# Automated End-to-End Timing Analysis of AUTOSAR-based Causal Event Chains

Padma Iyenghar[a], Lars Huning and Elke Pulvermueller

*Research Group Software Engineering, University of Osnabrueck, Germany*

Keywords: Unified Modeling Language, End-to-End Timing Analysis, AUTOSAR, Event Chains, Embedded Automation.

Abstract: Reflecting the ever-increasing complexity of automotive embedded systems and their stringent timing requirements, a significant level of automated tooling is imperative to enhance software quality. In this context, though AUTOSAR-based systems are nowadays more often developed using powerful Unified Modeling Language (UML) tools, there is a lack of an integrated approach and automated tooling for validation of timing properties. Addressing this aspect, a systematic software engineering and automation approach for timing analysis of AUTOSAR-based systems, already early in the design stages, in state-of-the-practice timing analysis tools is presented. A specific example of a widely used timing analysis measure for automotive systems namely, end-to-end delay analysis is described with the help of an AUTOSAR-based causal event chain example. The practical relevance of the approach is demonstrated by evaluating it in an elaborate automotive case study.

## 1 INTRODUCTION

AUtomotive Open System ARchitecture (AUTOSAR) is already a de facto industry standard for the design and implementation of software for automotive applications. Further, development of AUTOSAR-based embedded systems using Unified Modeling Language (UML) tools is gaining significant attention in the automotive industry.

In automotive embedded architectures, many functions have timing constraints for performance guarantees. Often, such systems are modeled with task chains, which consists of a number of tasks that are in a sequence and have a common ancestor. The timing behavior of the task chain is determined by calculating its end-to-end response time/delay. The end-to-end delay of a task chain is defined as the amount of time elapsed between the arrival of an event at the first task and the production of the response by the last task in the chain (Mubeen et al., 2019). In order to perform the timing analysis of such a system, its end-to-end timing analysis model with timing properties, should be available. Based on this information, timing analysis in state-of-the-practice tools such as SymTA/S (Henia et al., 2005), can predict the execution behavior of the system with respect to end-to-end timing. This is among the most critical and widely used performance guarantee measure in automotive embedded systems.

In AUTOSAR-based systems, an event chain, with an end-to-end timing constraint, describes the causal order for a set of functionally dependent timing events (AUTOSAR, 2017). Thus, incorporating an early, Automated End-to-End Timing Analysis of AUTOSAR-based Causal Event Chains (developed in UML tools such as (IBM Software, 2019)) in state-of-the-practice tools such as SymTA/S ((Henia et al., 2005)) is not only required but also highly beneficial for the AUTOSAR-based Embedded Software Engineering (ESE) development process as a whole. However, the aforementioned entities exist not only as island solutions (i.e., not well integrated together ) but also they require manual extraction and import/export of timing models for validation. Further, these procedures are often carried out late in the development process.

On the other hand, reflecting the increasing complexity of the embedded software domain, automated tooling for checking and monitoring software quality is growing more sophisticated and serving a wider community of stakeholders. Validating the timing behavior (a critical quality aspect) and checking for adherence to real-time constraints in early design stages could save costly corrections of potential errors in

[a] https://orcid.org/0000-0002-1765-3695



Iyenghar, P., Huning, L. and Pulvermueller, E.
Automated End-to-End Timing Analysis of AUTOSAR-based Causal Event Chains.
DOI: 10.5220/0009512904770489
In *Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2020)*, pages 477-489
ISBN: 978-989-758-421-3

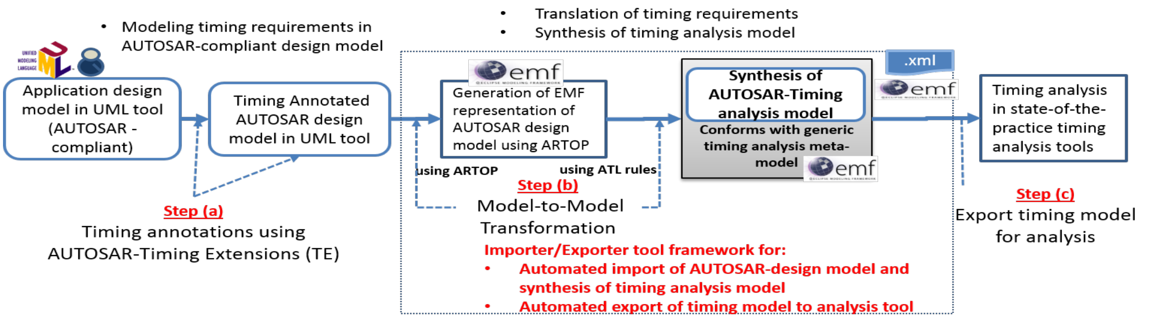


Figure 1: Systematic series of steps for early timing analysis in AUTOSAR-based ESE (introduced in (Iyenghar et al., 2020)).

system design. This is because fixing the errors becomes more costly during later development stages.

The AUTOSAR-Timing Extensions (TE) is a standardized formal timing model to capture timing constraints of automotive systems (AUTOSAR, 2017). With this, it is possible to include the system's timing behavior already in the design models. For instance, the timing requirements corresponding to end-to-end response time could be modeling using AUTOSAR-TE in the respective AUTOSAR-based design models in UML tools. Analysis of these models can be carried out in specialized timing analysis tools such as Timing Architect[1], chronSIM[2] and SymTA/S (Henia et al., 2005) to obtain the end-to-end timing behavior of the causal event chain.

In this context, a systematic series of steps for early timing analysis in AUTOSAR-based ESE, introduced in (Iyenghar et al., 2020), is shown in Fig. 1. Given an initial AUTOSAR-based system design model (already modeled in a UML tool), the series of steps illustrated in Fig. 1 is applied at an early stage in the development process. Step-(a) in Fig. 1 involves the specification of timing requirements in the design model using AUTOSAR-TE to obtain a timing annotated AUTOSAR-design model. With this model as input to step (b), a timing analysis model is synthesized in step (b) in Fig. 1. This timing analysis model can then be provided as input to a timing analysis tool as shown in step(c) in Fig. 1.

## 1.1 Relation to Authors' Previous Work

A timing metamodel for synthesis of a timing analysis model from timing annotated design models is proposed in (Iyenghar et al., 2016). It deals with time modeling in generic ESE using UML. In (Iyenghar and Pulvermueller, 2018), the timing metamodel in (Iyenghar et al., 2016) is extended with energy properties and a methodology for energy-aware timing analysis for IoT use cases is proposed. But, (Iyenghar et al., 2016) and (Iyenghar et al., 2016) do not deal with AUTOSAR-based ESE. More recently, a systematic series of steps as indicated in Fig.1 for the early synthesis of timing models in AUTOSAR-based embedded software system is introduced in (Iyenghar et al., 2020). The main scope of the work in (Iyenghar et al., 2020) concentrated on aspects such as modeling timing requirements in AUTOSAR-based design model using AUTOSAR-TE (step (a) in Fig. 1) and its mapping to a generic timing metamodel (Iyenghar et al., 2016). An initial generic example of AUTOSAR-TE-model to timing-model transformations was also described.

