Softw Syst Model (2019) 18:39–69
https://doi.org/10.1007/s10270-017-0579-8

THEME SECTION PAPER

Supporting timing analysis of vehicular embedded systems through the refinement of timing constraints

Saad Mubeen[1] · Thomas Nolte[1] · Mikael Sjödin[1] · John Lundbäck[2] · Kurt-Lennart Lundbäck[2]

Received: 20 March 2016 / Revised: 31 December 2016 / Accepted: 10 January 2017 / Published online: 31 January 2017

Abstract The collective use of several models and tools at various abstraction levels and phases during the development of vehicular distributed embedded systems poses many challenges. Within this context, this paper targets the challenges that are concerned with the unambiguous refinement of timing requirements, constraints and other timing information among various abstraction levels. Such information is required by the end-to-end timing analysis engines to provide pre-run-time verification about the predictability of these systems. The paper proposes an approach to represent and refine such information among various abstraction levels. As a proof of concept, the approach provides a representation of the timing information at the higher levels using the models that are developed with EAST-ADL and Timing Augmented Description Language. The approach then refines the timing information for the lower abstraction levels. The approach exploits the Rubus Component Model at the lower level to represent the timing information that cannot be clearly specified at the higher levels, such as trigger paths in distributed chains. A vehicular-application case study is conducted to show the applicability of the proposed approach.

Keywords Distributed embedded systems · Component-based development · Timing model · Component model · End-to-end timing analysis

1 Extended version

This paper extends our previous work [1] where we have discussed the refinement of two end-to-end delay constraints from higher to lower abstraction levels during model- and component-based development of vehicular embedded systems. As a proof of concept, we have selected the Timing Augmented Description Language (TADL2) [2] at the higher abstraction levels, whereas at the lower level (implementation), we have selected the Rubus Component Model (RCM) [3] which is already used in the vehicle industry for the development of control functionality in vehicular embedded systems. The work in this paper generalizes our previous work [1] by refining various other types of timing constraints (18 in total) from the higher to lower abstraction levels. Once again, we consider TADL2 and RCM for the proof of concept. We also conduct a detailed automotive-application case study to validate our refinement and timing model representation approach.

2 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

Communicated by Dr. F. Ciccozzi, J. Carlson, P. Pelliccione and M. Tivoli.

✉ Saad Mubeen
saad.mubeen@mdh.se

Thomas Nolte
thomas.nolte@mdh.se

Mikael Sjödin
mikael.sjodin@mdh.se

John Lundbäck
john.lundback@arcticus-systems.com

Kurt-Lennart Lundbäck
kurt.lundback@arcticus-systems.com

[1] Mälardalen University, Västerås, Sweden

[2] Arcticus Systems AB, Järfälla, Sweden

embedded software in heavy vehicle architectures such as a modern truck may consist of as many as 2000 software functions that may be distributed over 45 Electronic Control Units (ECUs) [4]. In order to deal with the software complexity, the research community has 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) [5,6]. This approach is intended to capture requirements early during the development,[1] lower development cost, enable faster turnaround times in early design phases, increase reusability, support modeling at higher abstraction levels and provide 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. In contrast, CBSE supports the development of large software systems by integration of software components. It raises the level of abstraction for the software development and aims to reuse software components and their architectures. Model- and component-based development of software architectures for vehicular embedded systems has had a surge in the last few years. It is evident from several large European research projects that have run in close collaboration between academia and the industry [7–11].

2.1 Problem statement

Most of the vehicular functions are developed as distributed embedded systems with real-time requirements specified on them. This means that the providers of the systems are required to ensure that logically correct actions are taken by the systems at times that are appropriate to their environment (i.e., the timing requirements are satisfied). One way to guarantee that the system meets its timing requirements is to perform pre-run-time analysis of it, e.g., *end-to-end response-time and delay analysis* [12,13]. Such analysis can validate the timing requirements without performing exhaustive testing. Note that the timing behavior of an individual task or a message can be determined by calculating its response time. The response time of a task or a message is defined as the amount of time elapsed between its activation and completion or reception respectively. Often, vehicular embedded systems are modeled with task chains. A task chain consists of a number of tasks that are in a sequence and have one common ancestor. Each task may receive an activation trigger, a data or both from its predecessor. Any two neighboring tasks in a chain may reside on two different nodes, while the nodes communicate with each other via network messages. In this case, the messages are part of the task chain. The timing behavior of the task chain is determined by calculating its end-to-end response time and/or delays. The end-to-end response time 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. If the tasks within a chain are activated by independent sources (e.g., clocks), then different types of end-to-end delays are also calculated to determine the timing behavior of the chain (Sect. 4.5.3 provides a detailed discussion on the end-to-end delays).

