Verifying Functional Behaviors of Automotive Products in EAST-ADL2 Using UPPAAL-PORT

Eun-Young Kang[1,2], Pierre-Yves Schobbens[1], and Paul Pettersson[2]

[1] Computer Science Faculty, University of Namur, Belgium
[2] MDH PROGRESS Research Centre, Västerås, Sweden
{eun-young.kang,pierre-yves.schobbens}@fundp.ac.be
paul.pettersson@mdh.se

Abstract. We study the use of formal modeling and verification techniques at an early stage in the development of safety-critical automotive products which are originally described in the domain specific architectural language EAST-ADL2. This architectural language only focuses on the structural definition of functional blocks. However, the behavior inside each functional block is not specified and that limits formal modeling and analysis of systems behaviors as well as efficient verification of safety properties. In this paper, we tackle this problem by proposing one modeling approach, which formally captures the behavioral execution inside each functional block and their interactions, and helps to improve the formal modeling and verification capability of EAST-ADL2: the behavior of each elementary function of EAST-ADL2 is specified in UPPAAL Timed Automata. The formal syntax and semantics are defined in order to specify the behavior model inside EAST-ADL2 and their interactions. A composition of the functional behaviors is considered a network of Timed Automata that enables us to verify behaviors of the entire system using the UPPAAL model checker. The method has been demonstrated by verifying the safety of the Brake-by-wire system design.

1 Introduction and Main Themes

EAST-ADL2 is an architecture description language for the development of automotive embedded systems [1]. Advanced automotive functions [15,6] are increasingly dependent on software and electronics. These automotive embedded systems are becoming progressively complex and critical for the entire vehicle. Model-based development (MBD) is a means to manage this complexity and develop embedded systems in a way that increases safety and quality. The EAST-ADL2 modeling approach addresses this topic and provides means to integrate the engineering information from documents, spreadsheets and legacy tools into one systematic structure, an EAST-ADL2 system model.

Our aim is to use formal modeling techniques at an early stage in the development life cycle of automotive embedded systems, and to use symbolic simulators and model checkers as debugging and verification tools to ensure that the predicted function behaviors of the modeled system in EAST-ADL2 satisfy certain requirements under given assumptions on the environment where the system is supposed to operate.

EAST-ADL2 expresses the structure and interconnection of the system. System behavior is defined based on the definition of a set of elementary functional blocks and

F. Flammini, S. Bologna, and V. Vittorini (Eds.): SAFECOMP 2011, LNCS 6894, pp. 243–256, 2011.

their triggers and interfaces. However, the behavioral definition inside each elementary functional block is not specified, which limits the automatic translation from EAST-ADL2 models to other formal models for efficient verification. Instead, the execution of each function is described with external behavioral annexes and legacy tools including general UML tool and domain-specific tools, e.g., Simulink or UML [13]. This restricts the construction of a complete system behavior model and verification of the behavior of the entire system model with verification tools.

To achieve our goal by improving the aforementioned restriction, we propose a formal approach which facilitates the verification of system function behaviors in EAST-ADL2 by using UPPAAL-PORT model-checker [8]: this approach specifies a behavior inside of each elementary function (block) in Timed Automata (TA) and constructs a complete system behavior model by the parallel composition of local behaviors. In particular, we specify the execution of each function behavior in the UPPAAL-PORT TA model and consider a composition of the function behaviors as a network of TA so that the behaviors of the entire system in EAST-ADL2 can be formally defined. Then this network TA can be analyzed and verified by UPPAAL-PORT model checker.

This work is organized as follows. Section 2 introduces technology and background, EAST-ALD2 and UPPAAL-PORT toolkit as used in our approach. Section 3 presents our approach for verifying system behaviors in EAST-ADL2 by using UPPAAL-PORT model checker: this approach formally captures the behavior inside each functional block and their parallel compositional interactions. Furthermore, the formal definition enables transformation of the given model to models of UPPAAL-PORT tool for model checking. In section 4, our method is demonstrated in verifying the safety of the Brake-by-wire system design. We discuss further work and conclude in Section 5.

2 Background

2.1 EAST-ADL2

The goals of modeling with EAST-ADL2 are to deal with complexity control and improve safety, reliability, cost, and development efficiency through MBD. For this, EAST-ADL2 structures a system model into multiple abstraction levels in terms of the development life cycle of automotive embedded systems.

EAST-ADL2 is an information model, connecting different views of the system. The views are influenced by the different engineering traditions and backgrounds. This concept allows EAST-ADL2 to handle various types of information including requirements, vehicle features, system environment, application functions, deployment of software and hardware resources, behaviors, non-functionality properties such as variability, timing constraint, dependability, and V&V related information. Abstract solution, design, and implementation details are found in different abstraction levels in the model: the highest abstraction level, *Vehicle(Feature) level*, characterizes a vehicle by means of features and defines implementation-independent information such as features and requirements. Fig. 1 depicts an overview of the system model and the abstraction levels of EAST-ADL2.

