# Towards Extraction of Interoperable Timing Models from Component-Based Vehicular Distributed Embedded Systems

Saad Mubeen[†*], Jukka Mäki-Turja[†*] and Mikael Sjödin[*]
[*] *Mälardalen Real-Time Research Centre (MRTC), Mälardalen University, Västerås, Sweden*
[†] *Arcticus Systems AB, Järfälla, Sweden*
[*]{*saad.mubeen, jukka.maki-turja, mikael.sjodin*}*@mdh.se*
[†]{*saad.mubeen, jukka.maki-turja*}*@arcticus-systems.com*

***Abstract*—In order to support the end-to-end timing analysis at various abstraction levels and development phases, the end-to-end timing models should be extracted from models of the applications in such a way that they are interoperable. We discuss the challenges and issues that are faced when the timing models are extracted at various abstraction levels during model- and component-based development of vehicular distributed embedded systems. We also present preliminary guidelines and solutions to address these challenges.**

## I. Introduction

Due to increase in the amount of advanced computer controlled functionality in vehicular distributed embedded systems, the size and complexity of embedded software has drastically increased in the past few years. For example, the amount of software in a modern premium car can exceed 100 million lines of code [1]. Similarly, the software in a heavy truck can consist of as many as 2000 software functions [2]. In order to deal with the software complexity, the research community proposed model- and component-based development of embedded real-time systems by using the principles of Model-Based Software Engineering (MBSE) and Component-Based Software Engineering (CBSE) [3], [4]. This approach allows to capture requirements early during the development, lowers the development cost, enables faster turn-around times in early design phases, allows reusability, supports modeling at higher abstraction levels, and provides possibilities to automatically perform timing analysis; derive test cases; and generate code. MBSE provides the means to use models to describe functions, structures and other design artifacts. Whereas, CBSE supports the development of large software systems by integration of Software Components (SWCs). It raises the level of abstraction for software development and aims to reuse SWCs and their architectures. Within the segment of construction-equipment vehicles and similar segments for heavy special-purpose vehicles, model-based development of software architectures for embedded real-time systems has had a surge the last few years.

Most of the vehicular functions are developed as distributed embedded systems with real-time requirements. This means, the time at which these systems respond to some stimulus is as important as logically correct response. In other words, logically correct but late response may be considered as bad as logically incorrect response. Hence, the providers of these systems are required to ensure that the actions by the systems will be taken at a time that is appropriate to their environment. One way to guarantee that the system will meet all its deadlines is to perform the end-to-end response-time and delay analysis [5], [6]. For this purpose, the end-to-end timing model should be extracted from the software architecture of the system.

### *A. Problem Statement and Paper Contributions*

The existing model- and component-based development approaches for distributed embedded systems in the vehicular domain support the extraction of end-to-end timing models at an abstraction level that is close to the system implementation. As a result, the end-to-end timing analysis cannot be performed at earlier development stages. In order to support the early timing analysis, the timing model should be extracted at higher abstraction levels. It is important to note that several modeling technologies and tools are used at various stages during the development of these systems in the industry. The extracted timing model should be interoperable by the tools that are used at various abstraction levels compared to the level where the timing model is extracted. We discuss the challenges and issues that are faced during the extraction of the timing models from component-based vehicular distributed embedded systems at higher levels of abstractions. The goal is to extract the timing models that can be inter-operated by various tools to enable early end-to-end timing analysis. We also present preliminary guidelines and solutions to deal with these challenges.

### *B. Paper Organization*

In Section II, we present the background and related works. Section III describes the end-to-end timing models. In section IV, we discuss the research challenges. Finally, Section V summarizes the current work.

## II. Background and Related Works

### *A. Abstraction Levels Considered by Various Methodologies*

Various models and methodologies used for the development of vehicular distributed embedded systems consider four *abstraction levels* as shown in Figure 1. Each abstraction level provides a complete definition of the system for a given purpose during the development process.

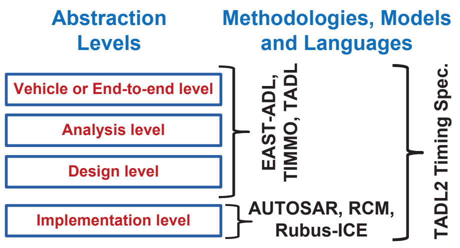

Figure 1. Abstraction levels considered during the development

*1) Vehicle or end-to-end level:* At the vehicle level, requirements, functionality and features of the vehicle are captured in an informal (often textual) and solution-independent way. This level captures the information regarding what the system should do [7]. It is better known as the end-to-end level because features and requirements on the end-to-end functionality of the vehicle are captured in an informal way.

