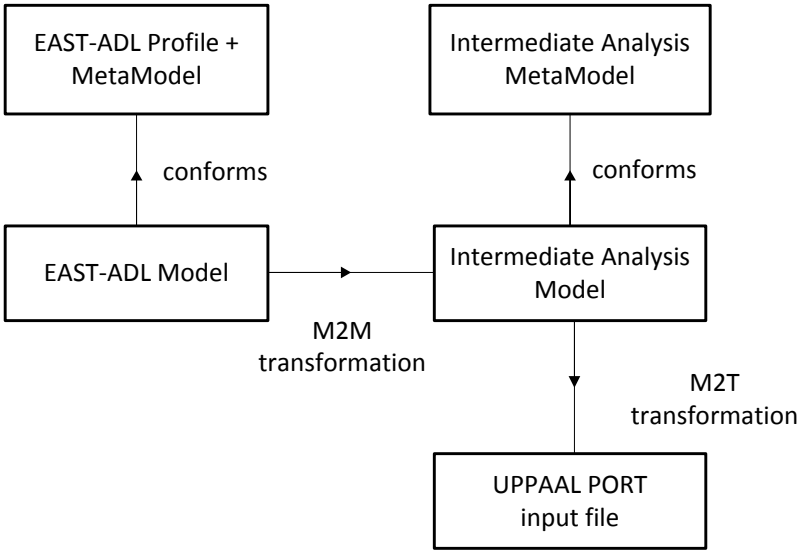

Figure 4. Model Export from EAST-ADL to UPPAAL PORT.

5.2.1. ViTAL and ICM Integration

We define a structural integration inside the ViTAL tool, which consists of the formal definition of the constructs of the EAST-ADL language. The ICM consists of the following elements: composite f_t, basic f_p, connections, ports, event functions, and timing constraints. Using these elements, we derive all constructs in the EAST-ADL system model. We consider that each modeling element, except for the FAA f_t, has a set of flow ports, through which it can interact. `Flow Ports` are associated by a set of EAST-ADL types named `EADataTypes`, which are used for data representation, e.g., integer with the specific range `EAInteger`. A `Flow Port` is associated with the same type of data as the associated variable.

Similar to the EAST-ADL language, `Function Connectors` define how data can be transferred between two $f_p s$. We assume there is no time consumption for data transmitted through a connector. This assumption is used when modeling the abstract functional system in EAST-ADL at analysis level, in which most implementation elements are hidden. Other structural EAST-ADL constructs are not represented directly by any modeling element, hence they do not influence the transformation. For the integration presented in ViTAL, the architectural information related to structure and timing are partially derived from the EAST-ADL model. Every f_p is annotated in the ICM with an `Event Function` that submits to a `Periodic Constraint`. An `Event Function` is a trigger generator annotated with a parameter T for period. A new period starts every T time units, and the event function generates a trigger after each period elapses.

The EAST-ADL model restricts the f_p behavior. For example, the *read-execute-write* semantics mentions that input `Flow Ports` may only be accessed at the beginning of each triggering, and output `Flow Ports` are only written at the end of the computation. Therefore, TA(f_p) defines its behavior augmented with an external ICM interface. The interface of an f_p consists of `Flow Ports` and the triggering information. An input `Flow Port` has an associated variable holding the current data value. A primitive f_p corresponds to a primitive ICM `Function Prototype` with an automaton that can capture the behavior of the associated f_t. The internal computation of an f_p starts with reading all input `Flow Ports`. This internal input data is used together with other functional information during the f_p execution, before writing the variables to the output `Flow Ports`. With these assumptions in mind we can describe the ICM's actual semantics.

The modeling elements of the ICM, used in the integration, are described in Fig. 5 where the elements represent the structure information at analysis level of abstraction, model behavior, and timing information. The ICM provides system modeling concepts which are mapped to concepts of component based design. `Element` can be mapped with a `FunctionType` and `EventFunction` elements. There are connectors such as `FunctionConnection` and ports