At *Analysis level*, functionality is realized based on the features and requirements. These features and requirements are refined by the decision of logical design with the

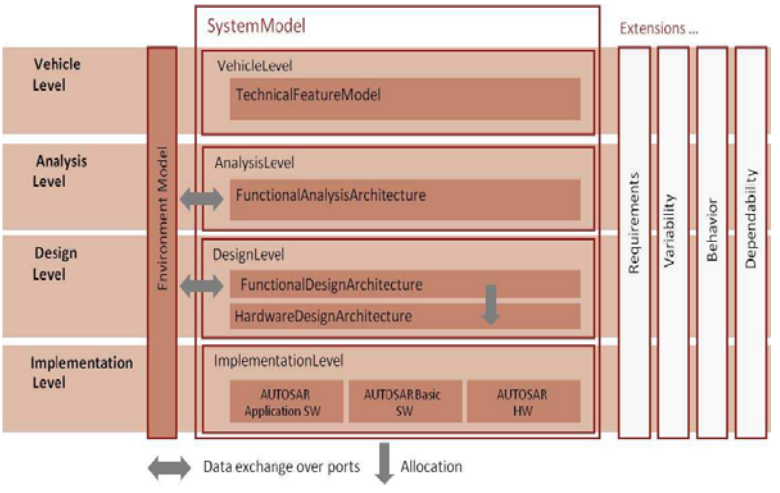

Fig. 1. The structure of an EAST-ADL2 System Model

definition of logical abstract functions of features and their interactions, and requirements. The model at this level is used for the analysis of control requirements, timing constraints, data consistency between interfaces, hazard identification, etc. *Design level* contains concrete functional definition according to the realized logical design. In particular, functional definition of application software, functional abstraction of hardware and middleware are presented, as well as hardware architecture being captured and function-to-hardware allocation being defined. *Implementation level*, i.e. the software architecture, is represented using the AUTOSAR standard and allocates software modules to a network of Electronic Control Units (ECUs) according to the AUTOSAR standard [2]. As in Fig. 1, EAST-ADL2 extensions are constructs for requirements, variability, behaviors, dependability, and V&V activities, etc. EAST-ADL2 is intended to be an integration framework for functionality defined in different notations and tools. The behavioral definition therefore relies on the definition of a set of elementary functions that are executed based on the assumption of run-to-completion execution (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 behavioral notations. Details of those issues are explained in Section 3.

2.2 UPPAAL-PORT

UPPAAL-PORT is a model checking tool for component based modeling, simulation, and verification of real-time and embedded systems modeled as real-time components. It can be used as an Eclipse plugin together with the SAVE integrated development environment (IDE)[16,17] in order to support graphical modeling of internal component behaviors as an UPPAAL-PORT TA and composition of components. 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

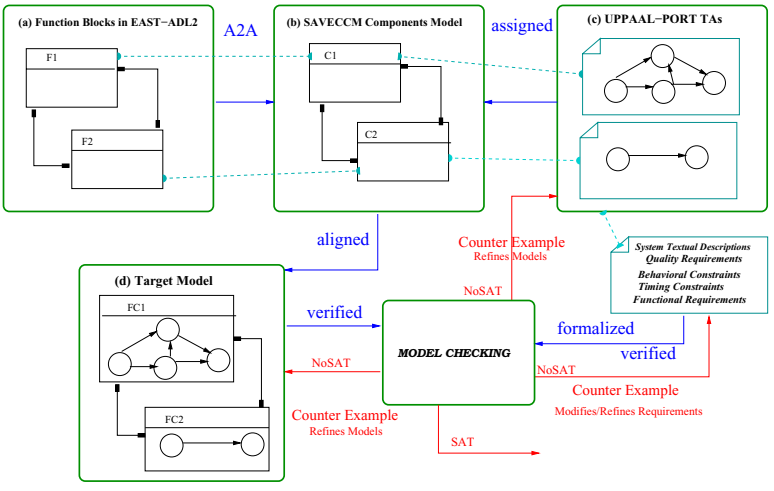

Fig. 2. Methodology Roadmap

modeling language, SAVE-CCM [3], which describes the architectural framework for modeling real time embedded applications with particular emphasis on automotive domain and safety concerns. In particular, SAVE-CCM is used to create components and interconnections among them, and supports run-to-completion semantics. We use this architectural framework of SAVE-CCM for mapping from functional blocks and their interconnectors in EAST-ADL2 to components and their interconnections in SAVE-CCM respectively.

For analysis purpose, an UPPAAL-PORT TA model is assigned to each of the SAVE-CCM components in order to describe timing and functional behaviors of the component. Since EAST-ADL2 is intended for use with different behavioral notations, UPPAAL-PORT TA is perfectly appropriate to use. This UPPAAL-PORT TA communicates with other ones through ports and the values of the ports defined by binding TA variables to the ports of components, and supports synchronous execution with other regular UPPAAL TA models. Thus, by defining local behaviors of each EAST-ADL2 function block with UPPAAL-PORT TAs, their synchronous run-to-completion execution semantics should make it possible to integrate the contained TA into a model representing the complete system. Since EAST-ADL2 also supports requirements, the invariants and other logical criteria used for modeling function behaviors with UPPAAL-PORT TAs refer to requirements in EAST-ADL2. The SAVE-CCM/UPPAAL-PORT is not an intermediate step. It is the target model we want to build as a long term goal. So the behaviors of a given system (functions and their interactions) will be more effectively analyzed.