*2) Analysis level:* At the analysis level, the requirements are captured in a formal way. Functionality of the system is defined based on the requirements and features without implementation details. A high-level analysis may also be performed for functional verification.

*3) Design level:* The artifact developed at the analysis level is refined into design functions at the design level. The resulting artifact at this level also contains middleware abstraction and hardware architecture. In addition, software functions to hardware allocation may be present.

*4) Implementation level:* At this level, the design-level artifact is refined to software-based implementation of the system functionality. The artifact at this level is the software architecture in terms of SWCs and their interactions.

## *B. Models and Development Methodologies*

We focus on some of the component technologies that are used for the development of distributed embedded systems in the automotive domain, specifically in the segment of the construction equipment and other heavy vehicles.

Rubus [8] is a collection of methods and tools for model- and component-based development of dependable embedded real-time systems. It is developed by Arcticus Systems in close collaboration with several industrial partners. Rubus is today mainly used for the development of control functionality in vehicles by several international companies, e.g., BAE Systems Hägglunds,[1] Volvo Construction Equipment,[2] Knorr-bremse,[3] and Mecel[4]. The Rubus concept is based around the Rubus Component Model (RCM) and its development environment Rubus-ICE which includes modeling tools, code generators, analysis tools and run-time infrastructure. The overall goal of Rubus is to be aggressively resource efficient and to provide means for developing predictable, timing analyzable and synthesizable control functions in resource-constrained embedded systems. The timing analysis supported by Rubus-ICE includes distributed end-to-end response-time and delay analysis [6]. Rubus methods and tools mostly focus at the implementation level in Figure 1.

AUTOSAR [9] is an industrial initiative to provide standardized software architecture for the development of embedded software. It is used at the implementation level in Figure 1. It describes the software development at a higher level of abstraction compared to RCM. Unlike RCM, it does not separate control and data flows among components within a node. It does not differentiate between the modeling of intra- and inter-node communication which is unlike RCM. The timing model in AUTOSAR is introduced fairly recently compared to that of Rubus. There are some similarities between AUTOSAR and RCM, e.g., the sender receiver communication in AUTOSAR resembles the pipe-and-filter communication in RCM. AUTOSAR is more focussed on the functional and structural abstractions, hiding the implementation details about execution and communication. AUTOSAR hides the details that RCM highlights.

TIMMO [10] is an initiative to provide AUTOSAR with a timing model [11]. It is based around a methodology and TADL [12] language. TADL is used to express timing requirements and constraints. It is inspired by MARTE [13] which is a UML profile for model-driven development of real-time and embedded systems. TIMMO methodology uses EAST-ADL language [14] for structural modeling and AUTOSAR for the implementation. TIMMO and EAST-ADL focus on the top three levels in Figure 1. However, TIMMO methodology uses AUTOSAR at the implementation level. In TIMMO-2-USE project [15], a major redefinition of TADL is done and released in TADL2 specification. TADL2 can specify timing related information at all abstraction levels shown in Figure 1. Most of these initiatives lack the focus on expressing low level details at the higher levels such as linking information in the distributed chains. These details are necessary to extract the end-to-end timing model from the architecture. Furthermore, there is no focus on how to extract this information from the model or perform timing analysis or synthesize the run-time framework. In our view, the end-to-end timing model means extracting enough information from the distributed embedded systems to be able to perform certain type of timing analysis, e.g., end-to-end response-time analysis.

In our previous works [16], [17], we presented a method to extract the end-to-end timing models only at the implementation level. However, we did not consider the aspect of interoperability. On the other hand, this paper focuses on the extraction of the timing models at higher abstraction levels with a focus on interoperable timing models and performing early end-to-end timing analysis.

[1]http://www.baesystems.com/hagglunds
[2]http://www.volvoce.com
[3]http://www.knorr-bremse.com
[4]http://www.mecel.se

## III. End-to-end Timing Model

This model consists of timing properties, requirements and dependencies of all tasks, messages and task chains in the system under analysis. It consists of two sub models.

### A. System Timing Model

It is composed of node and network timing models.

*1) Node timing model:* This model contains node-level timing information. We consider the transactional task model (i.e, tasks with offsets ) introduced in [18]. A node, $\Gamma$, consists of a set of $k$ transactions $\Gamma_1, \ldots, \Gamma_k$. Each transaction $\Gamma_i$ is activated by mutually independent events, i.e., the phasing between the events is arbitrary. The activating events can be a periodic sequence of events with a period $T_i$. In case of sporadic events, $T_i$ denotes the minimum inter-arrival time between two consecutive events.