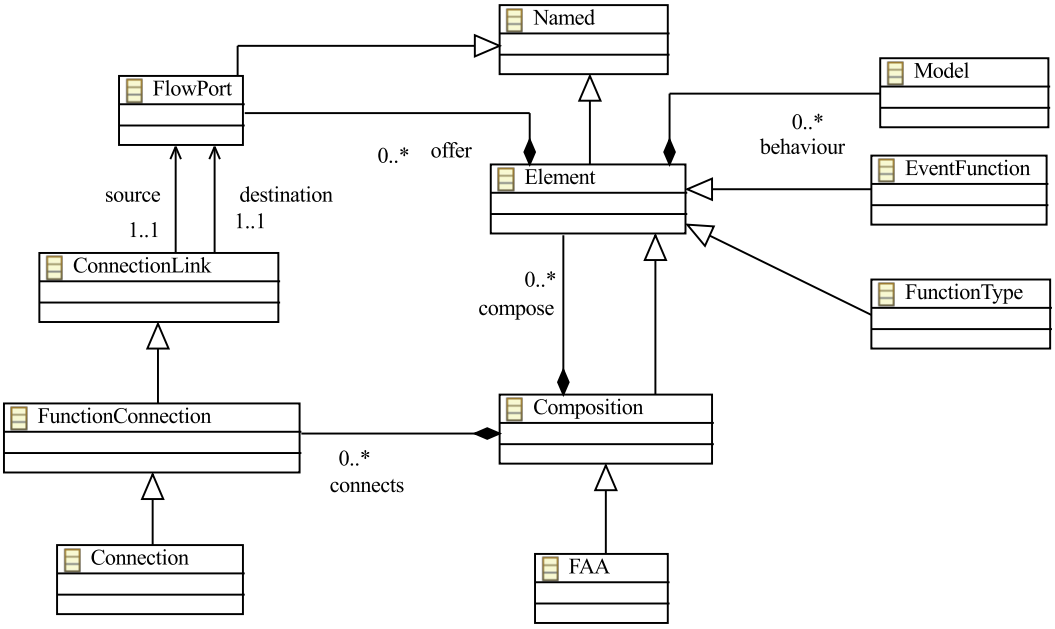

Figure 5. Simplified diagram representing the meta-model elements of the ICM.

such as `FlowPort`. A `FlowPort` is of type `Analysis Data Type`. Furthermore an `Element` can have a behavioral description as a `Model` element. Every `FlowPort` can be provided with an associated `EventFunctionFlowPort`, which is used together with the `DelayConstraint`.

The ICM obtained after the transformation represents the TA behaviors with triggering and timing information and some assumed functionality. ViTAL is used in this case to provide an extension to the internal behavior of f_p not only in terms of timing, but also functionality.

5.2.2. Implemented Model Transformation

The EAST-ADL language is implemented as a UML2 profile that provides the ability to describe EAST-ADL-compliant models using this profile. The Papyrus UML tool implements the stereotypes as defined in EAST-ADL specification. ViTAL is based on Eclipse[3], which ensures an integration with the UML Editor in Papyrus, needed for developing EAST-ADL models. The EAST-ADL editor plugin uses Papyrus UML to create the actual EAST-ADL system model. Papyrus saves its models in two files, one using a ".uml" extension and the other using a ".di" extension. The former file contains the actual model information, whereas the latter file contains all the graphical information. The VITAL tool provides an intuitive and user friendly graphical environment based on a popular development platform named Eclipse. Our transformation and verification tool is built upon an MDD set of Eclipse plug-ins: Eclipse Modeling Framework [4], Graphical Modeling Framework, Graphical Editing Framework, ATLAS Transformation Language (ATL), and Acceleo. Fig. 7 shows the EAST-ADL model transformation.

The Papyrus UML Editor generates EAST-ADL models, which are used for our mapping to UPPAAL PORT. As shown in Fig. 4, we introduce an ICM as the interface between the EAST-ADL model and UPPAAL PORT input model. The intermediate model conforms to the EAST-ADL metamodel that is aligned with the EAST-ADL profile and UML metamodel. The structural transformation maps an EAST-ADL model, which was created in the Papyrus UML modeling environment, into ICM. The Model to Model transformation is shown in Fig. 4. The structure of the ICM is

[3]Eclipse is a software development environment with an extensible plug-in system.
[4]Eclipse Modeling Framework (EMF) is a modeling framework for viewing and editing of models.

```
1  @path ICM=http://www.mdh.se/analysis @path
2  EAST=http://www.papyrusuml.org/EAST-ADL2/1
3
4  module EAST2ICM; create OUT1 : ICM from IN1 : EAST;
5
6  rule TopLevelADLContainer {
7      from
8          s : EAST!Model
9      using {
10
11         allFunctionTypes : Set(EAST!AnalysisFunctionType) =
12             EAST!AnalysisFunctionType.allInstances();
13
14         allConnectors : Set(EAST!FunctionConnector) =
15             EAST!FunctionConnector.allInstances();
16
17     }
18     to
19         t : ICM!FAA ( name   <- s.name, id      <- '1',
20             compose <- allFunctionTypes,
21             connects <- allConnectors -> collect(aLine | thisModule.Connector2Conn(aConn)))
22 }
```

Figure 6. ICM Transformation rules for the associated elements.

the basis of the Model to Text transformation needed for the integration of the UPPAAL PORT input model [5]. This step of the transformation achieves an integration between the modeling domain of EAST-ADL and that of UPPAAL PORT. We use the ATL Model to Model Component due to its simplicity and integration within the Eclipse platform. We have implemented the mapping rules presented in Section 5.2.1 as an ATL description of the transformation logic. For the transformation, a few modifications of both metamodels have been made. In principle, these changes are in fact ways to preserve the semantics of the original model. For instance, EAST-ADL uses the type - prototype constructions in which the declaration is in f_t, and the actual usage is in f_p. In this case, the structural transformation has a pointer between f_t and the contained TA(f_p).