3 Approach and Proposed Solution

To achieve our aforementioned goal in section 1, we propose a formal approach which facilitates the verification of system behaviors in EAST-ADL2 by using UPPAAL-PORT model-checker independently of any hardware constraints and topology mapping. It

mainly focuses on the higher level of functional behavior of applications at *Analysis level* in terms of its *Feature level* with three distinct phases – architectural and behavioral mapping, behavior specification, and verification (model checking). We will discuss those phases in more detail in following sections.

3.1 Architecture and Behavioral Semantics Mapping

This architectural mapping step, called A2A, is an architecture-to-architecture representation from (a) to (b) in our methodology roadmap Fig.2. The EAST-ADL2 model architecture frame at *Analysis level* in the Papyrus UML (Fig.2-(a)) is mapped to SAVE-CCM architecture frame (Fig.2-(b)). However, this stage is not concerned with the actual representation of the data.

This stage performs a semantic anchoring between the domains of EAST-ADL2 and that of SAVE-CCM. The purpose of the semantic anchoring is to map concepts from EAST-ADL2 to SAVE-CCM in a way that preserves the semantics of the original model without changing the structure of the model heavily. Each elementary *AnalysisFunction* (AF) has its own logical execution and no internal concurrency, therefore it maps well to a SAVE-CCM component. In this case, there is a convenient and obvious mapping possibility: we design a system model in SAVE-CCM from the given system at *Analysis level* in EAST-ADL2. We assign one SAVE-CCM component per AF element in the EAST-ADL2 system (i.e. BreakController, ABS, etc). The original interconnectors and associated ports in EAST-ADL2 are mapped to the interconnections between ports of SAVE-CCM components respectively and that enables communication with other components according to their original AF element. One or more ports and elements of EAST-ADL2 models may be realized by one port and one component of SAVE-CCM models, as these may have several signals or data elements per interface that are simplified (as abstracted design) in one port and one component in our SAVE-CCM design model.

Inside each AF, the data transformation and its own behaviors are described as TAs based on the assumption of synchronous run-to-completion execution. There are two types of function interactions in EAST-ADL2: either a *FlowPort* interaction whereby a function performs a computation on provided data, or a *ClientServer* interaction whereby the execution of a service is called upon by another function. The *FlowPort* interactions are matched to the interconnections of SAVE-CCM components. The *ClientServer* interactions are explained by the execution of the TA inside a component and its synchronization with other TAs.

The triggering of each AF is defined either as time-driven or event-driven on one of the input ports. There are two types of function entities, time-discrete function and time-continuous function. Time-discrete function is done after a computational delay, i.e. execution time. Time-continuous function defines the transfer function from input to output, and the computation rate is infinite. Since the semantic of AF is run-to-completion, there should be no infinite delays in the local UPPAAL-TA model, the time-continuous function is not concerned in our semantic anchoring and we deal only with time-discrete function. The time-discrete function is invoked either by time-triggered in which time alone causes execution to start, or event-triggered, which is caused by data arrival or calls on the input ports. Those trigger conditions are matched to those of a *clock* component, trigger ports and data type ports in SAVE-CCM, respectively.

3.2 Behavior Specification

We have shown a straightforward mapping from the informal semantics of EAST-ADL2 to the formal semantics of SAVE-CCM. Since EAST-ADL2 allows the use of different behavioral notations, we capitalize on this advantage to specify *FunctionBehavior* by assigning an UPPAAL-PORT TA model to each SAVE-CCM component mapped from its corresponding AF (especially *ADLFunctionPrototype*) respecting the triggering definition, and execution time of each AF as well as its requirements. This TA model encapsulates the "execution behaviors" of AF and is used for verification in terms of real-time properties by using UPPAAL-PORT model-checker. Our tooling composes such local automata in parallel to a composed TA (network TA). The purpose of this phase is to construct a target model (Fig.2-(d)) by filling the architectural frame model (Fig.2-(b)) with the corresponding UPPAAL-PORT TAs (Fig.2-(c)). In this case, textual system description, quality/functional requirements and behavioral/timing constraints are referred to specify *FunctionBehavior* in UPPAAL-PORT TAs (Fig.2-(c)).

We define an EAST-ADL2 model below. Essentially, this model is a tuple $\langle N, CE \rangle$, where N is a set of *ADLFunctionalPrototypes* AFs, and $CE \subseteq N \times N$ is the set of interconnectors between AFs. Output variables of one AF may be connected to input variables of another AF. The clock component in SAVE-CCM [18] is used to define time-triggered activations. It periodically generates the triggering event to activate the component and its connected components in a sequence by sending a trigger signals through the ports. For detailed semantics of the SAVE-CCM language (the subset of ProSave), we refer the reader to [18]. These triggering (or data) signals arrive at a port with a one-place buffer. It is stored in that buffer, and for other ports it is forwarded to connected ports.