There are $|\Gamma_i|$ tasks in a transaction $\Gamma_i$. Each task in $\Gamma_i$ may not be activated until a certain time, called an *offset*, elapses after the arrival of an external event. By task activation we mean that the task is released for execution. A task is denoted by $\tau_{ij}$. The first subscript, *i*, specifies the transaction to which this task belongs. Whereas, the second subscript, *j*, denotes the index of the task within the transaction. A task, $\tau_{ij}$, is defined by the following attributes.

- $C_{ij}$ denotes the worst-case execution time of the task.
- $O_{ij}$ denotes the offset of the task.
- $D_{ij}$ specifies the optional deadline of the task.
- $J_{ij}$ denotes the maximum release jitter.
- $B_{ij}$ represents the maximum blocking time which is the maximum time the task has to wait for a resource that is locked by a lower priority task.
- $P_{ij}$ denotes the priority of the task.
- $R_{ij}$ denotes the worst-case response time of the task.

There are no restrictions on offset, deadline or jitter, i.e., they can each be either smaller or greater than the period.

*2) Network timing model:* This model contains network-level timing information within the system. A network consists of a number of nodes that communicate via messages. Currently, we focus on the Controller Area Network (CAN) and its higher-level protocols, e.g., CANopen. However, this model can be easily extended to accommodate other automotive network protocols. Each message $m$ has the following attributes.

- $ID_m$ denotes a unique identifier.
- $P_m$ denotes unique priority.
- $C_m$ specifies the transmission time.
- $Transmission\_Type$ shows whether the message is periodic (P), sporadic (S) or mixed (both P and S).
- $J_m$ denotes the release jitter. Usually, it is inherited from the task that queues $m$.
- $s_m$ denotes the data payload in each message. It ranges from 0 to 8 bytes in a CAN message.
- $T_m$ specifies the period of a message in the case of periodic transmission. For a sporadic message, $MINT_m$ is used which is the minimum time that should elapse between the transmission of any two messages. For a mixed message, both $T_m$ and $MINT_m$ are specified.
- $B_m$ denotes the maximum blocking time during which $m$ can be blocked by the lower priority messages.
- $R_m$ denotes the worst-case response time.

### B. System Linking Model

The chains of components (tasks at run time) can be distributed over more than one node in a distributed embedded system. A task chain consists of a number of tasks that are in a sequence and have one common ancestor. Each task may receive a trigger, a data or both from its predecessor. A chain is said to be a trigger chain if first task in the chain is triggered by independent clock/event, while the rest of the tasks are triggered by their predecessors. Each task in a data chain is triggered independently. A mixed chain is a combination of trigger and data chains. The chain types in the modeled application must be unambiguously identified because each type of chain is analyzed using different type of the end-to-end timing analysis. Two neighboring tasks in a distributed chain may reside on two different nodes, while the nodes communicate with each other via network. The end-to-end timing model should also contain the linking information among all tasks and messages within the chain. All mapping and linking information of distributed chains is extracted into the system linking model.

## IV. Research Challenges and Issues

Ideally, the modeling technology used for the development of the systems should facilitate unambiguous extraction of the timing models with ease. The modeling technologies that are used at the implementation level such as RCM and AUTOSAR support the extraction of the end-to-end timing models. However, the modeling technologies that are used at the design or higher levels such as EAST-ADL, TIMMO and TADL do not support complete and unambiguous extraction of these models. The tool chain that uses all these technologies for the development of vehicular distributed embedded systems lacks the support for end-to-end timing analysis at higher levels. As a result, it may be too late to perform the timing analysis in some cases.

In this work, we focus on the design level within the context of this problem. We discuss some of the issues that hinder the extraction of the end-to-end timing model at the design level. We consider the modeling support of EAST-ADL, TIMMO and TADL at the design level. Whereas, the modeling support of RCM is considered at the implementation level. However, these issues are equally applicable for other modeling technologies at these levels.

*1) Annotation and extraction of timing parameters:* The timing information expressed with the models and tools used at higher levels (with respect to the implementation level) is not enough to extract the end-to-end timing model.

For example, one of the EAST-ADL based tools[5] used in the construction-equipment vehicles industry is able to specify only one timing parameter on the SWCs at the design level, i.e., the period of the component. Clearly, this information is not enough to perform the end-to-end timing analysis. TADL2 can specify timing requirements, constraints and properties at the design level in the EAST-ADL and AUTOSAR based development. However, it lacks the expression of some timing parameters, e.g., transmission type, jitter, and priority which are needed to perform the end-to-end timing analysis. In short, the models and tools used at the design and higher levels need to be more expressive to support the extraction of interoperable timing models.

