# Timing Challenges in Automotive Software Architectures

Licong Zhang[1], Reinhard Schneider[1], Alejandro Masrur[2], Martin Becker[1], Martin Geier[1] and Samarjit Chakraborty[1]
[1] TU Munich, Germany, [2] TU Chemnitz, Germany

## ABSTRACT

Most of the innovation in the automotive domain is now in electronics and software, which has led to several million lines of code in today's high-end cars. However, in contrast to software in the general purpose computing domain – where mostly functional correctness is of concern – *timing predictability* of automotive software is an important problem which is still largely unsolved. More importantly, this problem is solely addressed within the embedded systems domain with little or no participation from the mainstream software engineering community. The goal of this poster is to highlight some of the aspects of timing analysis of automotive software, as an attempt to involve the broader software engineering research community in this problem.

## Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design—*methodologies*

## General Terms

Design, Performance

## Keywords

Timing Analysis, Automotive, Software Architecture

## 1. INTRODUCTION

Software applications today control some of the most important functionalities in an automobile, ranging from basic safety functions like brake and airbag control, to driver assistance systems like adaptive cruise control and automatic parking and finally to infotainment systems. These applications currently contain around 100 million lines of software code and this number is expected to grow to 200 – 300 million in the near future according Frost & Sullivan, a business research firm. It has been reported that an S-class Mercedes-Benz requires 20 million lines of code for its radio and navigation system and that there are as many electronic control units (ECUs) in this car as in the Airbus A380 (excluding the in-flight entertainment system) [3].

While current validation, testing and debugging methods developed within the software engineering community primarily focus on functional verification, *timing predictability* and hence *timing analysis* is an important and still largely unsolved problem for automotive software, in part, because of its high complexity. In addition to the increasing amount of software, automotive electrical/electronic (E/E) systems are highly elaborate. There are currently 80 to 100 ECUs in high-end cars, which often feature different – increasingly multi-core – processor architectures. These ECUs are connected by several communication buses such as CAN, LIN, FlexRay and MOST in a hierarchical setting.

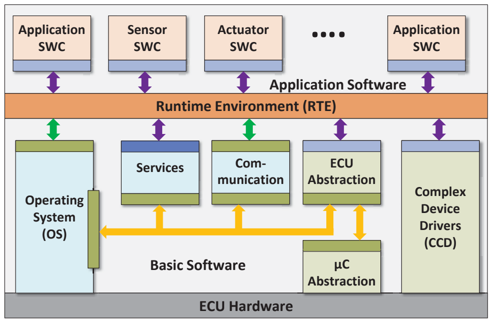

**Figure 1: AUTOSAR software layers**

Timing analysis in such a setting involves several stages – from worst-case execution time (WCET) analysis to system-level timing analysis, percolating through several layers of software and the underlying platform (ECUs, communication buses and gateways) architecture. Often separate timing estimates for the different modules and/or software layers lead to overly pessimistic estimates that are not acceptable in the cost-sensitive automotive domain (in contrast to, e.g., avionics). On the other hand, there has been lately a number of advances towards standardizing automotive software and systems (e.g., AUTOSAR and JasPar) and model-based software development and code synthesis are also on the rise, resulting in new opportunities for timing analysis.

In what follows, we outline some of the major issues in this domain, as an attempt to initiate a discussion and involve the mainstream software engineering community to join the embedded systems community towards addressing this problem. Our aim is also to influence software architectures and development processes in order to improve its timing predictability, since currently the software development and timing analysis processes are largely disjoint.

## 2. CHALLENGES AND OPPORTUNITIES

In the last few years, there has been a considerable effort in the industry towards standardizing automotive software and systems. The main goal is to allow for model-based software design and hardware-independent application software development, thus increasing the re-usability and portability of software components. AUTOSAR[1] (AUTomotive Open System ARchitecture) and JasPar (Japan Automotive Software Platform and Architecture) are part of that effort. They provide specifications for standardized interfaces, middleware and basic software components. Basically, AUTOSAR-compliant automotive software can be divided into three layers: (i) the application software, (ii) the runtime environment (RTE) and (iii) the basic software layer – see Fig. 1. The software developers focus on the implementation of application software components (SWCs), whereas the RTE and necessary basic software components such as the operating system, etc. are generated and configured according to the services required by the applications. The introduction of a standardized software architecture brings new opportunities and challenges. On the one hand, since automotive software is now organized into well-defined layers and modules, it is possible to perform a timing analysis at the layer or module level, which allows for a compositional

[1]www.autosar.org

approach. On the other hand, it also increases the complexity of the problem. There are a number of open problems that still need to be solved, which we will discuss in the following paragraphs.