The behavior $\mathscr{B}$ inside an AF, noted $[\![B]\!]_{AF}$, is modeled as an UPPAAL-PORT TA $= \langle L, l_0, l_f, V_C, V_D, E, I \rangle$, where L is a set of locations, $l_0 \in L$ is the initial location, $l_f \in L$ is the final location, such that no edges in E are leading out from l_f, and is used to model the termination of an execution of AF. V_C and V_D is a set of clock and data variables respectively. I assigns an invariant to each of the locations. E is a set of edges, represented as $l \xrightarrow{g,a,u} l'$, where l is a source location, l' is a destination location, g is a guard, a is an action, u is an update.

The execution of behavior inside FA is determined, (i.e, a SAVE-CCM component is triggered) in terms of triggering values, which can be generated from the clock component of SAVE-CCM (named *active*). When the triggering value is *active*, the component is triggered via its input trigger port and its input data ports are mapped to data variables. V_D in TA are updated with those variables by *read-input-from-ports* action, noted READ(P_{in}), (respectively *write-output-to-ports*, noted WRITE(P_{out})), 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* after the read action it switches to its executing locations until its internal computation is done. After the write action, which forwards data in variables via interconnections from the output ports, the component becomes *idle* again and the trigger port is updated to *inactive*. Formally the behavior of AF is defined as follows:

Definition 1 (Behavior of AF). *The behavior of AnalysisFunction AF is a tuple* $\mathscr{B} = \langle L \cup \{l_\perp\}, l_\perp, l_0, l_f, V_D \cup P, V_C, E \cup \{e_r, e_w\}, I_\perp \rangle$ *where*

- $l_\perp$ *is the idle location.*
- *P is the set of ports of the component described as* $P_{in} \cup P_{out} \cup P_{trig}$*, where* P_{in} *is a set of input ports,* P_{out} *is a set of output ports,* $P_{trig} \subseteq P_{in}$ *is the set of trigger input ports.*
- $e_r = l_\perp \xrightarrow{g,r,u} l_0$*, if g is triggered, r is the "read-input-from-ports" action, READ(*P_{in}*), and u updates* V_D *with input values* $(P_{in} \backslash P_{trig})$.
- $e_w = l_f \xrightarrow{g,w,u} l_\perp$*, if g is true, w is the "write-outputs-on-ports" action, WRITE(*P_{out}*), and u resets* P_{trig} *to "inactive"*
- $I_\perp(l_\perp) = true$*,* $I_\perp = I(l)$ *for* $l \neq l_\perp$

The TA of a composition C, TA(C), is defined as a network of local TA. For AF_i and its corresponding component $C_i \in C$, the write action in TA(C_i) is extended to update the input ports (noted $P_{in.j}$) of a target component $C_j \in C$ according to interconnections from the out ports of C_i (noted $P_{out.i}$). An interconnection connects a source port $p \in P_{out.i}$ to a target port $p' \in P_{in.j}$ whenever variables in P_{in} of C are enabled in a way that if p' is a trigger port then p' is activated, otherwise $p' = p$. The edges e of TA(C_i) are explained with extended write actions as follows.

Definition 2 (Extended Write Actions). *The behavior* $\mathscr{B}$ *inside* AF_i*,* $[\![B]\!]_{AF_i}$*, is TA(*C_i*)* $= \langle L, l_0, l_f, V_D, V_C, \{XWRITE_i(e) \mid e \in E\}, I \rangle$ *such that*

- $XWRITE_i\ (l \xrightarrow{g,a,u} l') \triangleq (l \xrightarrow{g,w,u} l'\ ;\ WRITE(P_{out.i}))$*, if* $a = w$ *and g is triggered (holds). Note that* ; *is defined as sequential execution*
- $XWRITE_i\ (l \xrightarrow{g,a,u} l') \triangleq l \xrightarrow{g,a,u} l'$*, for* $a \neq w$

The automata TA(C) is then the network of each 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 adaption of the $XWRITE$ action that is used to build $TA(C_i)$ in Definition 2.

3.3 Verification: Model Checking

The execution of each *FunctionBehavior* in EAST-ADL2 is specified by UPPAAL-PORT TA in SAVE-CCM and its composition is considered as the network TA: the formal semantics of SAVE-CCM used in this paper was given in section 3.2. For the semantics of the full SAVE-CCM language, we refer the reader to [7]. The entire system (network TA) is considered in terms of a timed transition system [18], then this entire system is verified by UPPAAL-PORT model-checker. Quality requirements (e.g, timing, safety, deadlock freedom) in terms of functional requirements (e.g, behavioral constrains, timing constraints), see Fig.2-Requirements aspect, are formalized in linear time logics based on the UPPAAL logic, which can be verified over the target model (Fig.2-(d)) by UPPAAL-PORT model checker.

In particular, the quality requirements are derived from a given system's textual descriptions. One may verify certain delay, reaction and synchronization constraints (i.e, overall behavioral constraints of a system) according to the quality requirements. For

example, a plausible reaction constraint is 250 ms. In contrast, functional requirements describe particular constraints of a function such as timing constraints and trigger elements linked to an AF block. They define the triggering and execution time of the AF. UPPAAL-PORT model checker verifies those two types of requirements as safety properties in a way that (a) if a property is satisfied by the target model, then a functional requirement linked to an AF is updated to a satisfy relation and generic constraints of the AF are stored as valid invariants in the V&V structure (VVOutcome linked to the explained requirement, VVCase, etc) of the EAST-ADL2 model. (b) If a property is violated (depicted as NoSAT arrows in Fig.2) then our UPPAAL-PORT model checker returns some counterexamples that can help analysts to refine the behavioral constraints of the system model or modify generic constraints, and identify correct constraints for the AF that it concerns. Thus, the models in EAST-ADL2 are updated with the timing assumptions analysts make as well as the analysis results.