In order to perform the timing analysis of the system, its *end-to-end timing model* should be available. The end-to-end timing model consists of the information containing timing properties, requirements, dependencies, control and data flows concerning all tasks, messages and task chains in the system. Based on this information, the timing analysis can predict the execution behavior of the system with respect to end-to-end timing. We refer the reader to [14] for details about the end-to-end timing models.

The majority of existing approaches for component-based vehicular distributed embedded systems support the representation of such timing models at an *abstraction level* that is close to their implementation [7,13–17]. An abstraction level provides a complete definition of the system for a given purpose during the development process. There are a few works including [18,19] that support the end-to-end timing analysis at the higher abstraction levels. It is shown in [18] that the timing analysis supported by the existing approaches at the higher abstraction levels cannot predict the end-to-end timing behavior of the system with a high *precision*. This is because the analysis is often not based on the actual implementation of the system. The precision of the analysis refers to how accurately the analysis results capture the end-to-end timing behavior of the final systems. The low-precision analysis results are overestimated due to educated guesses by the expert integrators about missing timing information at the higher abstraction levels and earlier phases during the development, whereas the high-precision analysis results do not include such overestimations. We assume, irrespective of the precision, the analysis results are not optimistic (underestimated). Recently, one of the main focuses of several international initiatives, involving both academia and industry, has been on supporting the timing analysis at various abstraction levels and development phases [8–11].

Representation of the timing model at the higher abstraction levels is challenging mainly because not all timing information is available at the higher levels. Moreover, a mismatch and incompatibilities among various methodologies, languages and tools that are used in different development phases also add to the complexity of representing the timing model. Since complete timing information may not be avail-

[1] The aspect "during the development" refers to the abstraction levels during the software development of vehicular embedded systems. Note that this aspect does not refer to the overall automotive development process.

able at the higher levels, the timing analysis results can be overestimated based on pessimistic assumptions. Hence, the analysis results may not represent accurate timing behavior of the final system. However, these results can provide useful information to the developer to guide her in performing model refinements earlier during the development [18].

We envision the representation of the end-to-end timing models and support for a high-precision end-to-end timing analysis at the higher levels of abstraction to be the state of the practice in the future. We believe the timing information will be formally modeled at the higher abstraction levels in the vehicle industry. In that case, we need to extract the specified timing information at the higher abstraction levels and connect it to the implementation to generate the end-to-end timing model. Otherwise, it can be too late to extract the timing model at lower abstraction levels that are close to the system implementation.

We have experienced that the timing information is modeled at higher abstraction levels in the vehicle industry. This may be carried out using, e.g., the SysML language [20]. However, it is done mostly in an informal and textual way, which cannot be used for any formal timing analysis. Today, the TADL2 language [2] provides the only viable formal method for modeling of timing information using timing constraints at various abstraction levels in the vehicle domain. This is evident from the fact that TADL2 has recently provided the timing model to the EAST-ADL language [21] and AUTOSAR [7]. EAST-ADL is an architecture description language in the automotive domain. The industrial members in the EAST-ADL association and the consortium that has developed the TADL2 language are shown in Fig. 3. AUTOSAR supports the development of standardized software architectures in the automotive domain. The AUTOSAR consortium consists of over 300 industrial partners including original equipment manufacturers (OEMs), tier-1 and tier-2 suppliers in the automotive domain.

In order to represent the complete end-to-end timing model and perform a high-precision end-to-end timing analysis, TADL2 has to be combined with a lower abstraction level execution modeling technology such as RCM. Since the TADL2 language and corresponding timing extensions in EAST-ADL and AUTOSAR have been introduced fairly recently, it may take some time for the automotive industry to use TADL2 for the development of vehicles. Note that TADL2 has been successfully employed for the development of some validators and prototypes in the automotive industry, e.g., electromechanic systems, brake-by-wire systems, steer-by-wire systems and adaptive cruise control systems [9]. We hope that the industry will start using TADL2 very soon. If they do so, we can reuse that information to perform a high-precision end-to-end timing analysis at the higher abstraction levels.

