Comparative Evaluation of Timing Model Extraction Methodologies at EAST-ADL Design Level

Alessio Bucaioni*†, Saad Mubeen*, Federico Ciccozzi* Antonio Cicchetti* and Mikael Sjödin*
* Mälardalen Real-Time Research Centre (MRTC), Mälardalen University, Västerås, Sweden
† Arcticus Systems AB, Järfälla, Sweden
*{alessio.bucaioni, saad.mubeen, federico.ciccozzi, antonio.cicchetti, mikael.sjodin}@mdh.se
†{alessio.bucaioni}@arcticus-systems.com

***Abstract*—There are various methodologies that support the extraction of timing models from EAST-ADL design-level models during the development of vehicular embedded software systems. These timing models are used to predict timing behavior of the systems by performing end-to-end timing analysis. This paper presents a comparative evaluation of three methodologies. We present an evaluation framework that consists of several evaluation features. Using the framework, we compare and evaluate the methodologies against each feature. Eventually, the evaluation results can be used as guidelines for the selection of the most suitable methodology with respect to the end-to-end timing behavior of a given vehicular embedded system.**

***Keywords*—EAST-ADL; RUBUS component model; timing model; timing analysis; model transformations; comparative evaluation**

I. Introduction

During the last two decades, the complexity of vehicular embedded systems has increased considerably, negatively affecting their development cost and time-to-market. In order to mitigate these issues, the research community has defined an architecture description language namely EAST-ADL [1]. It exploits the principles of Component-Based Software Engineering (CBSE) [2] and Model-Driven Engineering (MDE) [3]. Basically, it defines a top-down methodology for the development of these systems. It ensures the separation of concerns by relying on four different abstraction levels. Each level is provided with a specific modeling language and a set of activities to be performed as shown in Fig. 1.

A high-level analysis can be performed at the analysis level for consistency checking of the system requirements. However, timing behavior of the system can be analyzed only at the implementation level by performing, e.g., response-time analysis [4] and end-to-end delay analysis [5]. Considering the timing constraints typical of these systems, the industry is currently pushing to anticipate their timing behavior at the design level for driving architectural refinements and improving reuse [6].

A. Problem Statement

In order to support high-precision timing analysis of these systems, end-to-end timing models need to be extracted from their software architecture. Timing models consist of timing properties, requirements, dependencies, control flows, data flows and linking information of all tasks, messages and task chains contained in the system. The problem is that this timing information is not completely available at the design level. This hampers the application of a high-precision end-to-end timing analysis on the design-level models of the system. To attack this problem, the research community, together with the vehicle industry, has proposed several methodologies for the extraction of the timing models from the design-level models [7], [8].

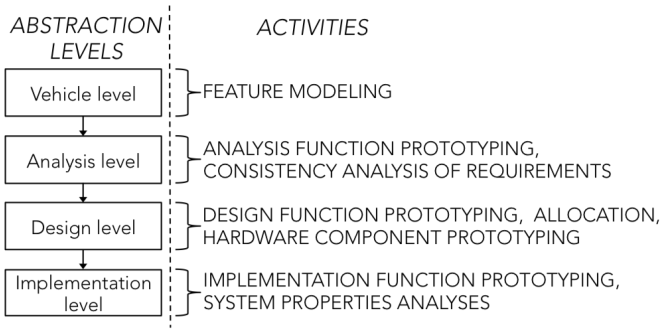

Fig. 1. EAST-ADL abstraction levels and related activities.

B. Paper Contribution

In this paper, we present a novel comparative evaluation of various methodologies that are used for the extraction of timing models from EAST-ADL design-level models. To this end, we i) identify three methodologies that support the extraction of the timing models at the design level, ii) establish a framework consisting of seven evaluation features and iii) compare and evaluate the selected methodologies against the elicited features composing the framework. Eventually, the comparative evaluation can be used as a driver for application-specific selection of the most appropriate methodology.

C. Paper Outline

The rest of the paper is organized as follows. Section II discusses the compared methodologies. Section III presents the evaluation framework, comparison of the methodologies and evaluation results. Section IV discusses the related work documented in the literature. Section V provides a short summary of the contribution and planned future enhancements.

II. Methodologies for Timing Models Extraction

There are several documented approaches for dealing with the transformation of EAST-ADL models into proper input

formats for timing analysis (see Section IV). Nevertheless, none of them can deal with the extraction of the timing models with the aim of supporting response-time analysis and end-to-end timing analysis at the design level. To the best of our knowledge, the methodologies that we take into account in this work are the only ones attempting at tackling such a challenge.

