Early timing analysis of vehicular systems: the road from single-core to multi-core

Alessio Bucaioni

Mälardalen University, Västerås, Sweden
alessio.bucaioni@mdh.se
Arcticus Systems AB, Järfälla, Sweden
alessio.bucaioni@arcticus.se

Abstract. In the software development for vehicular embedded systems, timing predictability is paramount for the development of the vehicles' safety features and for their customer value. Modern vehicles' features require new level of computational power. On the one hand, multi-core platforms can provide efficient support for these features. On the other hand, multi-core platforms complicate the software development of vehicular embedded systems as timing predictability is still an open issues for these platforms. In this paper we present a PhD work defining a model-based software development methodology which supports early timing analysis for vehicular embedded systems on multi-core.

Keywords: Model-driven engineering, model-based software development, vehicular embedded systems, timing analysis, multi-core

1 Problem Formulation

In the automotive domain, a cost-effective software development is nowadays paramount as the largest share of vehicle's innovation comes from features which are realised by means of distributed embedded systems [10].

The complexity of vehicular embedded software is constantly increasing, resulting in a growth in size from the few hundreds lines of code of the late 1970s to the 200-300 million lines of code running on modern vehicles [14]. However, size is only one dimension of complexity. Vehicular embedded software is, in fact, characterised by extra-functional properties whose verification complicates software and its development. In the automotive domain, for instance, timing predictability is paramount both for the development of the vehicles' safety features and for their customer value. Its importance has also been acknowledged by several international projects and industrial initiatives, e.g., TIMMO [4], TIMMO2USE [3], Rubus ICE [6]. In order to better deal with the increasing complexity of the software and its properties, new paradigms, focusing on abstraction, separation of concerns and automation, have been proposed and adopted. This is the case of model-driven engineering (MDE) [20].

MDE aims at mitigating the complexity of software and its development by i) shifting the focus from the coding to the modelling activities and by ii) automating error-prone development tasks. In the automotive domain, the adoption of MDE led to the definition of several modelling notations and accompanying model-based methodologies [2][1][11] for the development of predictable embedded software. Among others initiatives, EAST-ADL has been acknowledged as

the de-facto standard in the automotive domain. It relies on a development process which makes use of four abstraction levels for ensuring abstraction and separation of concerns. Typically, the embedded software architecture is represented by a set of functional models from which implementation models are derived. In turn, timing models[1], essential for running schedulability analysis[2] [21], are derived from the implementation models.

Currently, the majority of model-based methodologies allow the development of predictable embedded software for single-core platforms only. Nevertheless, single-core platforms can not provide efficient computational support for modern vehicles' functions which employ data-intensive sensors (e.g., cameras and ultra-sonic sensors) and complex coordination. Contrariwise, multi-core platforms can successfully provide additional processing power for running these computational-intensive functions. However, surveys [8] showed that multi-core software projects can have up to 25% longer schedules and can demand almost 3 times as many software engineers making the projects 4,5 times more expensive. In fact, how to support timing predictability and the extraction of timing models are still open issues for multi-core platforms. Moreover, multi-core platforms also introduce novel concerns such as, e.g., software to hardware allocation.

In this context, model-based techniques can be useful for aiding the software development of vehicular embedded systems for multi-core platforms. For instance, models and model transformations can be employed for raising the development abstraction reducing the need for further expertise and for automating transition among the development phases, respectively. Furthermore, model transformations can ease the extraction of timing models and provide support for the software to hardware allocation.

In this paper we present a PhD work defining a model-based software development methodology which supports early timing analysis of vehicular embedded systems on multi-core. The goal of the presented methodology is to contribute to the achievement of a cost-effective development for vehicular embedded software.

2 Related Works

In the last decades, several model-based methodologies have been proposed for the software development of vehicular embedded systems.

EAST-ADL [2] is layered architectural description language for automotive embedded systems. Its development methodology is a top down development methodology where the software architecture is refined with implementation-oriented details as it navigates down trough the abstraction levels stack. Unlike our methodology, EAST-ADL does not provide any automation means for shifting among the abstraction levels. Furthermore, EAST-ADL modelling notations do not focus on timing. Finally, EAST-ADL does not explicitly support multi-core platforms. AUTOSAR [1] is an industrial initiative to provide standardised software architecture for the development of vehicular embedded systems. As a

[1] A timing model contains timing, communication and implementation information of the software system.

[2] Schedulability analyses [21] are a priori mechanism for predicting the timing behaviour of a system. In our work, we use end-to-end timing analysis [22].

modelling language, AUTOSAR supports multi-core platforms. However, as AUTOSAR is designed for complementing EAST-ADL at the last abstraction level, such a support occurs at late stages of the development only. There are several commercial tools which implement the EAST-ADL and AUTOSAR development methodology. DaVinci Developer[3] by Vector offers a convenient graphical designer for assisting the developer in specifying the AUTOSAR ECUs software architecture. Arctic Studio by ArcCore[4] also supports the specification of AUTOSAR ECUs software architectures and it includes a set of commercial plugins for the Eclipse Modelling Framework. However, both DaVinci Developer and Arctic Studio tool suites only support single-core platforms.