**Task Level Analysis:** Estimating the WCET of individual functions or code blocks constitutes the most basic timing analysis problem [9]. Since the WCET of a piece of code depends on both the code and the processor, one main challenge is in the joint modeling of the code and the processor's microarchitectural state (e.g., caches, pipelining and speculative execution). While this problem in general is computationally intractable and needs manual intervention (e.g., infeasible path information), it was recently shown for a number of cases that WCET analysis of code generated from high-level models can be significantly easier and can lead to much tighter estimates [5, 8]. The basic technique here is to exploit the model-level information along with the model-to-code association to statically uncover information on loop bounds and infeasible paths which might be difficult to estimate statically directly from the executable or even from program code. Such techniques should be further developed – by especially incorporating standard modeling frameworks typically used in AUTOSAR-based software development such as Simulink/Stateflow – to be useful in the automotive domain. Moreover, it is possible from AUTOSAR R4.0 onwards to describe timing constraints of software modules [1], i.e., data structures are specified for describing timing properties at different levels of abstraction. This can then be used for analysis either based on simulation or formal techniques. However, AUTOSAR does not specify how to fill those data structures. What methods and techniques should be used for this purpose is still unclear. Although there have been first efforts towards how to integrate formal timing analysis to AUTOSAR [6, 7], this is still a matter of ongoing research.

On the other hand, multi-core architectures are gaining ground in the automotive domain. Multi-cores have a number of advantages, such as more computational capacity, low-energy consumption, etc.. However it is more difficult to analyze their timing behavior. The operating system within AUTOSAR was extended to support multi-cores (starting from R4.0 onwards). Although task migration is not allowed at runtime, the analysis of synchronization between tasks running on different cores is a challenging problem, in particular, considering the high inter-dependency of automotive software. In addition, cache effects further complicate the overall timing analysis, which, if not considered, might lead to pessimistic or even unsafe timing estimations. As a result, cache allocation techniques need to be included to take application's properties into account and, hence, allow improving the predictability of the overall timing behavior on the ECU.

**System-level Timing Analysis:** The problem of analyzing the timing properties in complex embedded real-time systems has gained a lot of attention in the embedded system's community. Towards this, analysis techniques like Real-Time Calculus (RTC) [2] and Symbolic Timing Analysis (SymTA) [4] have been developed. Both approaches rely on very general event stream representations and resource models and allow for efficient computation of timing guarantees in complex and heterogeneous embedded systems. A wide range of scheduling, arbitration policies and bus protocols can be modeled by these techniques. Further, they are fully compositional, enabling end-to-end timing analysis across several event chains, and have been integrated in commercially available tools. Since system-level timing analysis is intended to support design considerations in early design stages, the corresponding models often provide too abstract representations of the system architecture hiding implementation details that significantly affect the timing analysis. For instance, the RTC modeling framework relies on infinite buffers, and hence, does not capture buffer overwrite schemes as specified in the AUTOSAR communication stack. Such a loss of implementation details often results in overly conservative timing estimates leading to unnecessary resource over-provisioning. Hence, such details need to be an integral part of the timing analysis framework to achieve tighter estimates. Since enhancements in the analysis framework are quite challenging, the analyzability of the software layers with respect to timing properties gives rise to an important requirement in the design and specification of future automotive software architectures.

Another problem rising from compositional timing analysis is the potentially increased pessimism of estimates. If an application comprises a set of interconnected modules, computing and summing up the WCETs of separated modules might lead to an exaggeratedly large estimation of the overall WCET. In contrast, it might be beneficial to consider the interactions between the different modules towards a tighter WCET estimation. However, it is still unclear how to perform this analysis in the context of automotive/AUTOSAR software, especially since the AUTOSAR architecture was conceived to *abstract* application software from implementation details (such as ECU hardware, mapping onto ECUs, communication on buses, etc.). In reality, the application's timing behavior strongly depends on the implementation details (e.g., communication between two SWCs takes longer if mapped onto different ECUs). As a result, to allow for timing analysis at the system level, there is a need for methods and techniques that allow *back-annotating* AUTOSAR SWCs with details and analysis results of the implementation, which is still an open problem.

**Specification Models, Languages and Tools:** There is a large diversity of development processes and methods in the automotive domain, which has led to a vast number of software development tools over the last years with proprietary interfaces and models precluding seamless tool integration. As a seamless tool chain, including timing as an integral part, has become essential to master the growing complexity in timing analysis, there is an increasing need for standard specification models and tools that facilitate tool integration in the automotive domain.

## 3. REFERENCES

[1] Requirements on timing extensions. available at `http://www.autosar.org/download/R4.0/AUTOSAR_RS_TimingExtensions.pdf`.

[2] S. Chakraborty, S. Kunzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *DATE*, 2003.

[3] R. N. Charette. This Car Runs on Code. *IEEE Spectrum*, Feb. 2009.

[4] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis - the SymTA/S approach. *Computers and Digital Techniques*, 152(2):148–166, 2005.

[5] L. Ju, B. K. Huynh, A. Roychoudhury, and S. Chakraborty. Timing analysis of Esterel programs on general-purpose multiprocessors. In *DAC*, 2010.

[6] K. Klobedanz, C. Kuznik, A. Thuy, and W. Mueller. Timing modeling and analysis for AUTOSAR-based software development - a case study. In *DATE*, 2010.

[7] M.-A. Peraldi-Frati, A. Goknil, M. Adedjouma, and P. Y. Gueguen. Modeling a BSG-E automotive system with the timing augmented description language. In *ISoLA*, 2012.

[8] R. Wilhelm, D. Grund, J. Reineke, M. Schlickling, M. Pister, and C. Ferdinand. Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems. *IEEE Trans. on CAD*, 28(7):966–978, 2009.

[9] R. Wilhelm et al. The worst-case execution-time problem - overview of methods and survey of tools. *TECS*, 7(3), 2008.