Intermediate Model Transformation: This step of the integration is based on the ATL model transformation language [18], which is composed of the following elements: a header section that defines the name of the transformation module and the variables corresponding to the intermediate and EAST-ADL models, a set of helpers, and a set of rules that defines the way intermediate models are generated from the EAST-ADL ones. Based on these attributes introduced for model transformation, we have defined some rules of transformation. The M2M transformation depicted in Fig. 4 is described for the top level rule of the transformation in Fig. 6. We defined the M2M transformation within an `EAST2ICM Module` (line 4). It operates on a non-empty source model `EAST` producing a non-empty target model `ICM` (line 5). The `EAST2ICM` has a number of associated `ModuleElements` which are either `Rules` or `Helpers` (lines 7-23). The `FlowPort Helper` is used as a query function in the context of the `EAST2ICM Module` itself for matching the `EAST Connection`'s explicit flow direction. We provide these transformation `Rules` to produce target elements such as `TopLevelADLContainer` (line 7), including the inports and outports.

In addition to the structural integration, the behavioral TA transformation needs to be carried out. In order to model the timing and behavior of an f_p, we use a TA editor. The behavior can be represented in a graphical editor by the developer. These models are different from UPPAAL TA models as follows: (i) the timed behavior is extended with a final location out of which no edges are leaving, and (ii) synchronization channels are not allowed, because of the semantics used by ICM models. For more details on the specifics of the TA employed by UPPAAL PORT, we refer the reader to the work in [7]. We map TA variables to the flow ports of the EAST-ADL functions. We provide a variable to the port mapping plug-in. In the current version of ViTAL, the mapping uses the name of the TA file to automatically generate the parameters to be used.

Once the aforementioned steps have been completed, the TA and ICM can be merged into an XML-format accepted by the UPPAAL PORT tool [6]. This transformation is carried out using the Acceleo code generator to transform the

[5] We refer the reader to the UPPAAL PORT input language reference manual for further details [17]

[6] The XML syntax describing the element definitions from the Document Type Definition is available in the SaveCCM language reference

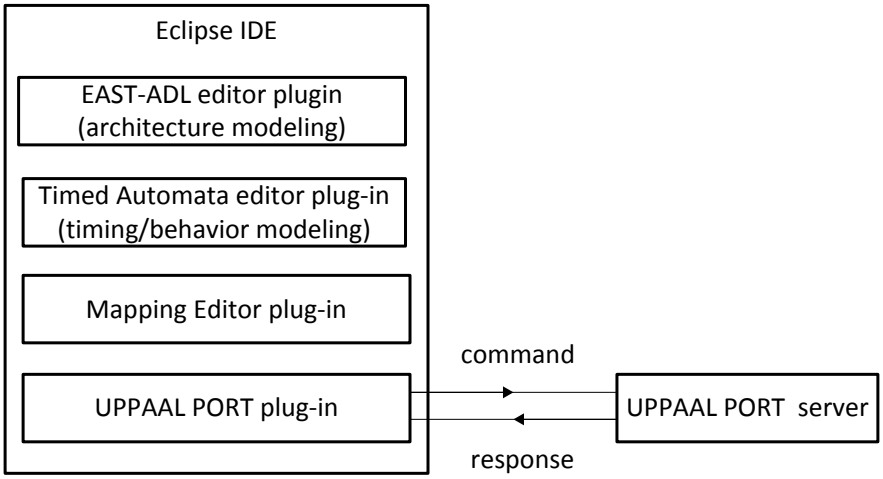

Figure 7. Overview of the ViTAL tool architecture.

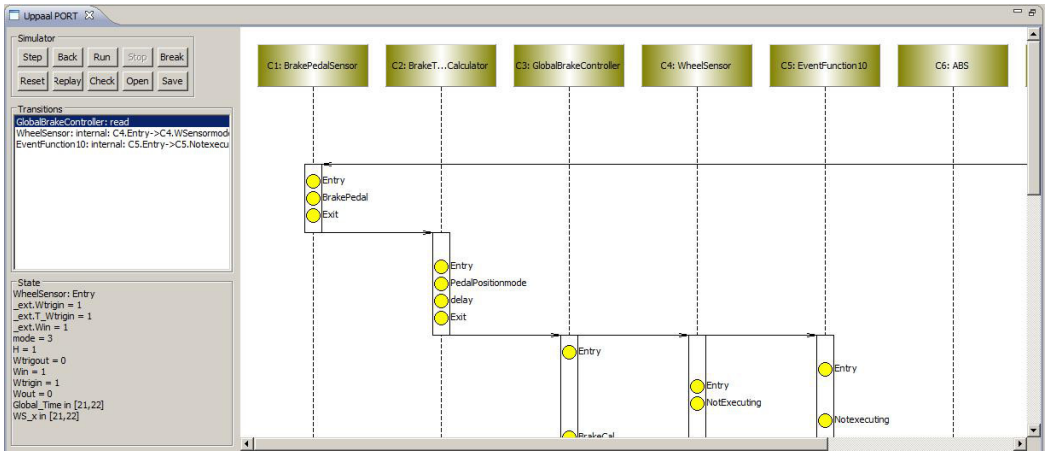

Figure 8. Simulator View in the ViTAL tool chain.