A. The TIMMO2USE Methodology

TIMMO2USE [7] is a large European research project with 17 industrial and academic partners. It aims at the development of novel languages, methodologies and tools capable of annotating timing information at various abstraction levels as shown in Fig. 1. It follows up on the TIMMO project and its results including the methodology and Timing Augmented Description Language (TADL).

The TADL language represents the first attempt in providing EAST-ADL with a language for expressing timing information. Compared with TIMMO, the TIMMO2USE project defines an extension of TADL together with the definition of a development methodology. The TIMMO2USE methodology consists of six activities that are performed for each EAST-ADL abstraction level as shown in Fig. 2. The methodology can be explained as follows.

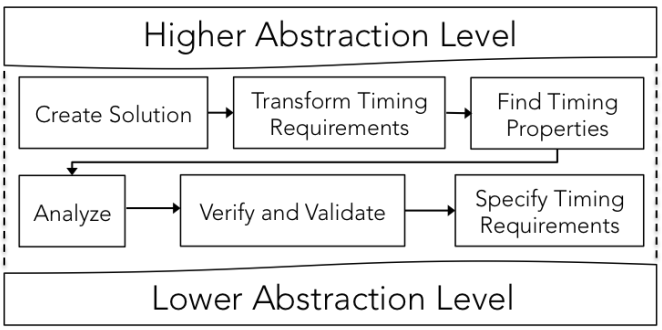

Fig. 2. The TIMMO2USE methodology.

1) Create Solution: Given the timing requirements originating from the higher abstraction level, a solution[1] is created or an already existing solution is revised. The solution is modeled according to the language provided for the considered abstraction level.

2) Transform Timing Requirements: Considering the elicited solution, timing requirements from the higher abstraction level are translated to suitable timing requirements for the current level. This means that the timing requirements are expressed using the timing information provided by the language at the current level.

3) Find Timing Properties: Timing properties of the system are specified based on the translated timing requirements. These properties must be selected in such a way that they can be evaluated against the timing requirements. At this stage, timing properties can be estimated or reused from the previous projects.

4) Analyze: Timing properties are assessed with the aim of deciding whether to proceed with the next task or repeat the previous tasks.

5) Verify and Validate: In this task, the software architecture and its timing properties are verified and validated against the specified requirements.

6) Specify Timing Requirements: The goal of this task is to identify which timing requirements must be considered for the next abstraction level.

B. The Design-to-implementation Level Model Transformation (DTILMT) Methodology

In [8] the authors proposed a semi-automatic methodology, which exploits the principles of MDE and CBSE, for performing end-to-end timing analysis at the design level.

Due to lack of complete timing information, the timing models can not be extracted from the design-level models. In order to solve this issue, the methodology translates design-level models to implementation-level models. In fact, the implementation-level models contain the necessary timing information (e.g., triggering information and control flows) for building timing models on which timing analysis can be performed.

The methodology generates a set of implementation-level models enriched with timing elements whose properties are set by the developer at generation time. This is due to the timing information that is available in the implementation-level model. In fact, the timing information represents variability points in the translation mechanisms. This means that more than one implementation-level model can be a valid translation of a given design-level model. Once the implementation-level models are generated, the end-to-end timing analysis is performed. The analysis results (e.g., end-to-end response times and delays) are compared and evaluated with respect to the specified timing constraints. Based on the evaluation, the methodology is able to select the most suitable implementation-level model based on a better schedulability. Finally, the analysis results are fed back to the starting design-level model. The complete methodology is depicted in Fig. 3.

C. In-house and Tool-based (IH&TB) Methodologies

There are several academic and commercial tools that support modeling of a system using EAST-ADL. Some examples include Mentor Graphics VSA, Papyrus, EATOP, MetaEdit+, Enterprise Architect, Rubus-EAST, No Magic, System Weaver and SE Tool [9]. These tools are used at the top three levels including the vehicle, analysis and design levels as shown in Fig. 1. Some of these tools are actually used in the industry, e.g., VSA, System Weaver and SE Tool[2] However, these tools rely on other tools that support the software development at the implementation level, e.g., AUTOSAR tool chain [10] and Rubus-ICE [11].

The translation from design- to implementation-level models depends upon the in-house translation methods that are exercised by each Original Equipment Manufacturer (OEM) for the integration between the corresponding tools. Some of the translation methods are semi-automatic, e.g., in the case of integration of an EAST-ADL tool with the AUTOSAR tool chain. Whereas, for the others, the translations can be close

[1]Here the term solution refers to a design-level software architecture modeling the vehicular embedded system.