2.2 Paper contributions

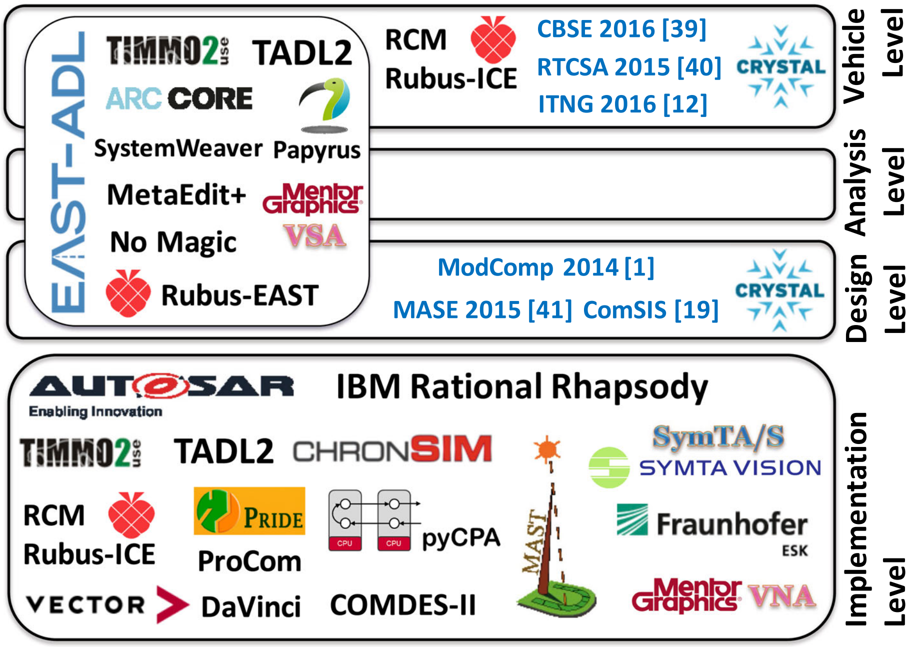
In this paper,[2] we propose an approach to represent the end-to-end timing models at a higher abstraction level compared to the level where the software architecture is implemented. At the higher level, this approach provides a representation of the timing information on the system models that are developed with the EAST-ADL language using the TIMMO methodology [22] and annotated with timing information using TADL2. At the lower level, the approach exploits RCM and its tool suite Rubus-ICE [23] to represent the timing information that cannot be clearly specified at the higher level, e.g., control paths in distributed chains. However, it is not straightforward to combine TADL2 with RCM due to various challenges such as providing an unambiguous refinement of the TADL2 timing constraints in RCM and supporting an unambiguous representation of the control and data flows at the higher level (Sect. 5 discusses these points in detail). The main focus of this paper is to deal with these challenges. In order to show a proof of concept, we model a vehicular application at the higher level and refine it along with the timing information to the lower level. We then perform the end-to-end timing analysis of the system to validate the timing constraints specified at the higher level. The main contributions in this paper are listed as follows.

1. Interpretation of all TADL2 timing constraints in RCM.
2. Extensions to RCM for unambiguous refinement of the TADL2 timing constraints.
3. Representation of the end-to-end timing information at the higher abstraction level.
4. Performing a vehicular-application case study to show the applicability and usability of the proposed refinement and timing model representation approach.

We choose RCM instead of AUTOSAR at the lower abstraction level for two reasons. Although AUTOSAR provides a timing model in its current specification [7], it still lacks a way to specify a few low-level details which are needed to perform the end-to-end timing analysis, e.g., control flow is not specifiable in an unambiguous way. The other reason is that the implementations built with RCM have relatively smaller run-time footprints, i.e., timing and memory overheads (Sect. 3.2 discusses this point in detail). The work in this paper brings us one step closer to the goal of developing a seamless tool chain for model-based development of vehicular embedded systems and supporting inter-operation of various modeling and analysis tools including the AUTOSAR-based tool chain [10].

[2] A version of this paper is provided as an internal report for industrial referencing at http://www.es.mdh.se/publications/3545-. It does not represent a published work.