ICM into code. The ViTAL tool architecture is shown in Fig 7. The user interface contains an editor for EAST-ADL models in the Eclipse framework, as well as a TA editor to model the behavior of EAST-ADL system models. UPPAAL PORT introduces support for simulation and verification, using a client-server architecture [19]. The UPPAAL PORT model-checker consists of two modules: the Eclipse plug-in used as the graphical simulator and the server executing the verification.

The simulator is used to validate the behavior and timing of an EAST-ADL model. The simulator allows stepping forward and backwards in the state space, while selecting possible transitions, as described in Fig. 8. In a trace, a data transfer is defined by the exchange of data between EAST-ADL f_ps (annotated with the timed automata locations) via their respective ports. Also, by using the verifier , it is possible to establish, by model checking the described TA behavior, whether the EAST-ADL system model satisfies the functional or timing properties, as specified by the requirements engineer in EAST-ADL language.

manual [17].

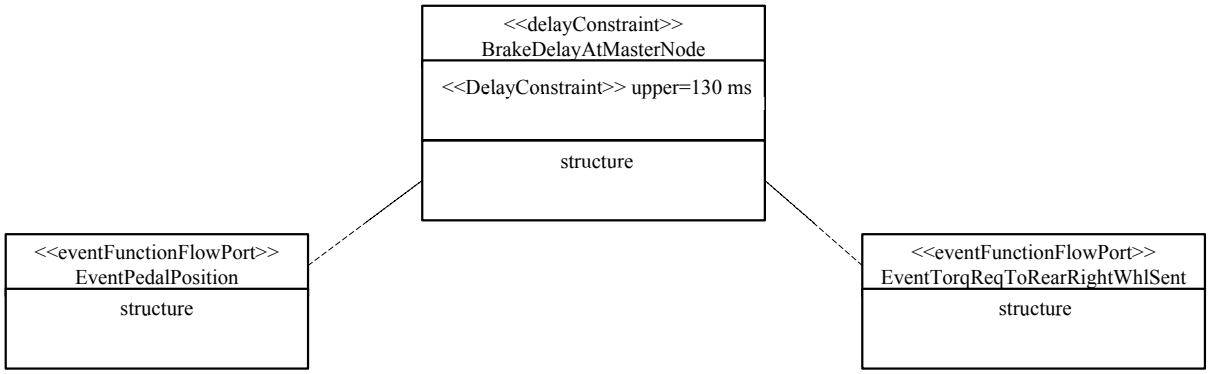

Figure 9. Associated timing constraints specified in the EAST-ADL Models.

5.2.3. Automatic verification of timing constraints

One of the key issues in the context of automotive software systems is verifying timing requirements on the bus or the ECU timing properties in EAST-ADL at Analysis and Design Level. These timing properties are specified in the EAST-ADL language in the structure package as `delayConstraint`. This artifact is attached to two `eventFunctionFlowPorts` that are used to constrain two `Flow Port` artifacts and defines bounds on system timing attributes.

The `delayConstraint` requires a delay between a set of input `Flow Ports` and a set of output `Flow Ports`. To illustrate, we use a simplified example of a timing constraint associated to an EAST-ADL system model, as depicted in Fig. 9. This structure specification can be used in the M2M transformation and in the intermediate model to validate existing realizations of the system.

We have augmented the intermediate metamodel with the `delayConstraint` structural information, which should also include the associated `Flow Ports`. With ViTAL, we may go a step further towards automatic verification of delay constraints. Starting with the delay constraints information contained in the EAST-ADL model, we build a refined model of the intermediate model. Refining a constraint may consist in fixing parameters values, in accordance with the initial specification. Then, ViTAL simulates the refined model, revealing possible inconsistencies in the specification, by using the UPPAAL PORT verification engine.

6. Case Study: Brake-by-Wire Control System

We check the applicability of our approach on the Brake-By-Wire (BBW) system, modeled in EAST-ADL. Fig. 10 depicts a simplified schematic illustration of the BBW with Anti-lock Braking System (ABS) function. The system reads the pedal position percentage used to compute the desired torque and calculates the torque required for each wheel. The wheel sensor measures the wheel speed for the controller and the ABS controls braking to prevent locking of the wheel based on the slip rate.

6.1. BBW System Model and Functionality

The functionality of the system can be divided between sensors and actuators f_ps, and computations f_ps. The `Brake Pedal Sensor` (BPS) transforms the raw data (the voltage that is related to the pedal angle) into a percentage of a maximum value of the brake force. In the case of the `Wheel Sensor` (WS), the raw data corresponds to the actual wheel speed. The `Wheel Actuator` (WA) is modeled as an f_p, which computes the brake force. The computation is divided between two f_ps: the `Brake Torque Calculator` (BTC) computes the requested brake force, which is used as input in the `Global Brake Controller` (GBC) that calculates a basic brake force on each of the wheels. The basic brake force may be adjusted depending on the difference between the estimated vehicle speed and the respective wheel speed.