[2]SE Tool is a version of System Weaver that is tailored for the development of vehicle functionality at Volvo.

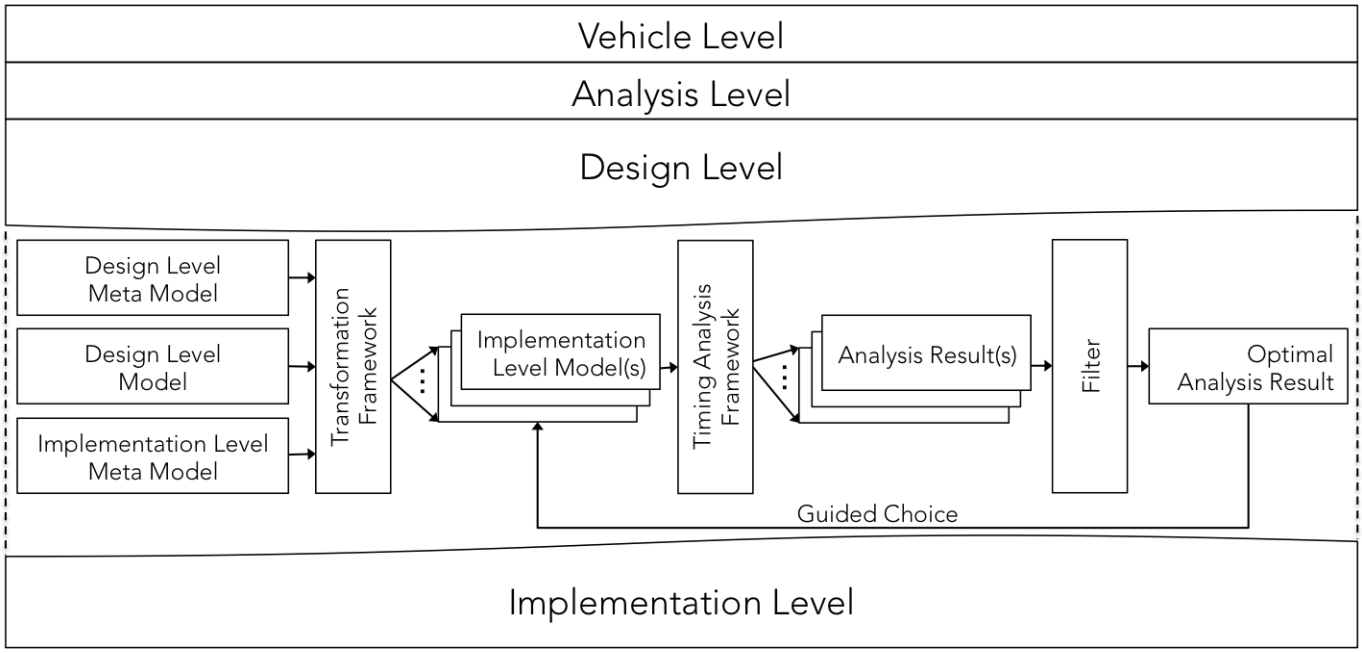

Fig. 3. Design-To-Implementation Level Model Transformation methodology.

to manual with a lot of human-in-the-loop interactions. For example, consider the case of integration between the SE tool and Rubus-ICE, both being commercial tools. The design-level models are manually translated to the implementation-level models by expert integrators using, e.g., MS Visio drawings. Moreover, the translations are based on a lot of assumptions, e.g., often a one-to-one mapping is considered between the design- and implementation-level software components.

III. Comparative Evaluation

In this section, we i) discuss the features comprising the evaluation framework and ii) present a comparative evaluation of the three methodologies (discussed in Section II) using the aforesaid framework.

A. Evaluation Framework

¡¡¡¡¡¡¡ .mine The features, that comprise the evaluation framework, have been elicited considering their relations to the EAST-ADL principles and relevance within the industrial domain. It may be argued that a different set of features could affect the evaluation results. While this can be a valid argument, we believe that a different set of features would lead to a different, yet complementary, evaluation of the methodologies not in contrast with the one presented in this paper. ======= The features that compose the evaluation framework have been elicited considering their relations to the EAST-ADL principles and relevance within the industrial domain. It may be argued that a different set of features could affect the evaluation results. While this can be a valid argument, we believe that a different set of features would lead to a different, yet complementary, evaluation of the methodologies not in contrast with the one presented in this paper. The evaluation framework is composed of the following seven features. ¿¿¿¿¿¿¿ .r7298

The evaluation framework is composed of the following seven features.