Additional model-based methodologies have been proposed for the software development of cyber-physical systems. AADL [16] is a design language for the model-based software development of real-time distributed embedded systems. AADL has been recently extended for supporting multi-core and partitioned architectures. AADL proposes a development methodology where the software and hardware architectures are i) modelled, ii) complemented with multi-core aspects (e.g., cores' shared resources) and iii) complemented with the so-called isolation specification (e.g., time allocation, memory isolation). Unlike our methodology, in the AADL development methodology timing predictability is only one of the software properties of interest. Therefore, no explicit focus and support is provided. Furthermore, transitions between the architectures are manual and there is no automatic support for software to hardware allocation.

UML-MAST [19] is a methodology and a tool suite for the modelling and analysing of real-time systems expressed in UML. It leverages the UML [7] and MARTE [5] modelling languages. Similarly to our methodology, UML-MAST focuses on the software timing predictability and its verifications and it supports timing analyses for multi-core platforms. Unlike our methodology, it does not provide support for modelling the hardware platform nor software to hardware allocation. Also, its usage is constrained by some restrictions[5].

3 Proposed Solution and Intended Contributions

The contribution of the PhD work presented in this paper is the definition of a model-based methodology for the development of predictable software for vehicular embedded systems on multi-core platforms. One uniqueness of the methodology is that timing predictability verification is enabled at early stages of the development. This is achieved by i) leveraging the Rubus Component Model [17] (RCM) and ii) generation of models, model-based analysis, back-propagation of the results through model transformations. RCM is a component model for the development of predictable embedded real-time software which explicitly focuses on and supports high-precision timing analysis. Another uniqueness of the methodology is the explicit support for modelling design uncertainty. In fact, it leverages a unique notation for representing a set of generated models with a

[3] `http://www.vector.com`
[4] `http://www.arccore.com`
[5] The interested reader may refer to `http://mast.unican.es/umlmast/`

single model with uncertainty, where models' differences are intensionally represented by ad-hoc modelling elements. Figure 1 depicts the workflow of the methodology.