The system is composed of five Electronic Control Units (ECU) connected by a network bus. The central ECU has three components: the BPS that reads the position of the pedal percentage, the BTC that computes the desired

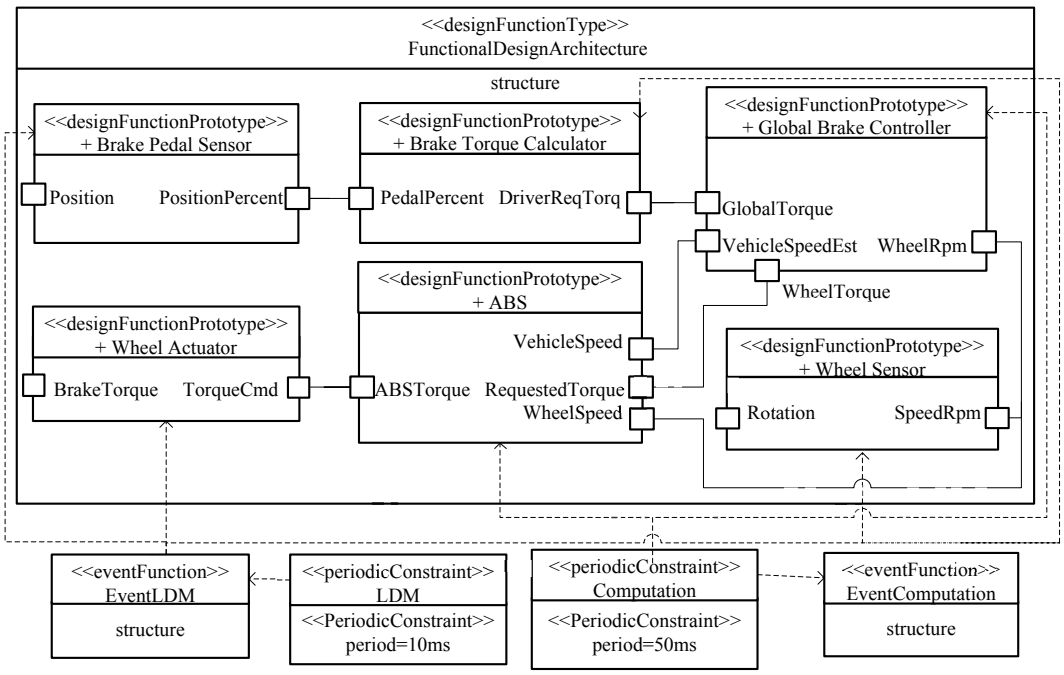

Figure 10. Brake by Wire control system.

global torque, and GBC that calculates the torque required for each wheel. The other four ECUs are connected to the four wheels, respectively. At each wheel, the WS measures the wheel speed and sends a signal to the GBC component. The ABS controls the wheel braking in order to prevent locking of the wheel, based on the slip value. The slip value is calculated by the equation:

$$slipRate = (vehicleSpeed - wheelSpeed \times r)/vehicleSpeed,$$

where *vehicleSpeed* is the estimated vehicle speed value, *wheelSpeed* the wheel speed sensor value, and *r* the wheel radius. The friction coefficient of the wheel has a nonlinear relationship with *slipRate*: when *slipRate* increases from zero, the friction coefficient also increases and the value reaches the peak when *slipRate* is around 0.2. After that, further increase in *slipRate* reduces the friction coefficient. For this reason, if *slipRate* is greater than 0.2 the brake actuator is released and no brake is applied, or else the requested brake torque is used. The system has been modeled in Papyrus UML Editor, where a UML profile is used for architectural description. We have modeled, simulated, and verified the BBW in our tool ViTAL, where we use only the structural and timing specifications. The architecture of the system is encapsulated in one FAA f_t that contains six interconnected f_ps modeled using the TA editor. Each TA(f_p) defines the actual functional and timing behavior of the f_p.

The slip rate calculation is controlled by variable `slipRate`. For instance, from location `calculateSlipRate`, based on the current vehicle speed `vSpeed`, the wheel speed `wSpeed`, and the wheel radius `wRadius`, the `TorqueCmd` controls the wheel braking in order to prevent locking of the wheel. Consequently, the ABS enters location `Torque` and jumps back to location `Start`, provided that `slipRate` is greater than 0.2, the brake actuator is released and no brake is applied, or else the requested brake torque is used.

6.2. Analysis and Verification

Different types of analysis can be used for functional and timing constraints in EAST-ADL. We demonstrate the new methodology through the BBW using the ViTAL tool. Requirements on the system design are expressed as

TCTL statements and can be verified by UPPAAL PORT. A list of requirements and the verification conditions are given below as a set of properties concerning the safety and liveness of the BBW. We discuss a few representative properties. The property of deadlock freedom is satisfied for all execution paths of the state-space.