## 1.2 Novel Contributions

The work presented in this paper builds up on the work in (Iyenghar et al., 2020) and provides the following novel contributions (N1, N2 and N3):

- N1 A light weight interfacing importer/exporter tool framework, with plug-ins in Eclipse Modeling Framework (EMF)(EMF, 2020), for automated tooling of step (b) in Fig. 1 (highlighted with a dotted rectangular box).
- N2 An elaboration of different types of transformation rules and helpers implemented for an AUTOSAR importer, along with examples (step b in Fig. 1).
- N3 A full-fledged automotive use case with a specific elaboration of an AUTOSAR-based causal event chain and a detailed experimental evaluation (including step (c) in Fig. 1)) are discussed.

In the remainder of this paper, background and related work is presented in section 2. An elaborate automotive use case of an Autonomous Emergency Braking System (AEBS) case study (novelty N3) is presented in section 3. An importer/exporter tool framework (novelty N1), detailed model transformations (novelty N2) and results of timing analysis are presented in

[1]https://www.timing-architects.com/
[2]https://www.inchron.com/tool-suite/chronsim/

sections 4, 5 and 6 respectively. Section 7 concludes the paper.

## 2 BACKGROUND AND RELATED WORK

Model Driven Development (MDD) [3], methodology employing UML has been increasingly used for design and development in ESE, especially for modeling automotive embedded software systems. Further, this has also been applied for Non-Functional Properties (NFR) such as timing, energy and reliability analysis (Iyenghar et al., 2016), (Noyer et al., 2016a). In non-UML contexts Matlab/Simulink (e.g. modeling control loops) (Franco and et. al, 2016), Rubus Component Model (RCM) (Bucaioni et al., 2017), (Mubeen et al., 2019) and EAST-ADL (Hans et al., 2009) are among other established solutions used for ESE and vehicular domain.

Apart from the aforementioned solutions, a comprehensive and well- established solution used in the automotive sector is the AUTOSAR standard (AUTOSAR, 2018).

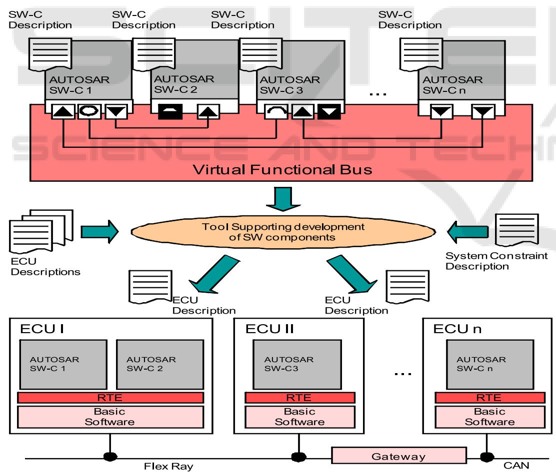


Figure 2: Various components of AUTOSAR framework & mapping of Software Components (SW-Cs) to ECUs.

The various components of the AUTOSAR framework are illustrated together with the mapping of software components to ECUs, in the system configuration step, in Fig. 2. AUTOSAR uses a component-based software architecture, with central modeling elements called *Software Components* (SWCs or SW-Cs). The SWCs ( e.g., *SW-C1* seen at the top of Fig. 2) are used to structure the AUTOSAR model and group functionality into individual components. These components can be connected together, oblivious of the hardware they are running on. This is handled by the *Virtual Function Bus* (VFB), which provides an abstraction layer for the SWC to SWC communication.

Components distributed over different ECUs however, may use the network bus for communication (e.g., *ECUs 1, 2..n* communicating over FlexRay and CAN buses in lower part of the Fig. 2). This is determined automatically by the *Run-Time Environment* (RTE), which is a communication interface for the software components. In each ECU, the RTE provides interfaces between SW-Cs (e.g. AUTOSAR SWC 1 and AUTOSAR SWC 2 in ECU 1 in lower part of Fig. 2) and between SW-C and Basic Software (BSW). In this paper, the design model is created using the AUTOSAR framework and the timing properties are annotated using AUTOSAR-TE.

### 2.1 Timing Modeling and Analysis

Modeling and Analysis of RealTime and Embedded Systems (MARTE)[4] is a standardized UML profile, which extends UML and provides support for modeling the platform, software and hardware aspects of an application (Anssi et al., 2011). Non-UML time modeling alternatives include *SystemC* (Bhasker, 2010) and Event-B[5] to name a few.

#### 2.1.1 AUTOSAR-Timing Extensions

The AUTOSAR-Timing Extensions (TE) metamodel feature an event-based model for the description of the software's temporal behavior and can be defined on top of a system architecture. The AUTOSAR release with timing extensions and its own timing model find extensive usage in the automotive industry (Hans et al., 2009), (Peraldi-Frati et al., 2012) and (Ficek et al., 2012). The TE metamodel provides five different views for timing specification, depending on the kind of timing behavior described in the AUTOSAR model (AUTOSAR, 2017). The five views are *VfbTiming*, *SwcTiming* (describes internal behavior timing of SWC), *SystemTiming*, *BswModuleTiming* and *EcuTiming*. In the experimental evaluation, the *SwcTiming* view is employed, because the system configuration and timing specification steps use the SWCs. Further explanation of AUTOSAR methodology and AUTOSAR-TE are not provided here because of space limitations (interested readers are referred to (AUTOSAR, 2018)).

[3]https://www.omg.org/mda/

[4]http://www.omg.org/omgmarte/

[5]http://www.event-b.org/index.html

#### 2.1.2 Autosar-based Causal Event Chain

In component-based ESE, individual sub-systems are modeled with chains of components that are translated to chains of tasks for scheduling analysis. The timing requirements of such chains, which we coarsely refer to as the end-to-end delay can be specified in the component model and also estimated/analyzed during a timing analysis.

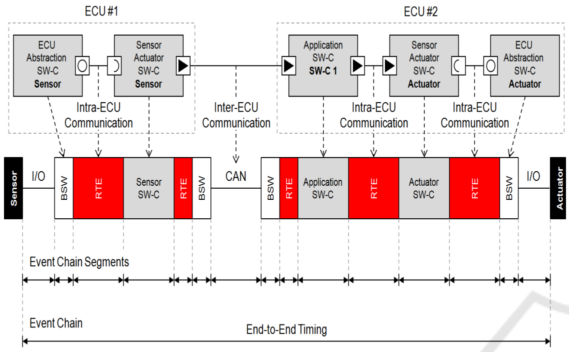


Figure 3: End-To-End timing of a AUTOSAR-based causal event chain (AUTOSAR, 2017).