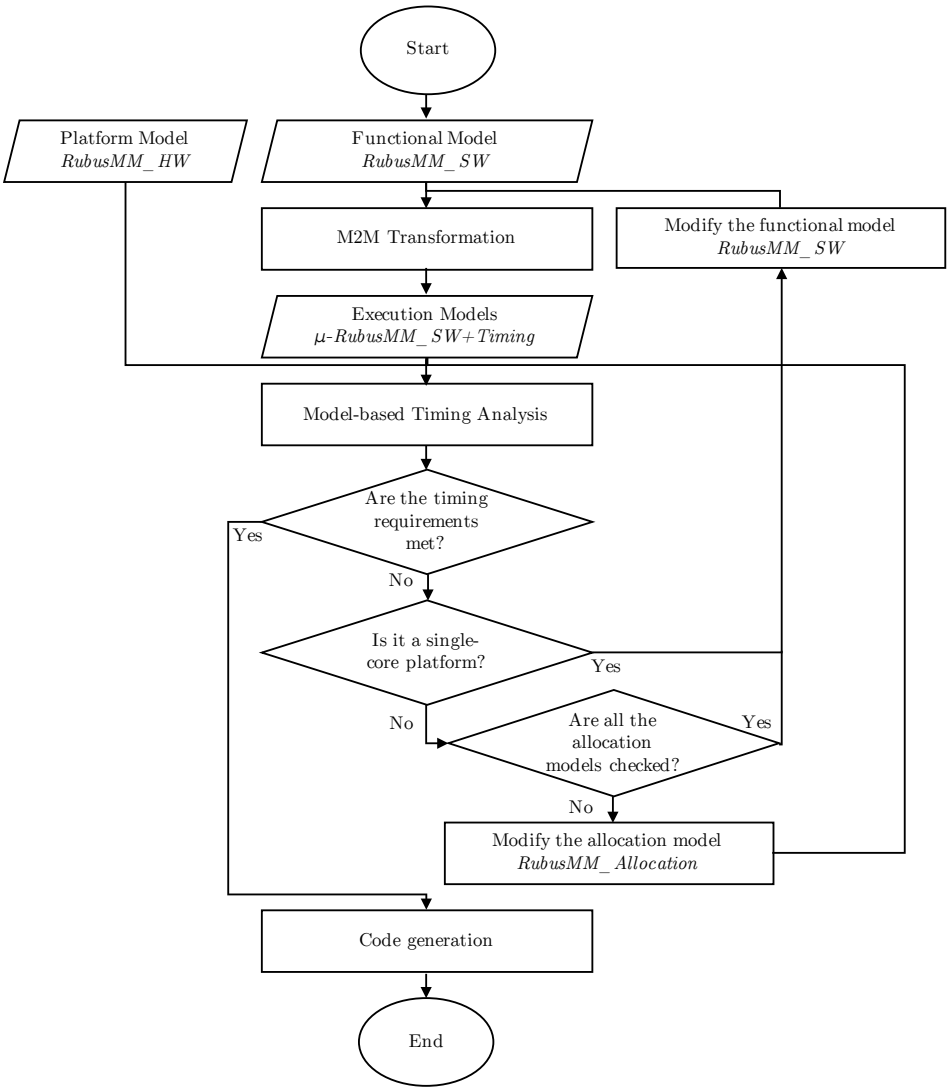

Fig. 1. Model-based methodology for the software development of vehicular embedded systems on multi-core platforms

The workflow starts with a functional model of the software system representing the main software functions and their logic connections. The modelling language leveraged for this task is the software package of the RCM metamodel [13] (RubusMM). At this stage, the model representing the hardware platform is defined too. To this end, the methodology leverages the RubusMM hardware package. Although possible, we assume that the platform model is fixed and does not

undergo changes during the workflow. A non-bijective model-to-model transformation is run on the functional model and generates a set of execution models equipped with timing information. The set contains all the execution models which are meaningful for the leveraged timing analysis. This step is paramount for enabling timing predictability verification at early stages of the development[6]. In order to ease the visualisation of the set of generated execution models, the methodology entails a compact notation able to represent a solution space by means of a model with uncertainty [12] (u-Rubus) where execution models, their commonalities and differences are represented by means of uncertainty points. Please note that, such a representation is obtained automatically via a model-to-model transformation by employing a metamodel-independent technique [15] and it is transparent to the engineer.

At this point, model-based timing analysis can run and the analysis results can be checked against the inherited timing constraints. To this end, the analysis engine considers the generated execution models together with the platform model and calculates possible software-to-hardware allocations satisfying the aforesaid requirements. The software to hardware allocations will be expressed using the allocation package of RubusMM and made available to the engineer for inspection and modification.

If the analysis results do not satisfy the timing requirements, the engineer has to consider the hardware platform, i.e., single- or multi-core. In the case of a single-core platform, the engineer is required to modify the functional model and run the process again as the current functional model does not have any corresponding execution model satisfying the given timing requirements. In the case of a multi-core platform, the engineer is required to modify the allocation model, if another allocation is possible, and run the timing analysis again. If no different allocation is possible, the engineer is required to modify the functional model and run the process again.

If more than one execution model has satisfactory timing performances, the engineer is required to select one and to proceed with the code generation and deployment [7].

4 Preliminary Work and Current Status

In [13] we provided a metamodel definition for RCM, namely RubusMM, focusing on the definition of metamodeling elements for representing the software architecture. We extended the aforesaid metamodel definition in a work which is currently under review at IEEE Access. The extension introduces new architectural elements together with modelling elements for describing timing information. We are working on a further extension of RubusMM which introduces modelling elements for describing the hardware platform and the software to hardware allocation. As Fig 2 shows, we plan to complete the extension of RubusMM by November 2016. In [11] we discussed a methodology for the extraction of timing models from EAST-ADL design level models with the aim of anticipating timing

[6] The interested reader may refer to [11] for further details

[7] The interested reader may refer to [6] [13] for further details on the code generation and deployment.

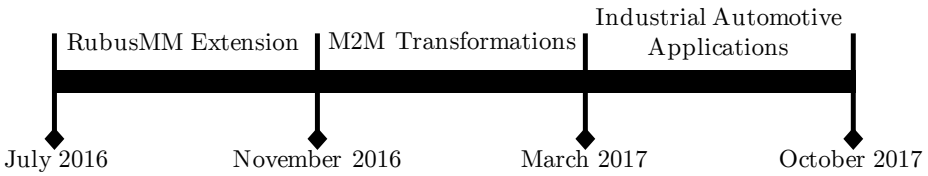

Fig. 2. Timeline for the completion

analysis at design level. The methodology automatically translates the software architecture at design level to all meaningful RCM models. End-to-end timing analysis is performed on each generated model and the analysis results are annotated back to the design level model. In the same work, we demonstrated the applicability of the methodology by exploiting a set of industrial automotive applications. In [12] we extended the methodology with a compact notation for conveniently representing the set of generated models as a single model with uncertainty. Models, commonalities and differences are represented by means of uncertainty points; therefore, the engineer can easily grasp them and consistently make decisions without manually inspecting each model individually. Currently, the methodology only supports single-core platforms. We are working on an extension of the methodology for supporting multi-core platforms. The extension comprises of a refinement of the model-to-model transformation (M2M Transformation in Fig. 1) for the generation of a tree of models with uncertainty and an enhancement of the analysis algorithm for running timing analysis directly on the tree of models with uncertainty. As Fig 2 shows, we plan to complete these enhancements by March 2017. Finally, we are planning to demonstrate the applicability of the new methodology by leveraging a set of industrial automotive applications. We are going to present the results of each of the three above mentioned activities in conference papers and journals.