Timing Verification: Our approach and the underlying tool support for verification of architectural properties supports the specification of timing properties in TCTL specification. As an example of such a property, which ensures the brake reaction delay specified in the BBW model, the following specification was verified:

$$A[\,](BBW.reaction\ imply\ (BBW.clock < 200))$$

We note that ViTAL automatically uses the delay constraints information from EAST-ADL models during verification. To handle automated integration during verification of TCTL properties of this type, this delay constraint information is considered a transformation parameter and then checked automatically in UPPAAL PORT. In order to verify *bounded response time properties* for the brake reaction delay, we use an observer BBW f_p with a certain syntactical manipulation on the system model. The BBW's input reaction from the BPS f_p and its output reaction from the WA f_p are modeled by the associated flow ports. This controllable model has no impact on the actual behavior of the system, but it interacts with the associated f_p to model the bounded response time. Similarly, in cases when other `DelayConstraint` requirements are specified, the interactions between the observer model and the constrained f_p s are updated in order to cope with the modified EAST-ADL timing information.

Functional Verification: One of the functional requirements of the system is related to the slip rate s. With ViTAL, we can verify the following functionality: in case the slip rate variable exceeds 0.2, the brake actuator is released and no brake is applied:

$$A[\,](BTC.slipRate > 0.2\ imply\ (ABS.brake == 0))$$

Other properties (*Lead-to properties* [1]) related to functional and timing requirements are verified and listed below:

- Based on the internal behavior every time the system runs it will eventually execute the ABS mode which triggers the calculation of the brake force when the brake pedal is activated:

$$(BPS.BrakePedal) \longrightarrow (GBC.mode == ABS \wedge GBC.BrakeForce)$$

- Based on the values of ports, if the brake pedal is activated then the value of its position should be received by the GBC f_p:

$$(BPS.BrakePedal) \longrightarrow (GBC.BrakeCtr \wedge GBC.in == 1)$$

Model-checking the properties requires an average of around 1.2 seconds per verified property (on an 2.4 GHz Intel Core i5).

7. Related Work

Several works on the formal analysis and verification of Architecture Analysis and Design Languages (AADLs), the main basis for behavioral verification of real-time embedded systems, have been carried out [20, 21, 22, 23]: An external behavior specification is used and verified by communicating TA [20] or source code [21]. To construct an

[1] Lead-to property: "whenever p holds q will eventually hold" is written "$p \longrightarrow q$" in UPPAAL queries

analyzable behavioral and architectural model, AADL enriched with its behavioral annex [22, 23] is considered the underlying model used for the verification. Berthomieu et al. [22] use AADL and its behavior annex into the Fiacre language towards behavioral verification with TINA tool. The main difference in our work is that we give a graphical behavior specification with support for model checking, system simulation, timing analysis, and an integrated tool based on Eclipse.

For safety-driven system development in the automotive domain, feature based analysis is prescribed by ISO standard as the state-of-the art approach to functional safety. However, at an early stage it is difficult to see function dependencies that would result in updated function requirements. A. Sandberg et al. [24] provide an approach that performs iterative analysis for managing changes in the safety architecture at analysis level and still meets function specific safety goals derived at vehicle level. In comparison to our work, their main concern is to define the semantics for requirement selection in order to ensure correct inclusion of requirements for a function definition. There is no formal modeling approach to the behavioral definition of the language.

Several methods have been developed for the formal analysis and verification of EAST-ADL models: F. Nallet et al. [25] describe the use of UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) [26] together with EAST-ADL for timing analysis [25]. In the context of EAST-ADL UML2 profile, Feng et al. [27] propose a translation of an existing behavioral principle of the system specified in activity diagrams into PROMELA and use the SPIN model checker for formal verification of EAST-ADL functional models. In contrast to our work, the integration of timing constraints in the behavioral model is not allowed and formal analysis on the real-time properties of the behavior model is not considered at all. Qureshi et al. [28] describe an integration effort towards formal verification of EAST-ADL models based on timing constraints, i.e., the EAST-ADL models are manually transformed into UPPAAL models that are then validated in terms of the timing- and triggering constraints. Despite their approach offering prototype support for model-checking reachability and safety properties corresponding to the timing constraints, it is limited in terms of model checking functional constraints and complex systems with respect to state space and time. The ViTAL tool, on the other hand, has this capability via the PORT model-checking technique. Refer to [7] for further information on the advantages of using a PORT technique.

In our previous work [10, 29] an integration effort towards formal modeling and analysis of EAST-ADL models based on timing constraints was investigated where the models were manually translated into TA formats in UPPAAL series tools [30]: Kang et al. [10] performed an earlier study towards verification of EAST-ADL models using UPPAAL PORT with the aim of identifying integration needs and the results were used as the basis of our approach in the current work. Furthermore, recent additions to EAST-ADL were considered in [29], including both timing and behavioral extensions in compositional and refinement reasoning and eliminated the need for intermediate formalism such as ICM and performed direct transformation of EAST-ADL specifications to an extended UPPAAL TIGA ECDAR [30]. Nevertheless, these early works did not support transformation in full automation, and in comparison to our current work, no implementation was available.

8. Conclusion and Future Work

In this work, we present the use of formal modeling and verification techniques at an early development stage of safety-critical automotive products which are originally described in the EAST-ADL architectural language. We employ the formalism of TA to facilitate the capture of the execution flow inside each functional block and the complex interactions between components mainly at *Analysis level*. We verify these constraints by using the UPPAAL PORT model checker.