1) Separation of Concerns: One of the first attempt to deal with the complexity of software development has been the establishment of views and viewpoints for ensuring separation of concerns. This, in turn, raises the level of abstraction for the software development.

EAST-ADL has been designed to implicitly ensure the separation of concerns by developing the vehicular software architecture at four abstraction levels. In this context, it is crucial for any EAST-ADL-based methodology not to violate such a principle. With respect to this feature, a methodology can fully comply with it *(F)*, partially comply with it *(P)* or violate it *(V)*.

2) Assumptions: A high-precision end-to-end timing analysis cannot be performed on the design-level models due to lack of timing information that is needed to extract the timing models.

One way to solve this issue is to derive the implementation-level models from the design-level models. Then, the timing models are extracted from the derived implementation-level models. Another solution is to extract the timing models directly from the design-level models. However, the extracted timing models lack some timing information, such as triggering and linking information, that is necessary to perform high-precision end-to-end timing analysis. Therefore, assumptions are made to fill the missing timing information in the extracted timing models.

Intuitively, a methodology can either rely *(Y)* or not rely *(N)* on the assumptions.

3) Automation: The development cost and time-to-market can be lowered by assembling a seamless tool chain. In this context, the industry is seeking novel methodologies with

as little as possible human-in-the-loop interactions. Hence, a methodology can be fully automated *(FA)*, partially automated *(PA)* or not automated at all *(NA)*.

4) Maturity: This feature indicates whether or not the methodology is mature enough for industrial exploitation. In this respect, the features indicates if a methodology has been already validated using industrial use cases *(I)* or academic case studies *(A)*.

5) Scalability: It is estimated that the software in modern vehicles consist of nearly 100 million lines of code that run on up to 100 Electronic Control Units (ECUs) [12]. Hence, a methodology must be able to scale and adapt to these software-extensive systems. With respect to this feature, a methodology can be classified as easy to scale *(ES)*, possibly scalable *(PS)* or hard to scale*(HS)*.

6) Number of Extracted Timing Models: The end-to-end timing analysis relies on complete timing models. Hence, the timing models must be extracted from the software architecture. The way in which the timing models are extracted from the design-level models can significantly impact on the analysis results. In fact, due to the lack of timing information, more than one valid timing model can be inferred from a design-level model depending upon how the timing information is assumed or annotated.

Assuming the underlying software architecture to be correct, the timing analysis results can vary depending on how the timing model(s) is (are) extracted. That is, if a methodology is able to extract a single timing model at a time *(S)*, it might happen that the timing analysis applied on it may result in the unschedulable system. Contrariwise, if a methodology is able to extract all the possible relevant timing models *(A)*, then it is highly likely that one of the timing models may correspond to a schedulable system.

7) Trace: In software engineering, the term *traceability* refers to the possibility to establish and use traces, i.e., links between software artifacts throughout the process of software development. Intuitively, given the problem of extracting timing models from the design-level models, a crucial aspect is how the timing model can be traced back to the model where it is extracted. To this end, a methodology can automatically create traces which can be exploited for such a purpose *(A)*. Alternatively, a methodology does not support the automatic creation of traces between the two models; therefore, manual tracing is required *(M)*.

B. Discussion

In Table I we illustrate the comparative evaluation of the selected methodologies according to the features that are included in the proposed evaluation framework. Using the framework, the three methodologies have been evaluated based on the authors' experience (e.g., DTILMT and IH&TB) and their interpretation of the technical specification of the methodologies (e.g., TIMMO2USE).

1) The TIMMO2USE Methodology: This methodology focuses on timing analysis by establishing a set of activities to perform at each EAST-ADL abstraction level. Since it is based on the EAST-ADL methodology, it ensures the separation of concerns by recognizing the EAST-ADL abstraction level.

In order to anticipate the timing behavior at the design level, the engineer has to enrich the design-level models with timing properties that are typical to the implementation-level models. Consequently, this activity partially violates the separation of concerns principle.

Most of the activities in this methodology rely on the engineer's expertise. For example, consider the *"Transforming Timing Requirements"* and *"Finding Timing Properties"* activities. Here, the engineer has to establish how timing requirements from the previous level must be transformed to the current level. In addition, which timing properties - and their values - must be specified at the current level.

Clearly, this methodology is able to generate one timing model per time. Since the methodology is based on the engineer's expertise, it can not be fully automated.