Fig. 1 Abstraction levels considered during the development



2.3 Paper layout

The rest of the paper is organized as follows. Section 3 discusses the background and related work. Section 4 discusses the refinement of the TADL2 timing constraints in RCM. Section 5 discusses other challenges and corresponding solutions. Section 6 provides a case study. Finally, Sect. 7 concludes the paper and presents future work.

3 Background and related work

There are several frameworks that support the modeling of timing information such as AADL [24], SCADE [25], MARTE [26], MAST [27], SysML, CHESS [28,29]. In this paper, we only target the vehicular domain, especially the segment of construction equipment and other heavy road vehicles, where the main focus is on EAST-ADL [21], EAST-ADL-like models[3] and AUTOSAR [7]. In addition, Rubus [30] is used complementary to EAST-ADL.

3.1 EAST-ADL

EAST-ADL is a domain-specific architecture description language that targets the development of software architectures for automotive embedded systems. Figure 3 depicts the industrial members in the EAST-ADL association that have been involved in the development and extension of the language. EAST-ADL is inspired by SysML which is a system modeling language used for systems engineering [31]. It is mapped to several automotive standards including ISO26262 [32] for functional safety and AUTOSAR for the implementation and run-time execution of the software architecture. It defines a top-down development methodology that advocates the separation-of-concerns principle by defining various abstraction levels for the development of vehicle software. Each abstraction level provides a complete definition of the system for a specific purpose. Figure 1 shows the abstraction levels along with various methodologies, models, languages and tools used at each level. This figure also depicts several recent works within the scope of this paper.

3.1.1 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. Within the segment of construction equipment and other heavy road vehicles, this abstraction level is better known as the end-to-end level because the features and requirements on the end-to-end functionality of the machine or vehicle are captured in an informal way.

3.1.2 Analysis level

At the analysis level, the requirements are formally captured in an allocation-independent way. Functionality of the system is defined based on the requirements and features without

[3] For example, SE Tool and SystemWeaver (http://www.systemweaver.se).

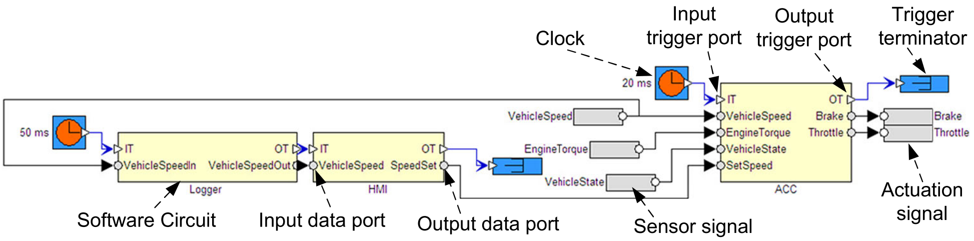


Fig. 2 Example of the software architecture of a system modeled in RCM

implementation details. A high-level analysis may also be performed for functional verification, e.g., consistency analysis.

3.1.3 Design level

The artifacts at this level are developed in an implementation-independent way. These artifacts are refined from the analysis level artifacts. In addition to the software architecture composed of the design-level software components, this level also contains the middleware abstraction, the hardware architecture and the software functions-to-hardware allocation.

3.1.4 Implementation level

At the implementation level, the design-level artifacts are refined to a software-based implementation of the system functionality. At this level, the EAST-ADL methodology describes the system in terms of AUTOSAR elements and their integration. However, in this work, our focus is on using RCM and Rubus-ICE at the implementation level. Hence, the artifact at this level consists of a software architecture of the system that is defined in terms of Rubus software components and their interactions.

In this work, we focus on the representation of the end-to-end timing models mainly at the design and implementation levels.

3.2 Rubus Component Model (RCM) and Rubus-ICE

Rubus is a collection of methods and tools for the model- and component-based software development of dependable embedded real-time systems. It is developed by Arcticus Systems[4] in close collaboration with several academic and industrial partners. It has been used in the vehicle industry for over 20 years. Rubus is today mainly used for the development of control functionality in vehicles by several international companies, e.g., BAE Systems Hägglunds,[5] Volvo Construction Equipment,[6] Knorr-Bremse,[7] Mecel[8] and Hoerbiger.[9] The Rubus concept is based on RCM and its development environment, Rubus-ICE, which includes modeling tools, code generators, analysis tools and run-time infrastructure. Rubus also includes a real-time operating system which has already been certified in the ISO 26262:2011[10] safety standard according to ASIL D. 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 the end-to-end response-time and delay analysis [13]. Rubus methods and tools focus on the implementation level and are used complementary to the EAST-ADL methodology at the top three levels.

