Towards the Integration of UPPAAL for Formal Verification of EAST-ADL Timing Constraint Specification

Tahir Naseer Qureshi, De-Jiu Chen, Magnus Persson, Martin Tömgren
KTH – The Royal Institute of Technology
Brinellvägen 83, 10044, Stockholm, Sweden
{tnqu, chen, magnper, martin}@md.kth.se

ABSTRACT
EAST-ADL is an architecture description language developed for specifications of automotive embedded systems at multiple abstraction levels. Based on the best practices in model-based system development (MBD), it provides necessary artifacts for integrating and managing various concerns in an entire system lifecycle. Requirements engineering, safety engineering and the assignments of non-functional constraints are few examples of the concerns supported by EAST-ADL. This paper presents an effort to investigate the support for a formal verification of the execution timing constraints declared in EAST-ADL using the UPPAAL model checker. The results include a transformation scheme and a prototype transformation employing MQL (Model Query Language). Two case studies, of an emergency braking scheme and a prototype transformation employing MQL, are used to support the work.

Keywords
Model-based development, timing analysis, EAST-ADL, UPPAAL, model transformation, MDWorkbench, MQL, timed automata, formal methods, model checking.

1. INTRODUCTION
Development of large embedded systems in automotive vehicles can be a complex process. The factors contributing to the complexity include an increasing number of highly coupled functions with stringent performance and safety constraints. In addition, the development involves engineering teams from multiple disciplines and organizations. This implies challenges during the integration of functions and components [1]. Separation of concerns with distinct views, the support for early verification and validation (V&V), and high reusability and maintainability are a few examples of the current industrial needs.

EAST-ADL and its methodology [5, 6] are emerging as promising approach to MBD (model-based development) [24] for automotive embedded systems. It provides a domain specific modeling basis for the formalization of architecture design and related functional and non-functional constraints at multiple abstraction levels along with the fulfillment of the industrial needs mentioned in the last paragraph. Due to these features and its life cycle orientation, EAST-ADL distinguishes from other architecture description languages (ADL). Despite this fact, the success of its industrial adoption depends on the availability of tool support.

The major objective of this work is to enable formal verifications of timing constraints specified with EAST-ADL. In particular, we have investigated the integration of UPPAAL [7] and related automation possibilities. UPPAAL is a model-checking tool based on timed automata theory for modeling and V&V of real-time systems. The tool has been applied in several industrial case studies. Both UPPAAL and EAST-ADL are further described in later sections.

The presented effort resulted in a transformation scheme and a partially automatic transformation from EAST-ADL models to UPPAAL models. The transformation allows a quick quality control of timing constraints in regard to consistency and completeness. As a conclusion of this work, a few challenges in achieving a fully automated model transformation support are also identified. To support the work two case studies, an emergency brake assistant (EBA) and a brake-by-wire (BBW) system, are used. While the first one is used for studying the relationship between EAST-ADL and UPPAAL elements, the latter is used as a case study for the verification of the mapping scheme and transformation solution.

The paper is organized as follows. An account of related work is presented in the next section followed by an overview of EAST-ADL and UPPAAL in section 3 and 4 respectively. In section 5 a brief discussion of different integration possibilities is presented. The case studies and the methodology followed in the work are discussed in section 6. The transformation scheme and prototype transformation along with verification results are presented in sections 7 and 8 respectively. The paper is concluded with a discussion in section 9.

2. RELATED WORK
A lot of efforts have been carried out to allow the analysis, verification and validation of system architecture design captured in EAST-ADL through the integration of external tools, see [http://91.189.41.70/~u4528772/index.php?sida=203&rubrik=92]. In [8], one effort to integrate SPIN model checker for formal verification of EAST-ADL models is presented. The SPIN transformation needs to be updated for the upcoming version of EAST-ADL. The results from the work presented in this paper will contribute to the tool support of that work.

The timing constraint package of EAST-ADL allows the declarations of timing constraints of system functions. This EAST-ADL modeling package was developed mainly through the TIMMO project [11] and also referred to as TADL (Timing Augmented Description Language) [21]. TIMMO has also developed a methodology [12] for timing design with EAST-ADL as the reference for related system artifacts. The presented work allows a formal verification of timing constraints declared for a system and its parts in regard to the consistency and completeness.

The authors of [9] and [10] have proposed the use of MARTE [22] for complementing EAST-ADL to enable timing analysis. Our work only relies on the EAST-ADL timing constraint...
package but can be extended to consider those external formalisms.

Similar to our work, [13] verifies EAST-ADL models using UPPAAL-PORT\(^2\). The work mainly uses structural information of EAST-ADL whereas UPPAAL is used to specify behavior as well as perform analysis. In contrast our work uses UPPAAL only for the verification of timing constraints declared in EAST-ADL.