Intuitively, it is hard to scale. The methodology has been evaluated on several validators and use cases from the industry including the brake-by-wire system, steer-by-wire system and adaptive cruise control system [13].

The TIMMO2USE methodology does not directly provide any mechanism for tracing the timing models to the design-level models. Nonetheless, the methodology is based on EAST-ADL which provides metamodeling constructs to support the tracing mechanisms. There are several tools that implement the EAST-ADL metamodeling constructs [9]. However, some of these provide automatic tracing support; while the others provide manual tracing support.

In conclusion, the TIMMO2USE methodology may be very suitable for less software-extensive vehicular embedded architectures as they can be easily managed by an engineer or a team of engineers. Moreover, it may be the favourable choice in the case of legacy systems where a lot of information, models and artifacts are often available for reuse.

2) The DTILMT Methodology: This methodology translates the design-level models to the implementation-level models from where the timing models can be extracted.

Using the principles of MDE, such a translation is automatically performed by means of a model-to-model transformation which generates the implementation-level models from the design-level models. Model transformations make use of the so-called *trace model*, i.e., a model that relates the elements from the generated model to the corresponding elements from the generating models.

The transformation generates the implementation-level models containing the timing elements needed for the timing model extractions. More precisely, considering the aforesaid timing elements, the transformation generates the set of all the meaningful implementation-level models. Therefore, the methodology does not rely on assumptions. Also, the engineer is not required to enrich the design-level models with timing elements. This fully ensures the separation of concerns.

Intuitively, the methodology is easy to scale because it does not have any human-in-the-loop interaction. However, the performance for generating the implementation-level models can downgrade as the set of meaningful implementation-level models can grow significantly in the case of a software-

		Evaluation Features						
		Separation of Concerns	*Assumptions*	*Automation*	*Maturity*	*Scalability*	*# Extracted Timing Models*	*Trace*
Methodology	*TIMMO2USE*	P	Y	NA	I	HS	S	A/M
	DTILMT	F	N	FA/PA	I/A	ES	A	A
	IH&TB	P	Y	PA/NA	I	PS/HS	S	M

TABLE I. COMPARATIVE EVALUATION.

extensive system. The methodology has been validated using an academic case study inspired from the industry.

In a nutshell, the DTILMT methodology proposes an automatic solution to the problem of extracting the timing models from the design-level models. Being an academic methodology, it must be improved prior to its adoption by the industry.

3) The IH&TB Methodology: This methodology does not allow the extraction of timing models at the design level. In the first step, the design-level models are translated to the implementation-level model. In the second step, timing models are automatically extracted from the implementation-level models. Since, this methodology involves human-in-the-loop interactions during the translation step, separation of concerns can not be fully guaranteed.

Although there is a large number of in-house and commercial tools and integrated tool chains, not all tools and tool chains provide automatic tracing support between the timing models and the design-level models. In addition, it relies on several assumptions during the conversion of artifacts from the design to implementation level.

Although the timing model extractions are fully automated, the translation from design- to implementation-level models is semi-automated or manual depending upon the in-house integration method used by the OEM. Since this methodology is used by several OEMs, it can be considered as mature.

The methodology is not very scalable because it is largely dependent upon the judgments of expert integrators. Its scalability varies depending upon the level of human-in-the-loop interactions.

This methodology produces only one implementation-level model corresponding to each design-level model. Hence, it results in the automatic extraction of only one timing model of the application.

IV. RELATED WORK

To the best of our knowledge there is no actual comparison of timing model extraction methods from the EAST-ADL design level models. A work which might be considered related is the one presented in [14], where the authors show an analysis of several modeling languages focusing on a brief comparison between MARTE and EAST-ADL focusing on timing issues. The comparison stresses though mainly modeling differences between the two languages. Nevertheless, in the literature there are several documented approaches that deal with the transformation of EAST-ADL models into proper input formats for timing analysis.

In [15] the authors present an integration of architectural models, especially behavioral descriptions, defined in EAST-ADL and formal verification techniques. More specifically, the approach, called ViTAL, provides a way to transform functional EAST-ADL behaviors to timed automata on which model checking (i.e., timing constraints checking) can be run through the UPPAAL PORT tool. The method makes it possible to identify hidden dependencies and potential conflicts between different vehicle functions before moving to the implementation level.

The work presented in [16] provides another way for transforming from EAST-ADL to timed automata. More specifically, it shows how EAST-ADL timing constraint specifications and the execution behavior of a component can be transformed into a network of timed-automata through a set of pre-defined timed-automata templates; the final goal is, as for the previous approach, model-checking.