Fig. 3 shows an event chain end-to-end timing describing the causal dependency between "Sensor" and "Actuator". The sequence of event chain segments shows the details of end-to-end timing according to the AUTOSAR timing views. A timing event chain describes a causal order for a set of functionally dependent timing events. Each event chain defines at least the relationship between two differing events, its stimulus and response which describe its start and end point respectively. This means that if the stimulus event occurs then the response event occurs after.

One way to guarantee that the system meets its timing requirements is to perform pre-run-time analysis of it (e.g. already during design level), using end-to-end timing analysis. Such analysis can validate the timing requirements without performing exhaustive testing. In (Becker et al., 2017), methods to compute end-to-end delays based on different levels of system information is presented and evaluated in an industrial case study. An evaluation of this work is carried out in an automotive industrial case study, using a commercial tool chain, Rubus-ICE. Similarly, (Mubeen et al., 2019) targets the challenges that are concerned with the unambiguous refinement of timing requirements, constraints and other timing information among various abstraction levels. However, the aforementioned works concentrates on usage of EAST-ADL and Rubus Component Model and does not concentrate on AUTOSAR, AUTOSAR-TE modeling in UML tool or automated tooling for end-to-end timing analysis.

#### 2.1.3 Model-based Timing Analysis

The specified timing behavior can be analyzed using dedicated timing analysis tools such as Cheddar (Singhoff et al., 2004) and MAST (Harbour et al., 2001). Popular proprietary timing analysis tools include chronSIM , SymTA/S and Timing Architect. These tools are independent of the modeling languages used. Therefore, they require the timing specifications to be in a particular format, although some provide import functions for common modeling languages. But, there is no tool support for automated synthesis and export of AUTOSAR-based timing analysis model (from AUTOSAR-based application design model in UML tools) to these timing analysis tools. AUTOSAR-TE is used for a model-based timing analysis in works such as (Klobedanz et al., 2010), (Kim et al., 2016) and (Scheickl et al., 2012). However, a systematic model-based approach in timing analysis of AUTOSAR-based systems in timing analysis tools is missing.

### 2.2 Gaps Identified and Challenges Addressed

On examining the related literature and state-of-the-practice tools, it can be stated that several related work deal with examination of the required timing properties for end-to-end timing analysis. Many industrial tools implement the end-to-end delay analysis for event chains. These exist as island solutions without an automated integrated workflow from timing specification in design model to timing analysis. They further require manual creation and/or export of *timing-analysis-model artifacts* in the analysis tool, for timing analysis. Thus, automated timing analysis (e.g. end-to-end delay) of AUTOSAR-based systems (e.g. causal event chains) described in UML tool (e.g. Rhapsody developer) in state-of-the-practice timing analysis tool (e.g. SymTA/S) is missing. This is a significant step to address and evaluate automated tooling for checking and monitoring software quality.

## 3 USE CASE: AUTONOMOUS EMERGENCY BRAKING SYSTEM (AEBS)

The main purpose of an Autonomous Emergency Braking Systems (AEBSs) is to warn the driver in case of an imminent frontal collision. This commonly happens through visual and acoustic warning signals as a first step. The next level of warning is often a tactile warning. The AEBS in cars use the Time-To-Collision (TTC) value (van der Horst and Hogema, 1993), (Kusano and Gabler, 2011) to estimate the danger of the situation. It is defined as the time left until a collision happens, if every object (e.g. both cars) continues to move at the same speed. To calculate TTC, AEBS needs data such as the distance to frontal objects (e.g. from radar sensors) and wheel speed sensor input at certain speed ranges. In this paper, based on the systems used in current automobiles, a simplified version of an AEBS is modeled as an use case and used for experimental evaluation.

## 3.1 Requirements Specification

To have an exact description of the functionality provided by AEBS, several Functional (FR) and Non-Functional Requirements (NFR) were implemented for this use case.

### 3.1.1 Functional Requirements

*FR 1: The assistant shall support the driver in avoiding frontal collision.*

*FR 2: The system needs constant information about the car's speed.*

*FR 3: The distance to the next car in the same lane shall be measured constantly.*

*FR 4: The assistant shall prepare the brake system when TTC falls below 6s.*

*FR 4: The assistant shall warn the driver when the TTC falls below 5 seconds:* Before intervening with an autonomous braking, the assistant needs to warn the driver in multiple stages of the imminent danger, so that the driver can react appropriately. These stages are entered consecutively with decreasing TTC.

- *Visual: Warning light (TTC=5s):* A warning lamp shall provide a first visual warning.
- *Accoustic: Warning sound (TTC=4s):* An audible signal shall provide a second warning.
- *Tactile: Warning jolt (TTC=3s):* A warning jolt shall precede initiation of a full brake.

*FR 5: The assistant shall engage an emergency braking when the TTC falls below 1.5 seconds:*

### 3.1.2 Non-Functional Timing Requirements

*NFR 1: The system estimates TTC every 50 ms*

*NFR 2: The relative speed of the next car must be calculated every 50 ms*

*NFR 3: The speed of the car must be sampled every 10 ms*

*NFR 4: The distance to the next car must be sampled every 10 ms*

*NFR 5: The load on the ECU processing cores must not exceed 80%*

*NFR 6: New sensor data must influence the TTC computation after a maximum delay of 200ms*: The system needs to react fast to sudden speed changes, so the end-to-end response time from sensor measurements to a reaction needs to be below 200 ms.

## 3.2 Control Flow

The AEBS use case is connected to sensors such as speed and radar sensors and actuators such as the warning LED, speaker and brakes via a software interface. Thus information such as the speed of the car in $ms^{-1}$ (from *wheel speed sensor*), distance in m and relative speed in $ms^{-1}$ (from *radar sensor*) are provided as inputs to the AEBS system. The system is connected to the output actuators such as, *Warning LED* (with on/off state), *speaker* (with the ability to create an acoustic warning signal) and *brakes* (need to have a preparation, warning brake and an emergency brake functionality) for respective output action.

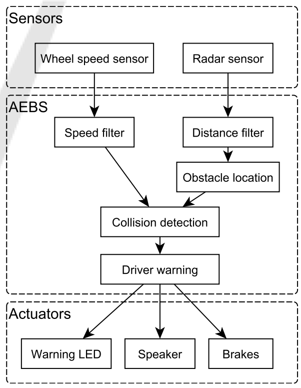


Figure 4: Control flow and modules of the AEBS.