5 Research Methodology

The research described by this PhD work is a collaborative research between industry and academia. Accordingly, the research methodology leveraged by this research is the engineering method given in [9], that is: "observe existing solutions, propose better solutions, build/develop, measure and analyze, and repeat the process until no more improvements appear possible". We are planning to validate and evaluate this PhD work following the hybrid methods described in [18]. In particular, we will demonstrate the applicability of the extended metamodels and model transformations by leveraging a set of industrial automotive applications. We will leverage a knowledge base of expert opinion which will complement the quantitative data gathered as results of the application of the extended techniques.

Acknowledgments

This work is supported by the Swedish Research Council (VR) through the SynthSoft project. We thank our industrial partners Arcticus Systems AB and Volvo CE, Sweden. Moreover, the authors are grateful to Antonio Cicchetti, Federico Ciccozzi, Saad Mubeen and Mikael Sjödin for their insights during technical discussions.

References

1. AUTOSAR Techincal Overview, Version 2.2.2. AUTOSAR – AUTomotive Open System ARchitecture, Release 3.1, The AUTOSAR Consortium, Aug., 2008. http://autosar.org.
2. 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.
3. TIMMO-2-USE. https://itea3.org/project/timmo-2-use.html.
4. TIMMO Methodology, Version 2, Deliverable 7, Oct. 2009.
5. The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems, 2010. OMG Group, January 2010.
6. Rubus-ICE: Integrated component Development Environment, 2013. http://www.arcticus-systems.com.
7. Introduction To OMG's Unified Modeling Language (UML). OMG Group, May 2016.
8. S. Balacco and C. Rommel. Next generation embedded hardware architectures: Driving onset of project delays, costs overruns and software development challenges. *Klockwork, Inc., Tech. Rep*, 2010.
9. V. R. Basili. The experimental paradigm in software engineering. In *Experimental Software Engineering Issues: Critical Assessment and Future Directions*, pages 1–12. Springer, 1993.
10. M. Broy, I. H. Kruger, A. Pretschner, and C. Salzmann. Engineering automotive software. *Proceedings of the IEEE*, 95(2):356–373, 2007.
11. A. Bucaioni, A. Cicchetti, F. Ciccozzi, R. Eramo, S. Mubeen, and M. Sjödin. Anticipating implementation-level timing analysis for driving design-level decisions in east-adl. In *International Workshop on Modelling in Automotive Software Engineering*, September 2015.
12. A. Bucaioni, A. Cicchetti, F. Ciccozzi, S. Mubeen, M. Sjödin, and A. Pierantonio. Handling uncertainty in automatically generated implementation models in the automotive domain. In *42nd Euromicro Conference series on Software Engineering and Advanced Applications*, September 2016.
13. A. Bucaioni, A. Cicchetti, and M. Sjödin. Towards a metamodel for the rubus component model. In *1st International Workshop on Model-Driven Engineering for Component-Based Software Systems*, Sep. 2014.
14. R. N. Charette. This car runs on code. *IEEE Spectrum*, 46(3):3, 2009.
15. R. Eramo, A. Pierantonio, and G. Rosa. Managing uncertainty in bidirectional model transformations. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering*, pages 49–58. ACM, 2015.
16. P. H. Feiler, D. P. Gluch, and J. J. Hudak. The architecture analysis & design language (aadl): An introduction. Technical report, DTIC Document, 2006.
17. K. Hänninen, J. Mäki-Turja, M. Sjödin, M. Lindberg, J. Lundbäck, and K.-L. Lundbäck. The rubus component model for resource constrained real-time systems. In *3rd IEEE International Symposium on Industrial Embedded Systems*, June 2008.
18. B. Kitchenham, S. Linkman, and D. Law. Desmet: a methodology for evaluating software engineering methods and tools. *Computing & Control Engineering Journal*, 8(3):120–126, 1997.
19. J. Medina, J. Drake, and M. G. Harbour. Uml-mast: modeling and analysis methodology for real-time systems developed with uml case tools. In *Proceedings of the Euromicro Conference on Real-Time Systems*, 2001.
20. D. C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, Feb. 2006.
21. L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real time scheduling theory: A historical perspective. *Real-time systems*, 28(2-3):101–155, 2004.
22. K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and microprogramming*, 40(2):117–134, 1994.