3. EAST-ADL

EAST-ADL (Electronics. Architecture and Software Technology – Architecture Description Language) is an automotive initiative in developing a common standard language for formalizing the architecture(s) of vehicular embedded systems and related life-cycle concerns. The language development has been supported through several European projects, including EAST-EEA, ATESST, ATESST2, and currently MAENAD. Architectural specifications are supported at multiple levels of abstraction, namely vehicle, analysis, design and implementation levels. These levels are related to overall product line features, system functionalities and interfaces, detailed design with hardware architecture and actual hardware/software configuration respectively.

In addition to the core language for architecture specification, EAST-ADL also has extensions for capturing the related behavior definitions of functions, environmental situations, requirements, timing and dependability constraints, and V&V cases. These extensions can be applied at multiple abstraction levels. The language allows explicit declarations of non-functional constraints, such as in regard to timing, reliability and safety, for early predictions of such properties in the system design.

4. UPPAAL

UPPAAL [7] is a model checking tool for modeling, validation and verification of real-time systems. The tool is implemented in Java and C++, having three main parts: an editor, a simulator and a verifier, for modeling, early fault detection (by examination i.e. without exhaustive checking) and verification (covering exhaustive dynamic behavior) respectively. The tool has been used in many industrial cases such as a gear box controller from Mecel AB and Philips Audio protocol.

A system in UPPAAL is modeled as a network of timed automata [15]. A subset of CTL (computation tree logic) is the basis for the query language in UPPAAL. The following three kinds of properties can be checked with UPPAAL.

- **Reachability** i.e. some condition can possibly be satisfied.
- **Safety** i.e. some condition will never occur.
- **Liveliness** i.e. some condition will eventually become true.

UPPAAL uses the concept of templates for reusability and prototyping of system components. Each template can be instantiated multiple times with varying parameters. The instantiation is called a *process*. Furthermore, a syntax similar to the C programming language is used with UPPAAL, which makes the transformation of syntax easier as compared to [8], where C syntax in EAST-ADL was transformed to PROMELA to enable model checking in SPIN.

5. INTEGRATION OUTLOOK AND FOCUS

There exist several integration possibilities which can be considered for EAST-ADL and UPPAAL. UPPAAL can be used for formal verification of application-internal behavior specified using EAST-ADL’s behavior extension [18], execution behavior specified by core EAST-ADL constructs and the timing extension. Schedulability analysis using frameworks like [19] is another possibility which can be explored.

For an efficient design process, the integration often needs to be automated required transformation of models. These transformations can be realized by different technologies such as ATL (Atlas Transformation Language) [3].

An embedded system behavior can be classified into several types. A few types are application behavior related to the functionality of a component, required behavior i.e. the intended behavior and the actual behavior i.e. the behavior which is observed during run-time etc. The presented work focuses on the core language constructs and behavior related to triggering along with timing constraints for execution declared in EAST-ADL. The transformation uses MQL (Model Query Language) for realization of the proposed mapping.

6. RESEARCH APPROACH AND CASE STUDIES

As mentioned earlier, the work is based on two case studies. The first case study is of an EBA (Emergency Brake Assistant). In [16] this case was used to demonstrate the usefulness of formal methods in the refinement process of large scale systems. This case is highly complex with more than 8 components working concurrently. In addition to the software behavior, it also includes a CAN (Controller Area Network) protocol implementation. In [16], the EBA case was verified for fulfillment of requirements such as the time consumed by ECUs, the overall end-to-end response time and a deadlock freeness. For further details about the case study and its verification, the readers are referred to [16].

EAST-ADL specifications [5] were analyzed to explore the correspondence between the concepts from the EBA case with various EAST-ADL artifacts. This was carried out by modeling all the possible applicable artifacts in PapyrusUML modeler [17], where a UML profile for EAST-ADL is used for architectural specification. Based on this work a possible transformation scheme described in next section was devised.

The brake-by-wire (BBW) system is a representative industrial case study. This case has been used in several EAST-ADL related projects like ATESST2 and TIMMO. While the EBA case covers the aspects required for formal verification using timed automata, BBW case provides coverage of EAST-ADL artifacts and methodology at multiple abstraction levels. In addition to the structural information, the BBW specifications include features, requirements, error models for safety analysis and constraints at multiple abstraction levels\(^3\).

A simplified version of the case is shown in Figure 1. Similar to the EBA system, BBW consists of multiple components. Apart from the actuator, all the components have an execution period of

---

\(^2\) [http://www.uppaal.org/port](http://www.uppaal.org/port)

\(^3\) The readers are referred to the tutorial on EAST-ADL usage available on request from the ATESST2 consortium.
One of the end-to-end requirements is that the duration between the outputs of the brake torque calculator and an actuator should be less than or equal to 300ms. The role of the components are as follows:

- **Brake torque calculator** – senses the brake pedal position and sends a desired torque command to the global torque calculator.
- **Global brake controller** – calculates the torque required for each wheel and sends the calculated value to ABS at each wheel.
- **ABS** – An anti-lock braking system controls the braking to prevent the locking of wheel to avoid skidding.
- **Actuator** – The component which performs the actual braking by applying the brake pads to the brake disc.