Based on the control logic of the Collision Mitigation Brake System used in (Sugimoto and Sauer, 2005), the AEBS basic control flow is shown in 4. It shows the above-mentioned sensors and actuators as system input and output, as well as the AEBS modules. The sensor values are both processed by filter modules (*Speed filter* and *Distance filter*), which retrieve them periodically, filter outliers and smooth the sensor noise. The distance information from the radar sensor is then further processed by the *Obstacle location* module. It stores the filtered distance values of consecutive measurements and calculates the relative speed of a proceeding car from the change of distance over time. These values are used in the *Collision detection* module, to estimate the TTC. If the filtered speed sensor value is above a certain threshold, indicating a normal driving state, the TTC is sent to the *Driver warning* module. This triggers different stages of warnings, depending on the criticality of the TTC value, or sets off an emergency braking as last resort respectively. If the warning measures are successful and the TTC rises above the thresholds again, the corresponding warnings are cancelled.

## 3.3 AUTOSAR-based Design Model

Given the requirements and the control flow of the AEBS, the AUTOSAR system description of the AEBS is modeled using the IBM Rational Rhapsody Developer modeling tool (IBM Software, 2019). This tool is among the popular UML modeling tool with AUTOSAR support used in the automotive industry, hence it is used in this paper. The first step in implementing the AUTOSAR design model is to define the software components, of which the system is composed of. This is shown in Fig. 5 and described below.

The sensor filter modules on the left-hand side of Fig. 5 are modeled as *SensorActuatorSwComponentTypes*. They have client ports (*speedSensorPort*, *radarSensorPort*) to connect to the corresponding sensors. These ports are typed by *ClientServerInterfaces* that provide an operation for retrieving the sensor value. This is illustrated by the association between the ports and the interfaces, which is stereotyped as a *portType*. The rest of the modules are modeled as *ApplicationSwComponentTypes*, as they do not directly represent a sensor or an actuator.

The communication between the sensor filters and the *CollisionDetection* and *ObstacleLocation* components happens through sender/receiver ports. The filtered *dataElements* get sent to the processing components. Equally, the *ObstacleLocation* sends a list of obstacles (comprising of distance and relative speed) to the *CollisionDetection*. The communication between *CollisionDetection* and *DriverWarning* is also typed as sender/receiver and the corresponding *dataElement* is the TTC value.

In the end, the *DriverWarning* component is connected by client ports (*ledPort*, *speakerPort* and *brakePort*) to the three actuators. The corresponding interfaces provide the necessary operations for the different levels of driver warning, e.g., setting the warning LED light status (*setLight*), playing a warning sound (*playWarningSound*) or performing an emergency brake (*emergencyBrake*).

### 3.3.1 Overall System Description

The system diagram of the AEBS design model is shown in Fig.6. It contains a *SystemMapping* element, which maps the software components to ECUs or to ECU cores respectively. The mapping is necessary for conducting a timing analysis, because it also maps the runnables of the software component's internal behavior to an ECU or an ECU core respectively. It also indicates, which function will be executed on which processing unit later on. In the system mapping, for every processing core, a *swMapping* element linking to the corresponding *hwElement* from the ECU diagram with *l_processingUnit* is created. Then, for every software component prototype, a *swMapping* that links to the core mappings with *l_mappingElement* is created. These prototype mapping elements reference the component prototypes with their tagged value *component*.

## 3.4 Timing Behavior

To perform a timing analysis of the AEBS and verify the results against the timing requirements from 3.1.2, the timing behavior of the AEBS has to be captured. For this, a more detailed division of the modules in functions and a specification of the Core Execution Time (CET)[6] is needed. The functions need to be allocated to tasks, which in turn are allocated to processing cores.

This detailed architecture of the AEBS can be seen in 7. The system is distributed over two processing cores on one ECU. The *ECU Core 2* is responsible for handling the sensor tasks, which consist of a function for retrieving the data (*getSpeed*, *getDistance*) and filtering/sending the data (*sendSpeed*, *sendDistance*). The *ECU Core 1* handles the *ObstacleTask* and *SystemTask*, which handle the tracking of obstacles and calculating/checking the TTC value.

Table 1 lists the detailed timing properties of the functions and tasks. It shows the guaranteed minimum and maximum CET of the functions, which can be obtained using experience values from experts (as in this paper) or run-time measurements of already implemented functions. The periodic activation of the functions is taken from the timing requirements and grouped into fixed priority tasks. The priority values are assigned depending on the importance of the

[6]Length of time the task could take to execute on a specific hardware platform.

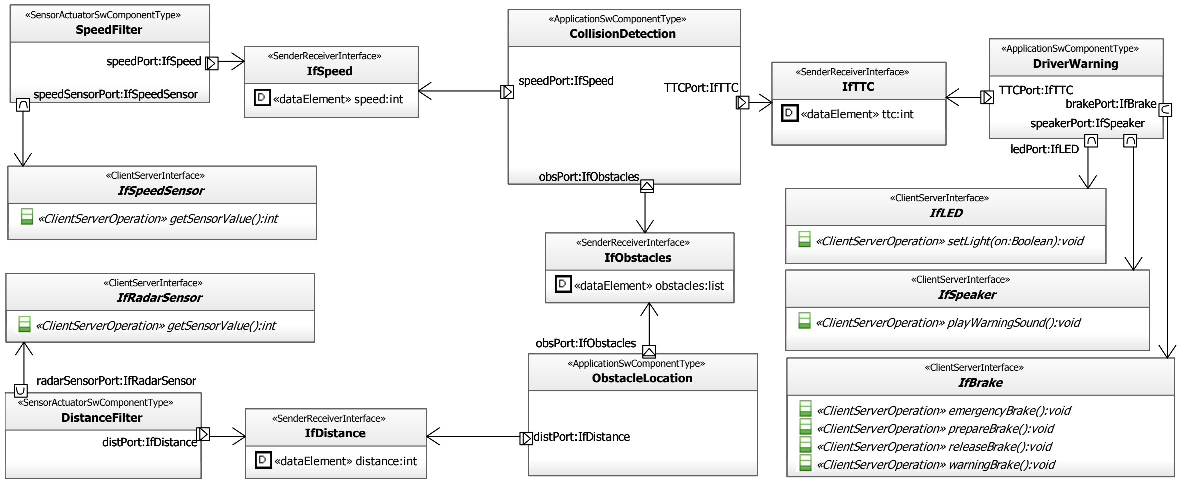


Figure 5: AEBS software components as seen in the software component diagram modeled in Rhapsody.