*2) Control and data flows:* One of the main challenges in the extraction of the timing model is the lack of clear separation between the control and data flows at the design and higher levels. At the implementation level, e.g. in RCM, these flows are clearly separated from each other by means of trigger and data ports as shown in Figure 2 (b). However, at the design level, the SWCs communicate via flow ports which can be interpreted as a data or trigger ports as shown in Figure 2 (a). Moreover, the component can be triggered via specified constraints, modes, or internal behavior of the component. The two flows should be clearly and separately captured in the end-to-end timing model because the type of the timing analysis depends upon it. For example, it is meaningful to specify the Age and Reaction delay constraints, introduced in TADL and also supported in RCM, on the data flows rather than on trigger flows. The analysis engines can calculate these delays only when they are specified on the data flows.

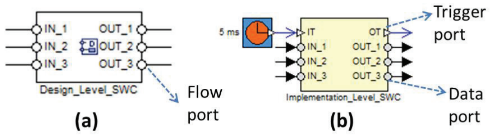

Figure 2. Model of the SWC at (a) design level, (b) implementation level

*3) Identification of chain types:* Since the control and data flows are clearly separated at the implementation levels, e.g., in RCM, the chain types can be easily identified as shown in Figure 3. Due to no clear separation between these flows at the design level, virtually it is not possible to identify the type of a chain. At the design level, a chain can be interpreted as a trigger or data chain. Without any explicit trigger information, the end-to-end timing model remains incomplete. The end-to-end timing analysis explicitly depends upon the type of the chain. This is because the trigger chains are analyzed using end-to-end response-time analysis, whereas, the data and mixed chains are analyzed using the end-to-end delay analysis [6].

*4) Execution order and priorities of components:* The execution order of SWCs must be captured in the end-to-end timing model. At the implementation level, e.g. in RCM, the execution ordering is determined by means of connecters, precedence constrains, and more importantly priorities. Although, the precedence constraints can be specified, there is no notion of priorities at the design level. Therefore, at the design level, it is very difficult to extract the execution order in these systems that contain branches (e.g., in the case of multiple producers and a single consumer).

*5) Extraction of linking information in distributed chains:* The linking model is an important part of the end-to-end timing model. The linking information within a chain is vital to identify the relation among the components that may be distributed over several nodes in a distributed embedded application. This information is modeled very explicitly at the implementation level, e.g., the signal database object handles this information in RCM. Hence, it can be easily extracted into the end-to-end timing model. However, this information is either very implicit or does not exist at the design level. Since, EAST-ADL relies on AUTOSAR at the implementation level, the linking model of chains at the design level is left to be handled by the Virtual Function Bus (VFB). The VFB is a concept in AUTOSAR to handle the distribution of SWCs, their virtual integration and communication. In fact, its purpose is to hide low-level communication details such as the linking information.

*6) Information duplication and ambiguity:* The modeling at the implementation level is unambiguous, and hence, the extracted timing model is also unambiguous. For example, RCM does not allow illogical operations such as specifying more than one clock on the same component without any synchronization or merge operation. However, these restrictions are not present at the design level, e.g., more than one execution time or periodic constraint can be specified on a single SWC in EAST-ADL and TADL. Similarly, if data age and reaction constraints are wrongly specified then the development environment does not complain about it. As a result, the extracted timing model may have redundant or erroneous information. Information duplication can lead to inconsistency in the timing model.

## V. Summary of Current Work

Currently, we focus on extracting interoperable end-to-end timing models at the design level. Later, we will move up to the analysis level. There are two different approaches to deal with these challenges. The first approach is to extend and improve the design-level models, languages and tools in such a way that the timing models can be completely and unambiguously extracted. Moreover, the extracted models can be operated by different models and tools especially at the implementation level. The only problem with this approach is that it requires collaboration among a number of tool suppliers and stake holders. This, in turn, raises other types of challenges and limitations. The second approach is to develop the execution-level modeling technology-dependent

[5]For IP protection, the name of the tool is not specified.