Rubus enables the designer to graphically describe systems as interconnected components. These interconnected components, following a hardware paradigm called Software Circuits (SWCs), define the structure of the application system that can be analyzed and synthesized entirely within the Rubus environment. An SWC is the lowest level hierarchical element in RCM. It encapsulates basic functions. An SWC has the run-to-completion semantics, i.e., upon receiving a trigger (activation) on its trigger input port the SWC reads data from its data input ports, executes its functionality, provides data on its data output ports and finally produces a trigger on its trigger output port. Figure 2 shows an example of the software architecture in RCM that is composed of SWCs, interconnections between SWCs and interactions of SWCs with external events and actuators with regard to both data and triggering.

The Rubus run-time framework maps the SWCs to tasks which are run-time entities. Each external event trigger in the software architecture defines a task. The SWCs connected through the chain of triggered SWCs (trigger chain) are allocated to the corresponding task. All clock triggered chains are allocated to an automatically generated static schedule that fulfills the precedence order and other temporal require-

[4] http://www.arcticus-systems.com.

[5] http://www.baesystems.com.

[6] http://www.volvoce.com.

[7] http://www.knorr-bremse.com.

[8] http://mecel.se.

[9] http://www.hoerbiger.com.

[10] http://www.iso.org/iso/catalogue_detail?csnumber=43464.

Fig. 3 Industrial members/partners in the EAST-ADL, TIMMO and TIMMO2USE consortia/projects

ments. Within the trigger chains, inter-SWC communication is aggressively optimized to use the most efficient means of communication for each communication link. For example, there is no use of semaphores in point-to-point communications within a trigger chain. Another example is sharing of memory buffers between ports. This means that a buffer can be shared between two ports belonging to different SWCs if it can be guaranteed that these ports will never use the buffer space at the same time. This is applicable in the case of a trigger chain because a task early in the chain can never be active at the same time as a task late in the chain (assuming that the deadline of each task is smaller than or equal to its period). Allocation of SWCs to tasks and construction of schedule can be subject to different optimization criteria to minimize, e.g., response times for different types of tasks, or memory usage. The run-time framework executes all tasks on a shared stack, thus eliminating the need for static allocation of stack memory to each individual task.

3.3 AUTOSAR

AUTOSAR [7] is an industrial initiative to provide a standardized software architecture for the development of embedded software. It is used at the implementation level in Fig. 1. It describes the software development at a higher abstraction compared to RCM. Unlike RCM, it does not separate control and data flows among components within a node. AUTOSAR does not differentiate between the modeling of intra- and inter-node communication which is unlike RCM. The timing model in AUTOSAR has been introduced fairly recently compared to that in RCM. 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.

3.4 TIMMO, TIMMO2USE, MARTE, TADL and TADL2

TIMMO [8] is an initiative to provide AUTOSAR with a timing model [33]. It is based on a methodology and the TADL language [34] which is used to express timing requirements and constraints. It is inspired by MARTE [26] which is a UML profile for model-driven development of real-time and embedded systems. The TIMMO methodology uses the EAST-ADL language for structural modeling and AUTOSAR for the implementation. TIMMO and EAST-ADL focus on the top three levels in Fig. 1. TADL is redefined and released in the TADL2 specification of the TIMMO2USE project [9]. TADL2 can specify timing-related information at all abstraction levels shown in Fig. 1. The industrial members in the TIMMO and TIMMO2USE projects are shown in Fig. 3. Most of these initiatives lack the support for expressing the low-level details at the higher levels such as linking information in distributed chains. It is important to extract these details from the software architecture for the representation of the end-to-end timing model. These initiatives do not provide sufficient support for representing this information or performing the end-to-end timing analysis. In our view, the end-to-end timing model includes enough information from the system to be able to perform the end-to-end response-time and delay analysis.