Table 1: Timing behavior of system functions.

| Module | Function | CET (ms) min – max | Period (ms) | Task | Task priority |
|---|---|---|---|---|---|
| Speed filter | getSpeed<br>sendSpeed | 1 – 2<br>4 – 6 | 10<br>50 | SpeedTask | 0 |
| Distance filter | getDistance<br>sendDistance | 1 – 2<br>4 – 6 | 10<br>50 | RadarTask | 1 |
| Obstacle location | trackObstacles | 6 – 10 | 50 | ObstacleTask | 2 |
| Collision detection<br>Driver warning | timeToCollision<br>checkTTC | 5 – 7<br>3 – 5 | 50<br>50 | SystemTask | 5 |



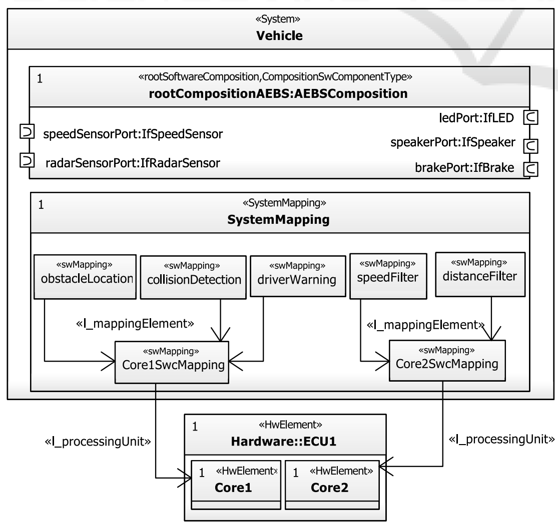


Figure 6: System diagram containing the system mapping and the root software composition.

task, where higher values correspond to more important tasks.

## 3.5 Timing Modeling

The timing constraints of the AEBS are added to the model in Fig. 5 with the help of AUTOSAR-TE in the UML tool Rhapsody.

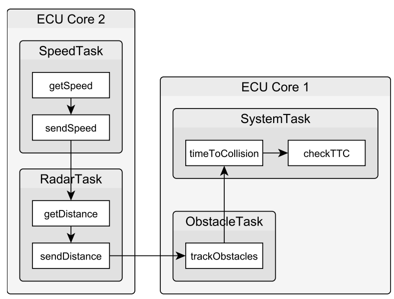


Figure 7: AEBS architecture and end-to-end execution path.

Fig. 8 shows a latency constraint for the *checkTTC* runnable entity of the *DriverWarning* software component (seen at top-right of Fig. 5). An *SwcTiming* is created for each software component in the AEBS, which link to the component's internal be-

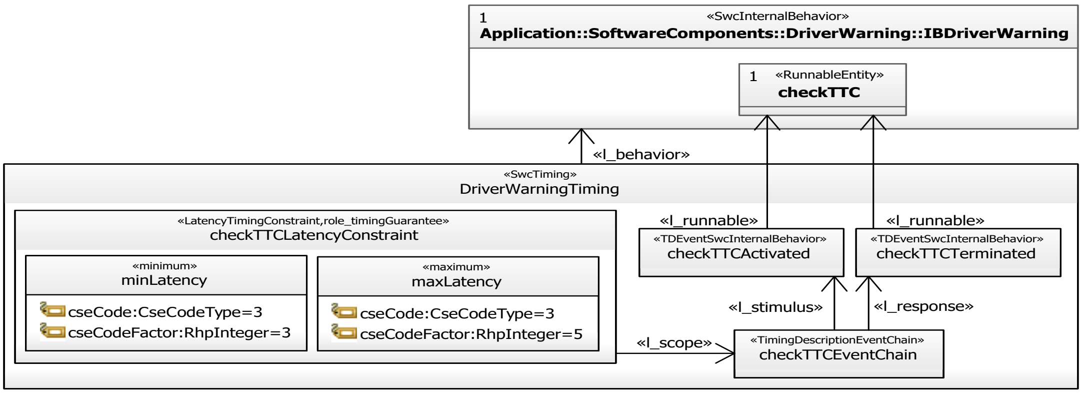

Figure 8: Timing attributes for the *checkTTC* runnable entity.

Table 2: Task properties and mapped runnable entities (configuration for timing analysis).

| Task | Priority | Period | Preemptible | Runnable entities |
|---|---|---|---|---|
| SpeedTask | 0 | 10 ms | Yes | getSpeed<br>sendSpeed |
| RadarTask | 1 | 10ms | Yes | getDistance<br>sendDistance |
| ObstacleTask | 2 | 50ms | Yes | trackObstacles |
| SystemTask | 5 | 50ms | Yes | timeToCollision<br>checkTTC |



havior with the *l_behavior* association. Inside these elements, two *TDEventSwcInternalBehaviors* are defined for each runnable entity (in this case, *checkTTC* of *IBDriverWarning*). The first event highlights the activation of the runnable, while the second highlights the termination. This is defined by setting the tag *tdEventSwcInternalBehaviorType* of the timing event to either *runnableEntityActivated* or *runnableEntityTerminated*. Both these events are now used to form a *TimingDescriptionEventChain*, in which the event chain stimulus is the runnable activation and the event chain response is the runnable termination.

Finally, the core execution time of the runnable *checkTTC* is specified by the *checkTTCLatencyConstraint* that links to its event chain with *l_scope*. The *role_timingGuarantee* stereotype declares that this constraint is the expected execution time instead of a requirement (*role_timingRequirement*). The related timing information can be given as maximum and minimum execution time and is specified by ASAM-CSE[7] codes. The *cseCode* specifies the time base (e.g., $2 = 100\mu s$, $3 = 1ms$ and $4 = 10ms$) and the *cseCodeFactor* determines an integer scaling factor. Thus, in this case, the execution time of the *checkTTC* runnable entity lies between 3ms and 5ms. Note that every runnable entity in the model is similarly specified with their respective timing requirements.

#### 3.5.1 Causal Event Chain Example

Given the timing annotated AUTOSAR-design model, the time critical path of AEBS system is identified. Let us consider  which states that, *new sensor data must influence the TTC computation after a maximum delay of 200ms*. Thereby, the system needs to react fast to sudden speed changes, so the end-to-end response time (delay) from sensor measurements to a reaction needs to be below 200 ms. The end-to-end flow of the AEBS system (cf. Fig. 4), from sensor input to *possible* actuator output, in which the  needs to be satisfied, is an example of a causal event chain (cf.section 2.1.2). This is referred hereafter as **systemEventChain:**. It is depicted by the function path in Fig. 7, namely **getSpeed** → **sendSpeed** → **getDistance** → **sendDistance** → **trackObstacles** → **timeToCollision** → **checkTTC**. This describes the time critical path between the event *requesting the speed of the vehicle (stimulus)* and the event *making available the determined TTC value (response)*.

During timing modeling, this event chain is mod-

[7] https://www.asam.net/

eled as an end-to-end execution path through the system. As this path spans over runnables from different software components, it does not belong to a specific *SwcTiming*. Thus, a *VfbTiming* element is created to contain the execution path event chain. It comprises of a sequence of runnable event chains (e.g. the *checkTTCEventChain* in Fig. 8) to highlight the execution path of runnable entities.