4 Current Result and Example

Our approach has been applied and demonstrated on a case study, the Break-by-Wire System (BWS), from our industrial partner VOLVO. It has been first modeled using Papyrus UML [11] for EAST-ADL2 in the ATESST2 project [1]. First, the BWS Papyrus UML model at *Analysis level* in EAST-ADL2 domain is translated to a SAVE-CCM model. This step is depicted in Fig. 2-(a) and the result is shown in Fig. 3.

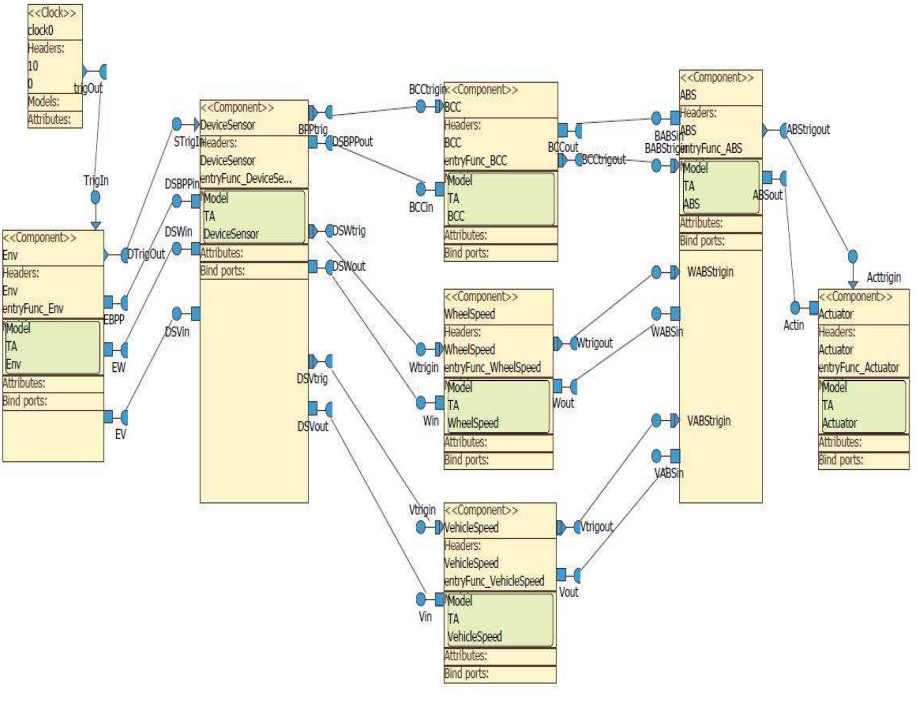

Fig. 3. The BWS architectural model (actual screenshot from the SAVE-CCM modeling tool)

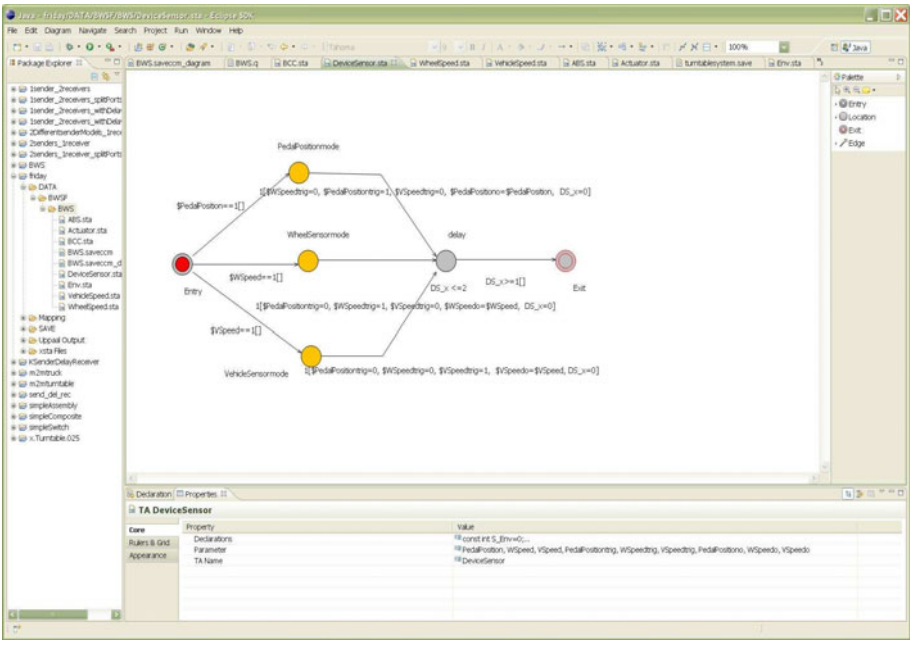

Fig. 4. BWS DeviceSensor TA

Secondly, *FunctionBehaviors* are specified in UPPAAL-PORT TAs and then are assigned to components in the SAVE-CCM model in terms of their corresponding function entities in EAST-ADL2. The functional and quality requirements of the system were given as either informal description in ATESST2 project case study reports or as timing/trigger constraints requirement entities linked to AFs in EAST-ADL2. The architecture of the system is represented by SAVE-CCM components filled with UPPAAL-PORT TAs. One of the TAs in this step is shown in Fig.4.