3.5 Other related models and approaches

There are several other related component models and modeling approaches such as COMDES-II [16], ProCom [15] and TECS [35]. ProCom supports timing analysis at the implementation level [36]. According to [13], the analysis supported by ProCom is not performed with such a high precision as it is done in Rubus-ICE. To the best of our knowledge, none of these models support the representation of the end-to-end timing models at the higher abstraction levels. This is because these models are developed to model the software architecture only at the implementation level. These models rely on EAST-ADL at the higher abstraction levels. The end-to-end timing models cannot be completely represented at the higher abstraction levels of EAST-ADL mainly for two reasons: (1) EAST-ADL does not differentiate between the control and data flows and (2) EAST-ADL cannot express the low-level details at the higher levels such as linking information in distributed chains [14].

There are middleware development technologies such as real-time CORBA, minimum CORBA and CORBA lightweight services for distributed embedded systems [17]. CUTS [37], based on CORBA, provides an execution modeling support to validate quality-of-service properties of the system. The downside of using CORBA-based development is that the run-time framework is heavyweight. Hence, it is not suitable for resource-constrained embedded systems that require a small run-time footprint. On the other hand, RCM has a small run-time footprint.

3.6 Modeling tools

DaVinci[11] is a tool for the software development of AUTOSAR applications. However, this tool does not support the representation of the end-to-end timing models at the higher abstraction levels. The Palladio tool[12] allows for modeling of the software architecture and its analysis based on several quality attributes including response times. However, this tool does not support the end-to-end timing analysis [13,38]. The refinement of timing constraints and representation of the end-to-end timing models to facilitate such analysis is the main focus (and contribution) of our work. There are several other tools that support modeling of the systems using the methodology shown in Fig. 1, e.g., Papyrus, Mentor Graphics VSA, Rubus-EAST, EATOP, MetaEdit+, Enterprise Architect, No Magic, System Weaver and SE Tool to name a few [39]. These tools are usable at the first three levels in Fig. 1. None of these tools support the representation and refinement of the end-to-end timing models from the higher levels to the implementation level.

3.7 Authors' previous work

In our previous work [14,40], we have presented a method to represent the end-to-end timing models at the implementation level. However, this method is not applicable at the higher abstraction levels. We have recently targeted the vehicle level by developing a modeling technique (denoted by CBSE2016 [41] in Fig. 1). We have developed a method to extract the end-to-end timing models from the extended models of legacy systems (previously developed) to support the end-to-end timing analysis at the vehicle level (denoted by RTCSA2015 [19] in Fig. 1). Moreover, we have developed a method to refine timing requirements using early timing analysis at the vehicle level (denoted by ITNG2016 [18] in Fig. 1). Note that these techniques rely on the reuse of software architectures from the legacy systems. Hence, these techniques are not applicable when the system is developed from the scratch using the top-down development approach. Moreover, these techniques are not applicable at the design level which is the main focus of this paper. Another work (denoted by MASE 2015 [42] in Fig. 1) uses model transformations to anticipate design-level decisions to support the end-to-end timing analysis. It results in one-to-many implementation-level models corresponding to a single design-level model. However, it does not support the representation and refinement of the end-to-end timing models from the higher to lower abstraction levels. In [43], we have discussed the basic idea for the representation of the timing models at the design level. We have discussed the refinement of two end-to-end delay constraints from the higher to the lower abstraction levels in [1]. In this paper, we generalize our previous work [1] by refining various other types of timing constraints (18 in total) from the higher to the lower abstraction levels. These constraints are concerned with synchronization, repetition, patterns and various types of delays. As a proof of concept, we select the EAST-ADL and TADL2 languages at the higher abstraction levels, whereas RCM is selected at the implementation level.

4 Interpretation of TADL2 timing constraints in RCM

In the first subsection, we present the model of constraints and events. In the following subsections, we discuss various timing constraints in TADL2. We also discuss the semantics of each timing constraint according to the specification of TADL2 [2]. Moreover, we interpret and refine these timing constraints in RCM.

4.1 Model of constraints and events

In TADL2, timing requirements are specified by means of timing constraints on events and event chains [22]. Constraints are used to put restrictions on, e.g., repetition of an event, delays between a pair of events and synchronicity of a set of events. An event denotes a distinct form of state change in a running system. It takes place at distinct points in time which are called its occurrences. There can be any number of

[11] http://vector.com/vi_davinci_developer_en.html.