The modeling and verification tool ViTAL provides an unambiguous behavioral description of EAST-ADL models, and brings formal verification capabilities to the EAST-ADL models. ViTAL enhances the behavioral definition of the EAST-ADL language and allows formal modeling, simulation, and verification of functional and timing requirements. With ViTAL, we have integrated such models, in order to be able to simulate and check whether a given requirement is satisfied, by model-checking the TA description with UPPAAL PORT. The independence introduced by the “run-to-completion" semantics, employed by the EAST-ADL functional modeling, is exploited by UPPAAL PORT, in order to reduce time and space requirements for model checking. Our technique improves behavior modeling, verification and analysis capability of EAST-ADL, and the result shows the applicability of model checking in safety-critical automotive products.

As future work, our research targets are: (i) richer transformation constructs to validate delay and synchronization constraints in fully automatic, (ii) the seamless integration of UML behavioral diagrams in particular activity diagrams or state machines as a specification formalism into ViTAL (to specify real-timing constraints on the UML level, we have investigated and defined an extended UML profile which integrates relevant concepts from EAST-ADL and MARTE profiles [31, 32]), and (iii) more elaborate verification of non-functional properties and more refined configurations of the generated model. For example, minimizing the use of certain resources, such as CPU, energy, memory, etc, while preserving functional correctness, timing requirements and other resource constraints. Our recent effort on modeling formal energy-aware real-time (ERT) behaviors in EAST-ADL has been initiated and an extended UML profile augmenting EAST-ADL and MARTE is presented to facilitate those behaviors by means of state machines [31].

Acknowledgments

This work was funded by FNRS (Fonds de la Recherche Scientifique) and TPA project in Belgium, ARTEMIS Joint Undertaking under grant agreement number 269335, from VINNOVA, the Swedish Governmental Agency for Innovation Systems, and Swedish Research Council, within the MBAT and ATAC projects. Special thanks to Henrik Lönn, Thomas Söderqvist, Daniel Karlsson from Volvo Technology and Lei Feng from KTH Royal Institute of Technology for their valuable feedback.

References

[1] M. Broy, Challenges in Automotive Software Engineering, in: Proceedings of the 28th international conference on Software Engineering, 2006, pp. 33–42.
[2] C. MAENAD, EAST-ADL Domain Model Specification, intermediate version Edition, http://www.maenad.eu/, 2011, pp. 2–143.
[3] P. Cuenot, P. Frey, R. Johansson, H. Lönn, Y. Papadopoulos, M.-O. Reiser, A. Sandberg, D. Servat, R. Tavakoli Kolagari, M. Törngren, M. Weber, The EAST-ADL Architecture Description Language for Automotive Embedded Software, in: Model-Based Engineering of Embedded Real-Time Systems, Lecture Notes in Computer Science, Springer, 2011, pp. 297–307.
[4] T. A. ATESST2 Consortium, EAST-ADL Profile Specification, 2.1 RC3 (Release Candidate), 2nd Edition, www.atesst.org, 2010, pp. 10–75.
[5] J. Rumbaugh, I. Jacobson, United Modeling Language User Guide, 2nd Edition, Addison-Wesley, 1998.
[6] P. Cuenot, S. Gerard, H. Lonn, M. Reiser, D. Servat, C. Sjostedt, R. Kolagari, M. Torngren, M. Weber, et al., Managing Complexity of Automotive Electronics using the EAST-ADL, in: 12th IEEE International Conference on Engineering Complex Computer Systems, 2007, pp. 353–358.
[7] J. Håkansson, P. Pettersson, Partial Order Reduction for Verification of Real-time Components, in: Proceedings of the 5th International Conference on Formal Modeling and Analysis of Timed systems, Springer-Verlag, 2007, pp. 211–226.
[8] E. Enoiu, R. Marinescu, C. Seceleanu, P. Pettersson, ViTAL: A Verification Tool for EAST-ADL Models Using UPPAAL PORT, in: 17th International Conference on Engineering of Complex Computer Systems, IEEE, 2012, pp. 328–337.
[9] R. Alur, D. L. Dill, A Theory of Timed Automata, in: Theoretical Computer Science, Vol. 126, 1994, pp. 183 – 235.
[10] E.-Y. Kang, P. Y. Schobbens, P. Pettersson, Verifying Functional Behaviors of Automotive Products in EAST-ADL2 using UPPAAL-PORT, in: Proceedings of the 30th International Conference on Computer Safety, Reliability and Security, Springer-Verlag, 2011.
[11] J. Bengtsson, B. Jonsson, J. Lilius, W. Yi, Partial Order Reductions for Timed Systems, in: Lecture Notes in Computer Science: Concurrency Theory, Vol. 1466, Springer Berlin / Heidelberg, 1998, pp. 485–500.
[12] P. Godefroid, P. Wolper, Using Partial Orders for the Efficient Verification of Deadlock Freedom and Safety Properties, in: Lecture Notes in Computer Science: Computer Aided Verification, Springer Berlin / Heidelberg, 1992, pp. 332–342.
[13] J. Carlson, J. Håkansson, P. Pettersson, SaveCCM: An Analysable Component Model for Real-Time Systems, in: Proceedings of the 2nd Workshop on Formal Aspects of Components Software, Vol. 160, Elsevier, 2006, pp. 127–140.
[14] C. ARTEMIS MBAT, Consortium Public Deliverables (Nov. 2013).
URL `www.mbat-artemis.eu/`
[15] J. Suryadevara, E.-Y. Kang, C. Seceleanu, P. Pettersson, Bridging the Semantic Gap between Abstract Models of Embedded Systems, in: 13th International Symposium on Component Based Software Engineering, Springer, 2010.
[16] T. A. ATESST2 Consortium, Evaluation Report EAST-ADL2 Behavior Support, D2.1 Appendix A3.4, d2.1 appendix a3.4 Edition, www.atesst.org, 2010.
[17] M. Åkerholm, J. Carlson, J. Håkansson, H. Hansson, M. Nolin, T. Nolte, P. Pettersson, The SaveCCM Language Reference Manual, Technical Report MDH-MRTC-207/2007-1-SE, Mälardalen University (January 2007).
[18] F. Jouault, I. Kurtev, Transforming Models with ATL, in: Satellite Events at the MoDELS 2005 Conference, Springer, 2006, pp. 128–138.
[19] J. Håkansson, J. Carlson, A. Monot, P. Pettersson, Component-Based Design and Analysis of Embedded Systems with UPPAAL PORT, in: 6th International Symposium on Automated Technology for Verification and Analysis, Springer-Verlag, 2008, pp. 252–257.
[20] T. Abdoul, J. Champeau, P. Dhaussy, P.-Y. Pillain, J.-C. Roger, AADL Execution Semantics Transformation for Formal Verification, in: 13th IEEE International Conference on Engineering of Complex Computer Systems, 2008, pp. 263 –268.
[21] J. Hugues, B. Zalila, L. Pautet, F. Kordon, From the Prototype to the Final Embedded System using the Ocarina AADL Tool Suite, in: Transactions in Embedded Computing Systems, Vol. 7, ACM, 2008, pp. 1–42.