#### 3.5.2 Task Configuration for Timing Analysis

There are four tasks specified in the AEBS model (Table 2), as part of ECU configuration. They are, two sensor tasks namely, *SpeedTask* and *RadarTask*, *ObstacleTask* for the *ObstacleLocation* component and *SystemTask* for the *CollisionDetection* and *DriverWarning* components. Every task is provided with a fixed *OsTaskPriority*, where higher values indicate a more important task and period. Then the runnable entities are mapped to the corresponding task (e.g. runnables *getSpeed* and *sendSpeed* are mapped to the task *SpeedTask* as shown in Table 2). The respective task properties and the mapping of runnables to tasks in their specified order are shown in Table 2. At this point, there is enough information contained in the model to analyze the timing behavior of the AEBS. The model is now exported from the underlying UML representation(in UML tool (IBM Software, 2019)) to an interchangeable ARXML file (AUTOSAR, 2018).

## 4 IMPORTER/EXPORTER FRAMEWORK

An AUTOSAR importer (cf. Fig. 9-(b)), is implemented, as part of a lightweight interfacing tool framework shown in Fig. 9-(a). A prototype of the *importer/exporter* tool (Fig. 9-(a)) is implemented in Java using the Plug-in Development Environment (PDE) of the Eclipse IDE. The tool provides interfaces for creating new model importers, which can be integrated with the tool using the Eclipse extension point mechanism. This avoids tight coupling between the tool itself and the modeling domain interfaces (Noyer et al., 2016b).

In this manner, a new plug-in for the AUTOSAR modeling domain is created, as seen in Fig. 9-(b). The main plug-in classes are contained in the *autosar* package. The class *AutosarImporter* extends the *AbstractImporter* class from the importer/exporter tool, to implement the necessary methods for importing models. It also provides the necessary user interface messages (from *Messages* class) and an option for selecting the source ARXML file. This file is the timing annotated AUTOSAR-based design model exported from UML modeling tool (cf. section 3.3–3.5).

The connection to the *importer/exporter* tool is then established by defining an extension point in the plug-in. The *AutosarImporter* class is hereby specified as the importer class, which will be called when the import process is invoked by the importer/exporter tool. Thereupon, the *AutosarToTimingCAT* class from the *converter.autosar* package is called from the importer class to handle the model transformation of the specified AUTOSAR source model. It contains the Model-to-Model (M2M) transformations (cf. section 5) and references to the used AUTOSAR and timing metamodels. The *AutosarResourceSetImpl* class from the Artop framework is thereby used for handling ARXML files.

## 5 MODEL TRANSFORMATIONS

The AUTOSAR importer imports the timing annotated AUTOSAR-based design model in ARXML format. With this as input, M2M transformations implemented in Atlas Transformation Language (ATL)[8], are invoked for synthesis of a timing analysis model (step (b) in Fig. 1). The ATL implementation in the prototype follows the regular structure of ATL transformations. The source, transformation and target models each have their own separate metamodels, which are each based on a common meta-metamodel, namely ECORE (EMF, 2020). Here the source model (i.e., the timing annotated AUTOSAR-based design model) in ARXML format, conforms with the AUTOSAR metamodel (AUTOSAR, 2018). The target metamodel is a timing metamodel. In this work, a generic timing metamodel developed using the EMF (EMF, 2020) and introduced in (Iyenghar et al., 2016) is employed. This metamodel comprises of a basic set of timing properties needed for performing a timing analysis. This can be termed as a generic metamodel, as it closely adheres with timing models used in several timing validation tools (e.g. SymTA/S).

The M2M transformations are implemented as an ATL module, *Autosar2Timing* in the file *autosarToTiming.atl*. In this module, there are 13 matched rules for all conditional mappings and 2 lazy rules for all unconditional mappings. The matched rules are used for source elements such as model, package and classes. Lazy rules are used for source elements that satisfy specific conditions and must be called explicitly for creating target elements. In addition, 16 helpers are implemented which may be invoked by

[8]http://www.eclipse.org/atl/

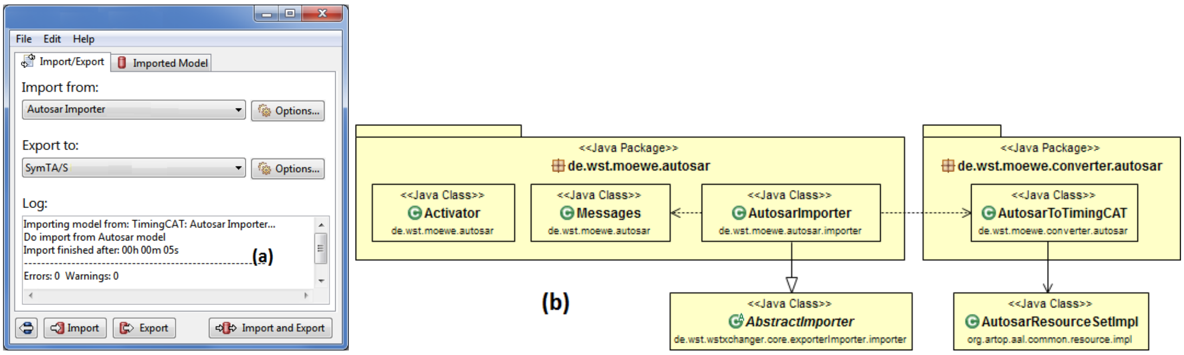


Figure 9: (a) Lightweight importer/exporter interfacing tool framework and (b) AUTOSAR importer class diagram.

the transformation rules. The helpers are used as getter() and setter() methods.

Listing 1: An example of ATL rules.

```
rule AtomicSWC2SWComponent extends
Identifiable2ICATObject{
from
    input : AR!AtomicSwComponentType
to
    output : Timing!SoftwareComponent(
 runnables <- input.internalBehaviors
    ->collect(ib | ib.runnables)
    -> flatten())}
```

A simple example of an ATL matched rule is shown in Listing 1. The `AtomicSWC2SWComponent` rule `extends` the parent rule `Identifiable2ICATObject` and thus, its target pattern is inherited. This means that, the target element `SoftwareComponent` automatically receives the name and description attributes from parent rule (i.e., `Identifiable2ICATObject`-not listed here) . Additionally, it receives the `runnables` attribute specified in the new target pattern, to link to the software component's runnables. The `collect` operation iterates through all internal behavior elements (`ib`) and returns the list of runnables for each. As this statement returns a two-dimensional list, the `flatten` operation ensures that a list directly containing the runnables is returned and assigned to the `runnables` attribute. The rest of the rules are created similarly, such that all the AUTOSAR-TE elements are mapped to the timing metamodel elements. In the end, the ATL execution engine is able to produce a corresponding output timing model from an AUTOSAR input model by use of the ATL file. This model transformation is integrated into the AUTOSAR importer in the extensible importer/exporter tool chain (cf. section 4).