Finally, the requirements formalized in UPPAAL logics over the result from the second step are verified by model-checking with some assumptions we make regarding timing: there is a data flow from a pedal to a brake actuator. The functions are periodic and mutually unsynchronized. A perfect clock is assumed in the sense that it generates periodic triggering in order to activate (run) the components with a periodicity of one time unit. Each function has its execution time which is modeled with a delay location in its TA. Based on those assumptions, properties of safety, deadlock freedom and liveness are verified successfully.

Data flows through ports between function blocks of BWS are simulated by using the UPPAAL-PORT plug-in for the Eclipse IDE in Fig.5. The direction of data flow is indicated by the arrow. We use this simulator in order to trace or detect fault flow paths. This is facilitated by its intuitive graphical interface that allows analysts to step forward and backward along the simulation. Apart from the simulation, we have so far verified 28 properties of the system. A list of selected properties is given below and their verification results are established as valid:

$\backslash * \; Definition\ of\ each\ component$
$C1 = Environment$
$C2 = DeviceSensor$
$C3 = BCC : Brake\ calculator\ and\ controller$
$C4 = WheelSpeed$
$C5 = VehicleSpeed$
$C6 = ABS : Anti\ lock\ Braking\ System$
$C7 = Actuator * \backslash$

- *Deadlock freedom*: $A[\,]$ [1] not deadlock
- *Leads-to property* based on the internal variables of function components: every time the system is invoked by its environment it will eventually execute ABS which calculates the brake force according to the brake pedal position, wheel speed, vehicle speed:
  - $(C1.WheelDynamic \vee C1.BrakePedal \vee C1.VehicleDynamic) \rightarrow (C6.mode == ABS \wedge C6.ForceCtr)$
- *Leads-to property* based on the values of ports: if the BrakePedal function device sends out a value of its position then the value should be received by the BrakeController function:
  - $(C1.BrakePedal \wedge C1.EBPP == 1) \rightarrow (C3.BrakeCtr \wedge C3.BCCin == 1)$
- State correspondence check: one internal state of a component corresponds to what is happening in the states of other environment components. The following three properties describe that while one of the function components, BCC, WheelSensor, VehicleSensor is executing, the other two function components are not executing:
  - $A[\,]\ C5.VSensormode \implies (\neg\, C4.WSensormode \wedge \neg\, C3.BrakeCal)$
  - $A[\,]\ C4.WSensormode \implies (\neg\, C3.BrakeCal \wedge \neg\, C5.VSensormode)$
  - $A[\,]\ C3.BrakeCal \implies (\neg\, C5.VSensormode \wedge \neg\, C4.WSensormode)$
- Execution time property: each function component should execute within its given local execution time ($t = 2$), $0 \leq clock \leq 2$. In other words, it should not exceed its given local execution time:
  - $A[\,]\ C7.exec \implies (C7.clock \leq 2 \wedge C7.clock \geq 0)$

Since the current version of UPPAAL tool only provides reachability analysis, we first verified a certain delay in an AF component, such as its local execution time, as an invariant property. In order to verify *bounded response time properties* formula of the form $f1 \rightarrow_{\leq T} f2$, meaning *if a request (f1) becomes true at a certain time point, a response (f2) must be guaranteed to be true within a time bound (T)*, we apply the early experiments in [9], which showed how to check such properties with a certain syntactical manipulation on the system model, to our work either by (1) adding observer components syntactically to the system model or (2) making observer automata and synchronizing them with the actual system automata. Then we verify if both observers success states can be reached in parallel with the main actual system under the synchronization constraints.

[1] A[ ] P: "P holds for any reachable configuration" is written $A[\,]$ in UPPAAL format.