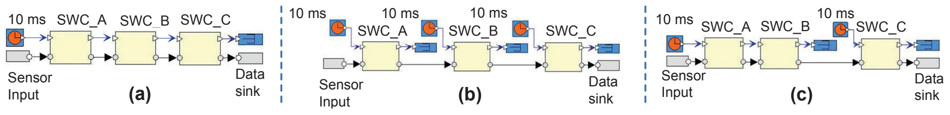

Figure 3. Example of (a) Trigger chain, (b) Data chain, and (c) Mixed chain.

interpretation of the design level. For example, developing the Rubus interpretation of EAST-ADL (this is an ongoing work). It is important to note that this interpretation can be a subset of the full expressiveness of EAST-ADL. No doubt, this may result in a number of these interpretations by several other modeling technologies. This can be a good solution as long as these interpretations support unambiguous extraction of interoperable end-to-end timing models.

Now, we propose some guidelines to deal with the discussed challenges. In order to clearly identify the trigger and data flows, we assume implicit trigger when data is received at any input flow port of a component. For example, the component in Figure 2(a) may receive three individual triggers when data is separately received at the three input flow ports. In order to deal with multiple implicit triggers (corresponding to multiple flow ports), we can introduce a synchronization object at the design level or import it form RCM at the implementation level. This object gets the multiple triggers at input, synchronizes them, and produces a single trigger that can be used to trigger the design-level component with multiple flow ports. In this way, the timing model extracted at the design level can be easily operated by the analysis engines at the implementation level.

If the worst-, best- and average-case execution times are not available at the design level, they can be estimated at the implementation level by the analysis engines provided the extracted timing model is interoperable.

Unless explicit trigger requirements are not specified at the design level, we assume each design-level software function to be triggered by independent clock. Hence, the chains are assumed to be data chains. Moreover, we assume that the execution order of the design-level components is specified, otherwise, we make implicit assumption about the precedence constraints along the chain. That is, each component is assumed to execute only after the successful execution of the preceding component in the chain, unless specified otherwise. This means that the data provider component is assumed to be always executed before the data receiver component. Since, this assumption fixes the execution order, it is safe to assume the priorities of the components are equal within the chain.

The analysis engines treat these chains in two ways. First, the chains are considered trigger chains and the end-to-end response-time analysis is performed. Second, the chains are treated as data and mixed chains and the end-to-end delays are calculated along with the end-to-end response times.

## ACKNOWLEDGEMENT

This work is supported by the Swedish Research Council (VR) and Swedish Knowledge Foundation (KKS) within the projects SynthSoft and FEMMVA repectively. We thank our industrial partners Arcticus Systems and Volvo CE, Sweden.

## REFERENCES

[1] R. N. Charette, "This Car Runs on Code," *Spectrum, IEEE*, vol. 46, no. 2, 2009, http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code.

[2] Keynote talk, EAST-ADL Open Workshop, Gothenberg, Oct., 2013.

[3] T. A. Henzinger and J. Sifakis, "The Embedded Systems Design Challenge," in *Proceedings of the 14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science*. Springer, 2006, pp. 1–15.

[4] I. Crnkovic and M. Larsson, *Building Reliable Component-Based Software Systems*. Norwood, MA, USA: Artech House, Inc., 2002.

[5] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocess. Microprogram.*, vol. 40, pp. 117–134, April 1994.

[6] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study," in *Computer Science and Information Systems, vol. 10, no. 1, pp 453-482, January 2013. ISSN: 1361-1384*.

[7] "Hans Blom et. al. EAST-ADL- An Architecture Description Language for Automotive Software-Intensive Systems. White paper, Version M2.1.10, 2012, http://www.maenad.eu."

[8] "Rubus models, methods and tools," http://www.arcticus-systems.com.

[9] "AUTOSAR Techincal Overview, Version 2.2.2. AUTOSAR – AUTomotive Open System ARchitecture, Release 3.1, The AUTOSAR Consortium, Aug., 2008," http://autosar.org.

[10] "TIMMO Methodology, Version 2, Deliverable 7, Oct. 2009."

[11] "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," 2012.

[12] "TADL: Timing Augmented Description Language, Version 2, Deliverable 6, October 2009," The TIMMO Consortium.

[13] "The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems, 2010." OMG Group, January 2010. [Online]. Available: http://www.omgmarte.org/

[14] "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.

[15] "TIMMO-2-USE," http://www.timmo-2-use.org/.

[16] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Extraction of end-to-end timing model from component-based distributed real-time embedded systems," in *TiMoBD workshop located at Embedded Systems Week*. Springer, Oct. 2011, pp. 1–6.

[17] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Communications-Oriented Development of Component- Based Vehicular Distributed Real-Time Embedded Systems," *Journal of Systems Architecture, http://dx.doi.org/10.1016/j.sysarc.2013.10.008*, Oct. 2013.

[18] K. Tindell, "Adding Time-Offsets to Schedulability Analysis," University of York, Tech. Rep., January 1994.