The synthesized AUTOSAR-timing analysis model of the AEBS use case is shown in Fig. 10. The necessary elements for a timing analysis were

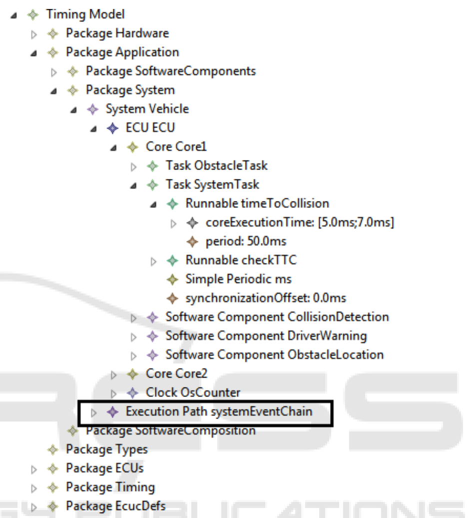


Figure 10: Synthesized timing model of AEBS use case.

extracted from the AUTOSAR design model annotated with timing properties (cf. Fig. 5, 8). As seen in Fig. 10, the AEBS model is structured by different *Packages* and the *System* element contains the complete software and hardware elements in a hierarchy. For example, the runnable *timeToCollision* with its corresponding execution time can be seen, allocated to the *SystemTask*, which is in turn allocated to *Core1* of the ECU. The execution path *systemEventChain* in Fig. 10 is the causal event chain **systemEventChain** described in section 3.5.1.

The resulting timing analysis model is exported to the timing analysis tool SymTA/S for end-to-end timing analysis. This is carried out using the SymTA/S exporter (cf. Fig. 9-(a)), which communicates with the SymTA/S tool using its remote interface. If the export process was successful, a new project is created with the corresponding elements from the timing model in the SymTA/S representation. The same elements from the timing model, annotated with timing properties, are now available inside SymTA/S tool.

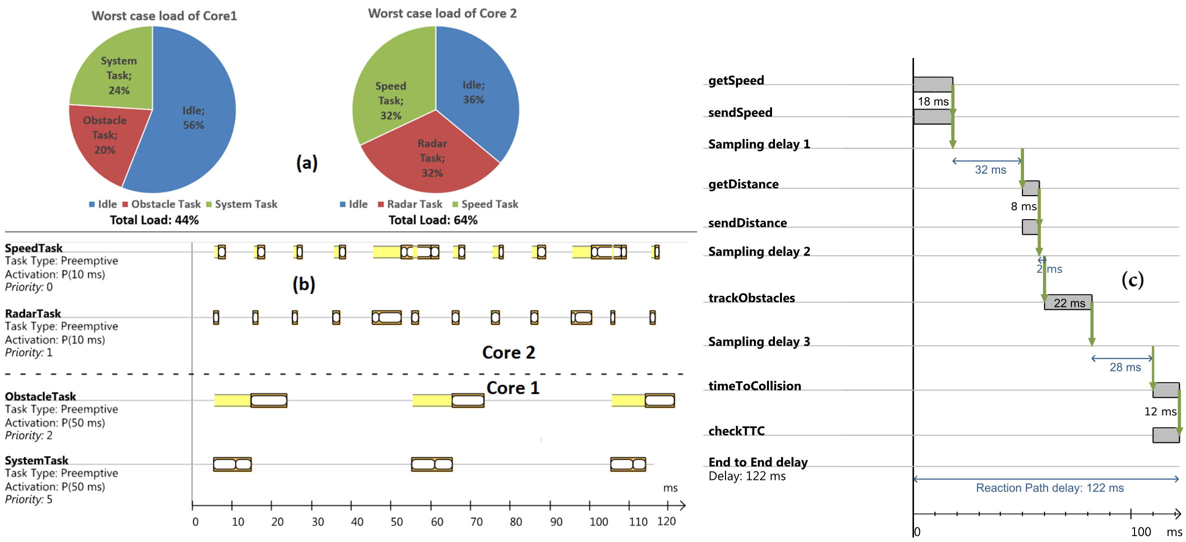


Figure 11: (a) Worst case load of processing cores 1 & 2 (b) Exemplary task scheduling for system distribution analysis and (c) Worst case response time for the causal event chain **systemEventChain** modeled in use case (ref. section 3.5.1).

## 6 RESULTS OF THE TIMING ANALYSIS

Once the timing model (from section 5) is exported to SymTA/S, timing analysis can be carried out and results can be visualized graphically in the tool. For the AEBS use case, the results from processor utilization analysis, worst-case scheduling analysis and end-to-end latency analysis are described below.

Fig. 11-(a) shows the worst case processor workload as pie charts for processing cores 1 and 2. It is seen that, even in the worst case, the system is schedulable and both cores still have resources left (56% idle time for core1 and 36% idle time for core 2). Thus, the timing requirement in section 3.1.2 is satisfied for this set of function execution times.

The detailed scheduling of the AEBS tasks for the cores is seen in a Gantt chart by Fig. 11-(b). It provides an exemplary iteration of the system distribution analysis. The upper half shows the scheduling of *Core2* and the lower one shows the scheduling of *Core1*, which are independent of each other. Each task has different activation periods and priorities and executes a set of runnables, which are depicted as blocks on the timeline. If a task is preempted, e.g., because of a higher priority task, this is expressed by a yellow backdrop in the timeline.

As seen in Fig. 11-(b), the *SpeedTask* tries to execute the *getSpeed* runnable every 10ms. But as the *RadarTask* also executes its *getDistance* function every 10ms and has a higher priority this is delayed until the end of the *RadarTask* execution. The same applies for the *sendSpeed* operation, which is to be executed every 50ms, but has to wait for the *sendDistance* operation. Thus, this provides an extensive overview of the possible scheduling behavior of the software system. It also confirms that the ECU will be able to schedule the system and obstacle tasks every 50ms, and the sensor tasks every 10ms. Hence, the timing requirements to from section 3.1.2 are satisfied.

The worst case response times for end-to-end execution paths as evaluated during timing analysis is shown in Fig. 11-(c). It shows the *systemEventChain* as specified in the AUTOSAR model in a Gantt chart. The runnables of the AEBS contained in the event chain are executed consecutively. This leads to a worst case response time of 122ms, which satisfies the system end-to-end path timing requirement from section 3.1.2.

## 7 CONCLUSIONS

In this paper, we have presented a detailed evaluation of a novel approach in ESE towards automated timing analysis of AUTOSAR-based systems. A specific example of a widely used timing analysis measure for automotive systems namely, end-to-end delay analysis is described with the help of an AUTOSAR-based causal event chain example.