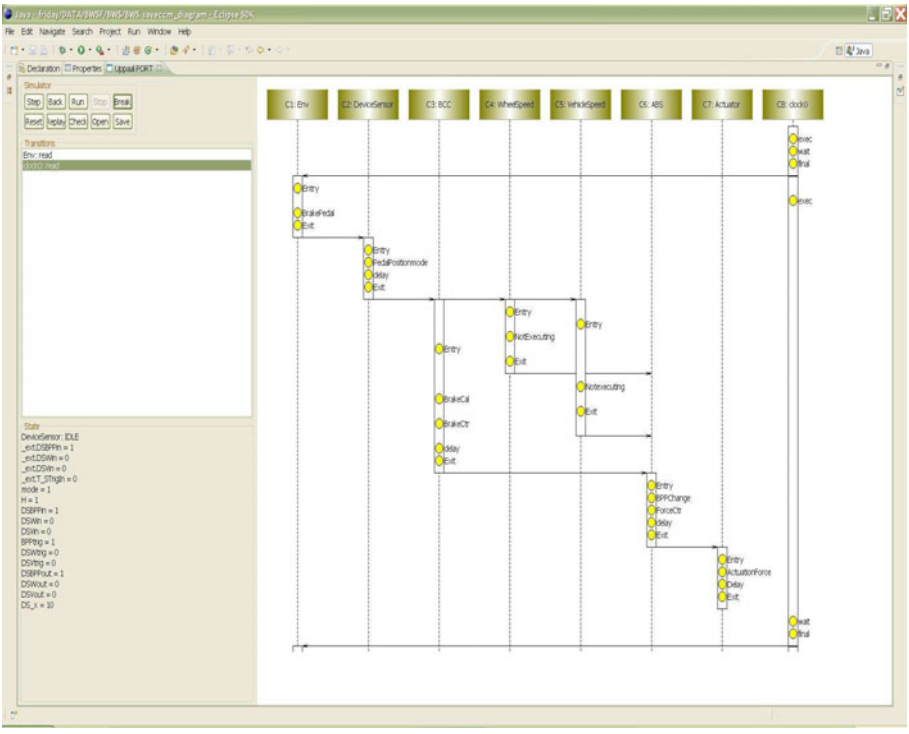

Fig. 5. BWS data flow simulation trace using UPPAAL-PORT

We construct one observer TA, illustrated in Fig 6, which contains an observer clock constraint (obsClock) as an invariant. This observer restricts the time bound of response time (MAX_TIME). By applying this observer TA in our experiment, we successfully evaluate bounded response time properties in a way that the error location, which violates the bounded time condition, is never reached from any location of the main actual system model. The verification result is given below:

- When the brake pedal mode is activated, the actuator reacts timely under its given time bound (MAX_TIME) as a failsafe against serious accident. i.e.,
  - $A[\,]\ C1.BrakePedal \implies (\neg\ ObsTA.error \wedge C7.Actuator)$. The property is valid. In other words, if the BrakePedal function component is invoked, it should not reach the error location of the observer TA, which violates the MAX_TIME bounded time condition, while the Actuator component is executing.

Search order is breadth first and uses conservative space optimization. The state space representation uses difference bound matrices (DBM). Verifying properties takes an average of around 2 seconds per verified property on an Intel T9600 2.80 GHz processor. The verification tool only needs to explore a maximum of 3584 states to verify properties such as deadlock freedom.

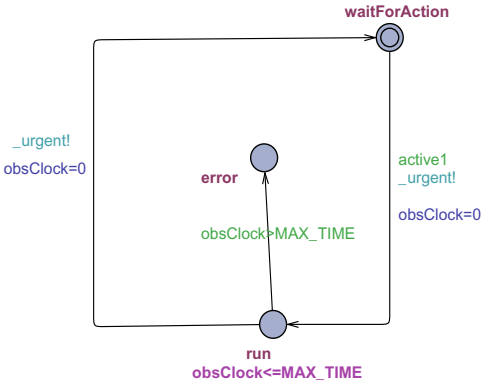

Fig. 6. Observer TA of bounded response time properties

5 Related Work

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 early stage it is difficult to see function dependencies that would result in updated function requirements. Therefore, A. Sandberg et al. [14] provide one approach that performs iterative analysis to manage changes in the safety architecture at analysis level and still meet 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.

L. Feng et al. [5] bring modeling formalisms to the existing behavioral principle of the system by transforming EAST-ADL2 behavior model to the SPIN model. Thus the requirements on the system design can be verified by model checking. In contrast to our work, there is no notion for the timing constraints in the behavior model. Indeed, formal analysis on the real-time properties of the behavior model is not considered at all.

6 Conclusion and Future Work

In this paper, we studied the use of formal modeling and verification techniques at an early stage in the development of safety-critical automotive products which are originally described in the EAST-ADL2 architectural language. While EAST-ADL2 focuses on the structural definition of functional block, we propose a method to formally specify behaviors inside each functional block in TAs mainly at *Analysis level* in terms of *Feature level* in EAST-ADL2, and verify them by using the UPPAAL-PORT model checker. The formal syntax and semantics of functional behaviors are defined. A composition of those behaviors is considered as a network of TA that allows us to verify the entire system using the UPPAAL-PORT model checker. Moreover, this paper presents a technique to verify *bounded response time properties* by adding observer components

or TA syntactically to the system model and synchronize them in parallel with the actual system automata. The contribution improves behavior modeling, verification and analysis capability of EAST-ADL2, and the result shows the applicability of model checking in safety-critical automotive products. We started from the informal description of quality and functional requirements in order to model the execution behaviors of *ADLFunctions* and manually specified them in TAs. A possible further work would be to define a formal, real-time semantics of UML diagrams so that engineers can use this familiar language. They can then be translated automatically to TAs.