Figure 1. Simplified brake-by-wire control system.

Figure 2. Brake torque calculator specifications in EAST-ADL

For the presented work, the parts of selected architectural specifications are mainly from structure, behavior and timing packages. This is illustrated for the brake torque calculator (BTC) in Figure 2. A BTC has a periodicity of 10ms and takes a maximum of 3ms for execution. A design function type and function trigger belong to structure and behavior specifications respectively. The remaining artifacts in the figure belong to the timing extension. In the figure, the function trigger is used to specify the trigger and event function for specifying timing constraints.

The structural modeling related to the transformation is carried out at EAST-ADL design level of abstraction shown in Figure 3. For simplicity and space limitation, only one out of four ABS and Actuator components are shown in the figure. It can be seen in the figure that the overall system is an EAST-ADL Design Function Type (DFT) and its components are Design Function Prototypes (DFP). For example, the DFP BTC prototypes a Brake Torque Calculator shown in Figure 2.

Figure 3. Structural model of BBW system in EAST-ADL.

### 7. Transformation Scheme

Before proceeding further, we would like to mention a few assumptions which are made for the sake of simplicity:

1. EAST-ADL uses a concept of type and prototype. One function can contain instances of other functions in the form or prototypes. For the presented work, no function type is allowed to have instances of other function types except for Functional Design Architecture (FDA). This is to simplify the work and have one to one correspondence with UPPAAL. In this way, an FDA becomes the overall system and the instantiated functions as UPPAAL templates.
2. Models are complete implying that all the artifacts and properties for a function type or a template required for transformation are available.
3. The communication channel between different processes in UPPAAL is of broadcast type. This assumption was made only to simplify the development of the transformation prototype described in the next section.

With the above mentioned assumptions, a transformation scheme has been developed which is described in Listing 1. The modeling elements of EAST-ADL used in the transformation can be found in Figure A1 of the Appendix, where gray, yellow and white colors in the figure represent structure (design level of abstraction), behavior and timing artifacts respectively. Interested readers are referred to [4] for a list of UPPAAL artifacts.

**Listing 1. Transformation scheme**

1. For each elementary DFT, create an UPPAAL template and a clock named LocalClock in the template declaration.
   a. If the function trigger policy for the corresponding DFT is TIME,
      - Create three locations namely Init, Execute and Finished.
      - Create transitions from Init to Execute, Execute to Finished and Finished to Init respectively.
      - If a worst-case execution time is specified, add an invariant i.e. “LocalClock<=ExecutionTime” for the Execute location.
      - Add an invariant “LocalClock<=Period” for the Finished location.
      - Add a guard “LocalClock==Period” for the transition from Finished to Init.
• Add a clock update “LocalClock:=0” for the transition from Init to Execute.
• For each port,
  • Define a synchronization “Port.name?” for the transition from Init to Execute and “Port.name!” for the transition from Execute to Finished.
  • Add a template parameter “broadcast chan &Port.name”

b. If related function trigger policy is EVENT,
• Create two locations namely Init and Execute.
• Create two transitions from Init to Execute, and Execute to Init respectively.
• If a worst-case execution time is specified, add an invariant i.e. “LocalClock<=ExecutionTime” for the Execute location.
• Add a clock update “LocalClock:=0” for the transition from Init to Execute.
• For each port,
  • Define a synchronization “Port.name?” for the transition from Init to Execute and “Port.name!” for the transition from Execute to Init for the direction in and out respectively. Here “Port.name” refers to the name of a port in EAST-ADL.
  • Add the template parameter “broadcast chan &Port.name”

2. For the non-elementary DFT define an UPPAAL-system as follows:
   a. For each Function Connector (FC) define a channel with the same name as the FC.
   b. Define system declaration (SD).
   c. For each DFP in the DFT
      • Define a process with the same name as that of DFP.
      • For each connector in DFT, if the connected prototype == DFP then add the connector name as the process declaration as an input argument related to the parameter defined in the last parts of step 1a and 1b.
      • Add the process name to the SD.

3. For each Reaction Time Constraint
   a. Add a template with a clock named “timeClock” and two locations, namely Init and Finished.
   b. Add two transitions between the two locations with opposite directions.
   c. For the transition from Init to Finished, update the clock to zero and add synchronization and associated channel for receiving the stimulus event.
   d. For the transition from Finished to Init, add synchronization and associated channel for receiving the response event.

The timed automata obtained after the transformations only represent the execution behavior triggering including triggering and timing of communication. This means the internal application behavior corresponds to the execute state mentioned in the above scheme, and that only timing of communication is modeled, not content.