The elaborate use case presented in this paper, was in early design phase without implementation (e.g. no function definitions). However, the timing analysis provided an outlook on the estimated timing behavior of the system. This additional knowledge at an early stage could be very beneficial to the ECU development process. For instance, the additional resources (e.g. processor workload) could be utilized to further extend or improve the functionality of AEBS (e.g. by decreasing the period of the system end-to-end path) or replace the ECU with a slower, cheaper version, to save costs in mass production. At this development stage, such design changes are still possible without major consequences.

## ACKNOWLEDGEMENTS

This work is supported by a grant (id: KF2312004KM4) from BMWi-ZIM co-operation, Germany and carried out in cooperation with Willert Software Tools GmbH and SymtaVision GmbH. The authors thank Stephan Wessels M.Sc (Uni Osnabrueck) and Dr. Arne Noyer (Willert Software Tools) and for their contribution with AEBS use case and importer/exporter tool respectively.

## REFERENCES

Anssi, S., Gérard, S., Kuntz, S., and Terrier, F. (2011). AUTOSAR vs. MARTE for enabling timing analysis of automotive applications. In *International SDL Forum*, pages 262–275. Springer.

AUTOSAR (2017). Specification of timing extensions. https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TPS_TimingExtensions.pdf. Accessed Jan 2020.

AUTOSAR (2018). Release 4.4.0: Methodology and templates. https://www.autosar.org/standards/classic-platform/classic-platform-440/. Accessed Nov 2019.

Becker, M., Dasari, D., Mubeen, S., Behnam, M., and Nolte, T. (2017). End-to-end timing analysis of cause-effect chains in automotive embedded systems. *Journal of Systems Architecture*, 80:104 – 113.

Bhasker, J. (2010). *A SystemC Primer*. Star Galaxy.

Bucaioni, A., Cicchetti, A., Ciccozzi, F., Mubeen, S., and Sjödin, M. (2017). A metamodel for the rubus component model: Extensions for timing and model transformation from east-adl. *IEEE Access*, 5:9005–9020.

EMF (2020). Eclipse modeling framework. https://www.eclipse.org/modeling/emf/. Accessed Jan 2020.

Ficek, C., Feiertag, N., Richter, K., and Jersak, M. (2012). Applying the AUTOSAR timing protection to build safe and efficient ISO 26262 mixed-criticality systems. *Proceedings of ERTS*.

Franco, F. R. and et. al (2016). Workflow and toolchain for developing the automotive software according autosar standard at a virtual-ecu. In *2016 IEEE 25th International Symposium on Industrial Electronics (ISIE)*, pages 869–875.

Hans, B., Rolf, J., and Henrik, L. (2009). Annotation with Timing Constraints in the Context of EAST-ADL2 and AUTOSAR-the Timing Augmented Description Language. In *STANDRTS'09*.

Harbour, M. G., García, J. G., Gutiérrez, J. P., and Moyano, J. D. (2001). Mast: Modeling and analysis suite for real time applications. In *Real-Time Systems, 13th Euromicro Conference on, 2001.*, pages 125–134. IEEE.

Henia, R., Hamann, A., Jersak, M., Racu, R., Richter, K., and Ernst, R. (2005). System level performance analysis – the symta/s approach. *IEE Proceedings – Computers and Digital Techniques*, 152(2):148–166.

IBM Software (2019). Ibm rational rhapsody developer. https://www.ibm.com/software/products/en/ratirhap. Accessed Nov 2019.

Iyenghar, P., Huning, L., and Pulvermueller, E. (2020). Early synthesis of timing models in autosar-based automotive embedded software systems. In *8th International Conference on Model-Driven Engineering and Software Development*.

Iyenghar, P., Noyer, A., Engelhardt, J., Pulvermüller, E., and Westerkamp, C. (2016). End-to-end path delay estimation in embedded software involving heterogeneous models. In *11th IEEE Symposium on Industrial Embedded Systems, SIES, 2016*, pages 183–188.

Iyenghar, P. and Pulvermueller, E. (2018). A model-driven workflow for energy-aware scheduling analysis of iot-enabled use cases. *IEEE Internet of Things Journal*, 5(6):4914–4925.

Kim, J. H., Kang, I., Kang, S., and Boudjadar, A. (2016). A process algebraic approach to resource-parameterized timing analysis of automotive software architectures. *IEEE Transactions on Industrial Informatics*, 12(2):655–671.

Klobedanz, K., Kuznik, C., Thuy, A., and Mueller, W. (2010). Timing modeling and analysis for AUTOSAR-based software development: a case study. In *Proceedings of Conference on Design, Automation and Test in Europe*, pages 642–645. European Design and Automation Association.

Kusano, K. D. and Gabler, H. (2011). Method for estimating time to collision at braking in real-world, lead vehicle stopped rear-end crashes for use in pre-crash system design. *SAE International Journal of Passenger Cars – Mechanical Systems*, 4(1):435–443.

Mubeen, S., Nolte, T., Sjödin, M., Lundbäck, J., and Lundbäck, K.-L. (2019). Supporting timing analysis of vehicular embedded systems through the refinement of timing constraints. *Journal of Software and Systems Modeling*, 18:36–69.

Noyer, A., Iyenghar, P., Engelhardt, J., Pulvermueller, E., and Bikker, G. (2016a). A model-based framework encompassing complete workflow from specification until validation of timing requirements in embedded software systems. *Software Quality Journal*, pages 1–31.

Noyer, A., Iyenghar, P., Pulvermueller, E., Engelhardt, J., and Bikker, G. (2016b). Coupling of timing properties for embedded realtime systems using a hybrid tool integration approach. In *International Conference on Emerging Technologies and Factory Automation (ETFA)*.

Peraldi-Frati, M.-A., Blom, H., Karlsson, D., and Kuntz, S. (2012). Timing modeling with autosar - current state and future directions. In *Design, Automation Test in Europe Conference, DATE*.

Scheickl, O., Ainhauser, C., and Gliwa, P. (2012). Tool support for seamless system development based on autosar timing extensions. In *Proceedings of Embedded Real-Time Software Congress(ERTS)*.

Singhoff, F., Legrand, J., Nana, L., and Marcé, L. (2004). Cheddar: a flexible real time scheduling framework. In *ACM SIGAda Ada Letters*, volume 24-4. ACM.

Sugimoto, Y. and Sauer, C. (2005). Effectiveness estimation method for advanced driver assistance system and its application to collision mitigation brake system. In *19th Int. Tech. Conf. Enhanced Safety Vehicles*.

van der Horst, R. and Hogema, J. (1993). Time-to-collision and collision avoidance systems. In *Proceedings of the 6th ICTCT Workshop*.