While existing works focus on either model-checking or on EAST-ADL levels that conceive timing information as part of the model, our focus is mainly on high-precision end-to-end timing analysis to be run as early as possible in the development process (i.e., design level). Our research group has introduced in [17] challenges and issues to be faced when extracting timing models at various abstraction levels. The comparative evaluation presented in this work represents a natural continuation of that work towards the definition of an effective solution.

V. CONCLUSION AND FUTURE WORK

In the context of software development for vehicular embedded systems, anticipating its end-to-end timing behavior at the design level is a common practice for mitigating development issues such as cost and time-to-market. In this direction, the research community has proposed several methodologies for supporting timing analysis at the design level.

In this paper, we have proposed, for the first time, a comparative evaluation of some of the methodologies supporting the extraction of timing models from the design-level models. To this end, we i) select three methodologies, ii) identify seven features for the comparative evaluation and iii) evaluate the methodologies with respect to the aforesaid features. The comparative evaluation results can be used to select the best methodology for a given application. The evaluation also emphasizes crucial problems in the current methodologies, thus highlighting possible improvements.

In the future, we plan to extend the set of features that compose the evaluation framework to conduct a more detailed evaluation of the methodologies. Also, new methodologies can be brought in to perform more extensive evaluation. Another interesting future work is to compare the methodologies using different case studies with the aim of complementing the comparative evaluation with a quantitative analysis.

ACKNOWLEDGMENT

This work is supported by the Swedish Research Council (VR) and the Swedish Foundation for Strategic Research (SSF) within the projects SynthSoft and PRESS respectively, as well as the Knowledge Foundation (KKS) through the SMARTCore project. The authors would like to thank the industrial partners Arcticus Systems and Volvo Sweden.

REFERENCES

[1] EAST-ADL Domain Model Specification, Deliverable D4.1.1, 2010. http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-06-02.pdf.

[2] Ivica Crnkovic and Magnus Larsson. *Building Reliable Component-Based Software Systems*. 2002.

[3] Jean Bézivin and Olivier Gerbé. Towards a precise definition of the omg/mda framework. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering*, 2001.

[4] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *Computer Journal*, 29(5):390–395, October 1986.

[5] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study. *Computer Science and Information Systems*, 10(1), 2013.

[6] CRYSTAL - CRitical sYSTem engineering AcceLeration, http://www.-crystal-artemis.eu, accessed May, 2014.

[7] TIMMO-2-USE. https://itea3.org/project/timmo-2-use.html.

[8] Alessio Bucaioni, Saad Mubeen, Antonio Cicchetti, and Mikael Sjödin. Exploring timing model extractions at east-adl design-level using model transformations. In *12th International Conference on Information Technology : New Generations*, April 2015.

[9] EAST-ADL Tooling, http://www.east-adl.info/Tooling.html, accessed March 2015.

[10] AUTOSAR Techincal Overview, Release 4.1, Rev. 2, Ver. 1.1.0., The AUTOSAR Consortium, Oct., 2013. http://autosar.org.

[11] Rubus ICE-Integrated Development Environment. http://www.arcticus-systems.com.

[12] R. N. Charette. This car runs on code. *IEEE Spectrum*, 46(3):3, 2009.

[13] Mastering Timing Information for Advanced Automotive Systems Engineering. In the TIMMO-2-USE Brochure, 2012. Available at: http://www.timmo-2-use.org/pdf/T2UBrochure.pdf.

[14] Damjan Temelkovski and Ljerka Beus-Dukic. Specifying timing requirements in domain specific languages for modeling. In *3rd Mediterranean Conference on Embedded Computing (MECO)*, pages 264–267, June 2014.

[15] E.P. Enoiu, R. Marinescu, C. Seceleanu, and P. Pettersson. Vital: A verification tool for east-adl models using uppaal port. In *17th International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 328–337, July 2012.

[16] Tahir Naseer Qureshi, De-Jiu Chen, and Martin Trngren. A timed automata-based method to analyze east-adl timing constraint specifications. In *Modelling Foundations and Applications*, volume 7349 of *Lecture Notes in Computer Science*, pages 303–318. Springer, 2012.

[17] Saad Mubeen, Jukka Mäki-Turja, and Mikael Sjödin. Towards Extraction of Interoperable Timing Models from Component-Based Vehicular Distributed Embedded Systems. In *11th International Conference on Information Technology: New Generations*, pages 655–659, April 2014.