In future works, we plan to extend our work: (1) From tooling perspective, in Papyrus (an Eclipse based tool platform for EAST-ADL2), UML modeling tools and domain-specific tools, e.g. Simulink, are used as external tools. They describe the data transformation inside each AF and exchange information via Plugins. Thus, an UPPAAL-PORT, which is in fact also an Eclipse based Plugin tool, would be developed as an EAST-ADL2 Plugin and be integrated with other EAST-ADL2 tools for analysis. The analysis invariants and outcome should be recorded in the EAST-ADL2 structure as valid constraints for requirements and V&V. Ideally this process should be done fully automatically. (2) Another future work encludes more elaborated 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. The results presented here are promising steps towards these goals. (3) Since the current UPPAAL tool only provides reachability analysis, observer TAs were used to verify *bounded response time properties* in this work. In order to extend this restriction, Memory Event Clocks Temporal Logic (MECTL) formula, created in our early work [12,10], will be adapted to improve fully decidable real-time expressiveness, using a tool chain that employs the UPPAAL model checker to verify properties on a system.

Furthermore, we plan to study a new design interface theory for timed-component systems [4], considered as Timed I/O automata with game semantic, that would support compositional design and verification of timed component-based embedded systems. We will employ an extended UPPAAL-TIGA, which is an engine for solving timed games in order to manipulate this design methodology.

Acknowledgement. This work was funded by PROGRESS Research Centre at MDH in Sweden, FUNDP PRECISE Research Center in Information Systems Engineering (CERUNA project) Namur University, and Belgian Science Policy (MoVES project). We also wish to acknowledge the participation of collaborator Volvo Technology Corporation from Sweden. Special thanks to Henrik Lönn and Lei Feng (Volvo Technology Corporation, Gothenburg, Sweden) for their valuable feedback.

References

1. Advancing Traffic Efficiency and Safety through Software Technology Phase 2, European project (2010), `http://www.atesst.org`
2. AUTomotive Open System Architecture (2010), `http://www.autosar.org`

3. Carlson, J., Håkansson, J., Pettersson, P.: SaveCCM: An analysable component model for real-time systems. In: Liu, Z., Barbosa, L. (eds.) Proceedings of the 2nd Workshop on Formal Aspects of Components Software (FACS 2005). Electronic Notes in Theoretical Computer Science, vol. 160, pp. 127–140. Elsevier, Amsterdam (2006)
4. David, A., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: Timed i/o automata: a complete specification theory for real-time systems. Hybrid Systems, 91–100 (2010)
5. Feng, L., Chen, D., Lönn, H., Törngren, M.: Verifying system behaviors in east-adl2 with the SPIN model checker. In: IEEE International Conference on Mechatronics and Automation, Xi'an China (August 2011)
6. Grimm, K.: Software technology in an automotive company - major challenges. In: International Conference on Software Engineering, p. 498 (2003)
7. Håkansson, J.: Design and verification of component based real-time systems. PhD thesis, Uppsala University (2009)
8. Håkansson, J., Carlson, J., Monot, A., Pettersson, P., Slutej, D.: Component-based design and analysis of embedded systems with UPPAAL PORT. In: Cha, S., Choi, J.-Y., Kim, M., Lee, I., Viswanathan, M. (eds.) ATVA 2008. LNCS, vol. 5311, pp. 252–257. Springer, Heidelberg (2008)
9. Lindahl, M., Pettersson, P., Yi, W.: Formal design and analysis of a gear controller. In: Steffen, B. (ed.) TACAS 1998. LNCS, vol. 1384, pp. 281–297. Springer, Heidelberg (1998)
10. Jerson Ortiz, J., Legay, A., Schobbens, P.-Y.: Memory event clocks. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 198–212. Springer, Heidelberg (2010)
11. Open Source Tool for Graphical UML2 Modeling (2010), `http://www.papyrusuml.org`
12. Raskin, J.-F., Schobbens, P.-Y.: State clock logic: A decidable real-time logic. In: Maler, O. (ed.) HART 1997. LNCS, vol. 1201, pp. 33–47. Springer, Heidelberg (1997)
13. Rumbaugh, J., Jacobson, I.: United Modeling Language User Guide, 2nd edn. Addison-Wesley, Reading (1998)
14. Sandberg, A., Chen, D., Lönn, H., Johansson, R., Feng, L., Törngren, M., Torchiaro, S., Tavakoli-Kolagari, R., Abele, A.: Model-based safety engineering of interdependent functions in automotive vehicles using EAST-ADL2. In: Schoitsch, E. (ed.) SAFECOMP 2010. LNCS, vol. 6351, pp. 332–346. Springer, Heidelberg (2010)
15. Sangiovanni-Vincentelli, A., Di Natale, M.: Embedded system design for automotive applications. Computer 40(10), 42–51 (2007)
16. SAVE-IDE project at source net, `http://sourceforge.net/projects/save-ide/`
17. Sentilles, S., Håkansson, J., Pettersson, P., Crnkovic, I.: SAVE-IDE, an integrated development environment for building predictable component-based embedded systems. In: Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering, ASE 2008 (September 2008)
18. Suryadevara, J., Kang, E.-Y., Seceleanu, C., Pettersson, P.: Bridging the semantic gap between abstract models of embedded systems. In: Grunske, L., Reussner, R., Plasil, F. (eds.) CBSE 2010. LNCS, vol. 6092, pp. 55–73. Springer, Heidelberg (2010)