8. PROTOTYPE TRANSFORMATION AND VERIFICATION RESULTS

Based on the proposed algorithm a prototype transformation from EAST-ADL models to UPPAAL has been implemented with MQL (Model Query Language) using MDWorkbench [2]. The approach is inspired by the work introduced in [14]. The choice of MQL is motivated by its simplicity as compared to other languages like ATL (Atlas Transformation Language) [3]. The transformation is exogenous and unidirectional where two Ecore meta-models were developed. The first one is a simplified EAST-ADL meta-model, including only the artifacts relevant for the presented work. The second one is an UPPAAL meta-model. The UPPAAL meta-model is generated by automated conversion from the XTA format [4] to Ecore.

For the transformation, a few adaptations of both meta-models were made. These refinements generally include the addition of opposite references for traceability purpose. Addition of the attribute ownedFunctionType (an opposite of reference “connector” for a DFT) for Function Connector shown in Figure 4 is one example of such modifications. These modifications are necessary for MQL when using containment associations to reduce transformation complexity. For example, defining an ‘ownedfunction’ of a port (shown in Figure 4) will automatically create a ‘port’ link for the associated design function type and hence only transformation rule in MQL is required to be written instead of two.

Figure 4. EAST-ADL meta-model modification illustration.

In addition to the above mentioned automated structural space transformation a manual technical space transformation4 from UPPAAL Ecore to UPPAAL-readable XML was also carried out. Figures 5 (compare with Figure 2 and 3), 6 and 7 illustrate UPPAAL templates corresponding to steps 1a, 1b and 3 of the transformation scheme in the previous section. Some additional synchronizations, such as StopTimeMeasure and StartTimeMeasure, represent the step 3d implemented manually for the presented work.

Figure 5. Brake torque calculator template in UPPAAL.

Figure 6. Actuator template in UPPAAL.

4 Refer to [23] for a comparison the two types of transformations
As a continuation of the work, schedulability analysis of EAST-ADL models using UPPAAL will be investigated. For this the results from [10] and [19] will be analyzed and utilized if feasible. Planned future work also includes enhancement of the transformation prototype by considering the upcoming EAST-ADL behavior extension, of which an early concept is introduced in [18], together with the timing constraints.

Other planned enhancements include consistency checking between requirements model and timing and behavior constraints at EAST-ADL analysis and design levels.

10. ACKNOWLEDGMENTS

The work is partially financed by the MAENAD project. We would also like to thank the authors of [16] for the provision of the EBA models in UPPAAL.

11. REFERENCES


---

Figure 7. A template for end-to-end reaction constraint.

The following reachability and safety properties corresponding to the timing constraints have been verified for the BBW system:

1. The system is deadlock free. In UPPAAL query syntax, \texttt{A[\neg \text{deadlock}].}
2. The execution time is always less than the period for Brake Torque Calculator. In UPPAAL syntax \texttt{A[\neg (BTC.timecheck \implies (BTC.LocalClock<10))]. In this case we added an extra location “timecheck” to verify this requirement.}
3. The reaction time is always less than 300ms. In UPPAAL syntax, \texttt{A[\neg (TMFR.Finished \implies (TMFR.timeClock<300)).}
4. It is possible for the reaction time to be less than 300ms. In UPPAAL syntax, \texttt{E[\neg (TMFR.Finished \implies (TMFR.timeClock<300)).}

All of the above mentioned properties except for the third one were found to be satisfied. The reason for non-satisfaction is the assumption of broadcast type communication for every component. To handle this issue only the signals passed to multiple components should be considered as broadcast type. This issue will be considered for the further enhancement of the prototype.

9. DISCUSSION

In this paper, an effort to integrate UPPAAL for a formal verification of consistency and completeness of timing constraints declared in EAST-ADL is presented. The transformation scheme is focused on the EAST-ADL design level architecture, but can easily be extended to the analysis level architecture. This will require only changing the type of functions and prototypes in the transformation scheme.

Furthermore, the non-blocking semantics of EAST-ADL imply always a deadlock free execution which is also confirmed during the study. The current transformation only considers the sender / receiver type of data-flow interaction. Inclusion of additional information such as an application’s internal logical behaviors with client-server type communication which can create logical deadlocks will be considered for future enhancement of the presented work.

Out of the three parts in the presented transformation scheme, the last part (i.e. number 3 in the transformation scheme) is not included in the prototype. This is due to the need for further investigation on the use of event chains and related delay constraints. This is challenging especially with multiple and parallel traces in addition to the possible dependencies between different applications.

Another possibility is the use of EAST-ADL requirement and V&V models for the purpose of deriving UPPAAL queries and describing the analysis results.


APPENDIX:

Figure A1. Simplified EAST-ADL Meta-model