[22] B. Berthomieu, J.-P. Bodeveix, S. Dal Zilio, P. Dissaux, M. Filali, S. Heim, P. Gaufillet, F. Vernadat, Formal Verification of AADL models with FIACRE and TINA, in: 5th International Congress and Exhibition on Embedded Real-Time Software and Systems, 2010.
[23] S. Björnander, C. Seceleanu, K. Lundqvist, P. Pettersson, ABV Verifier for the Architecture Analysis and Design Language (AADL), in: International Workshop on UML and AADL, 2011.
[24] A. Sandberg, D. Chen, H. Lönn, R. Johansson, L. Feng, M. Törngren, S. Torchiaro, R. Tavakoli-Kolagari, A. Abele, Model-based Safety Engineering of Interdependent Functions in Automotive Vehicles using EAST-ADL2, in: Proceedings of the 29th International Conference on Computer Safety, Reliability, and Security, Springer-Verlag, 2010, pp. 332–346.
[25] F. Mallet, M.-A. Peraldi-Frati, C. Andre, Marte CCSL to Execute East-ADL Timing Requirements, in: IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, 2009, pp. 249 –253.
[26] C. André, F. Mallet, R. De Simone, Modeling of Immediate vs. Delayed Data Communications: from AADL to UML MARTE, in: The Forum on specification & Design Languages, ECSI, 2007, pp. 249–254.
[27] L. Feng, D. Chen, H. Lönn, M. Torngren, Verifying System Behaviors in EAST-ADL2 with the SPIN Model Checker, in: International Conference on Mechatronics and Automation, 2010, pp. 144 –149.
[28] T. Naseer Qureshi, D. Chen, M. Persson, M. Törngren, Towards the Integration of UPPAAL for Formal Verification of EAST-ADL Timing Constraint Specification, in: The International Workshop on Model-Based Design with a Focus on Extra-Functional Properties, 2011.
[29] E.-Y. Kang, P.-Y. Schobbens, A. Legay, Verification of component-based architectural models on autonomous truck systems, in: Proceedings of the 9th International Conference and Workshops on the Engineering of Autonomic and Autonomous Systems, IEEE Computer Society, 2011.
[30] ECDAR: Environment for Compositional Design and Analysis of Real-time systems, http://ecdar.cs.aau.dk/ (2010).
[31] E.-Y. Kang, G. Perrouin, P.-Y. Schobbens, Towards Formal Energy and Time Aware Behaviors in EAST-ADL: An MDE Approach, in: Proceedings of the 12th International Conference on Quality Software, IEEE Computer Society, 2012.
[32] E.-Y. Kang, G. Perrouin, P.-Y. Schobbens, XFG language and its profile for modeling and analysis of energy-aware and real-timed behaviors, in: Technical Report, 2012.