[12] http://www.palladio-simulator.com/tools/quality_dimensions.

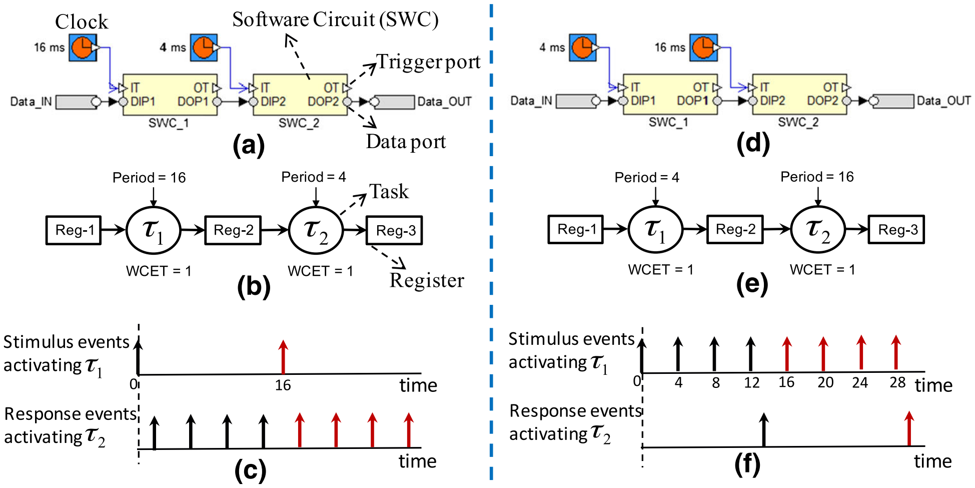


Fig. 4 Event occurrences modeled with colors in the multi-rate chains

occurrences of an event. The set of all the occurrences of an event is called the sequence of the event. A subsequence of the event is a subset of its sequence. For example, if there are ten occurrences of an event within a given time interval, then the size of the event sequence is ten. Any set of two consecutive occurrences within this sequence represents a subsequence of the event within the given time interval. Similarly, any set of three consecutive occurrences within this sequence also represents a subsequence of the event within the given time interval. An event is used to trigger an analysis- or design-level function. When the function is triggered, input data are consumed followed by processing and transformation of the data and then production of the data at the output. A function can also be time triggered.

A timing constraint is denoted by TC. The constraint can be specified on the occurrences of a single event or a set of events. In the former case, the sequence or any subsequence of the single event is constrained. In the later case, the occurrences of the set of events are constrained. In order to clarify the notations that are used to define timing constraints in the following subsections, consider the following example. Consider two events that are denoted by source and target. We use the object-oriented notation to define the attributes of the constraint. For example, TC.source refers to the source event on which TC is specified. Let us denote an occurrence of the event TC.source by an attribute s. The value of this attribute is basically a time point when an instance of the event occurs. These time points can be added, subtracted and compared. A constraint often puts limits on the occurrences of events. These limits can be specified in terms of time distances using upper and lower attributes. In that case, the occurrences of the events are required to happen within these limits. The following provides an example for the semantics of constraint TC.

A system behavior satisfies a specified timing constraint denoted by TC if and only if (iff) for every occurrence of TC.source at time s, there is an occurrence of TC.target at time t such that

$$\texttt{TC.lower} \leq (\texttt{t} - \texttt{s}) \leq \texttt{TC.upper} \tag{1}$$

This means that the timing constraint TC is satisfied if both of the following conditions are satisfied: (1) if the time distance between time points t and s is greater than or equal to the time distance specified by the lower attribute and (2) if the time distance between time points t and s is smaller than or equal to the time distance specified by the upper attribute.

It should be noted that the software components in an event chain can be triggered by independent clocks with different activating periods as shown in Fig. 4a, d. This phenomenon is common in multi-rate systems which are frequently found in the vehicular domain [13,38]. Due to different activating periods along the chain, there can be multiple response occurrences corresponding to a single occurrence of the stimulus in an event chain. For example, the two components in Fig. 4a are activated independently with different periods. Figure 4b shows the task chain that corresponds to the component chain in Fig. 4a at run-time. In this chain, there are four occurrences of the response event corresponding to each occurrence of the stimulus event as shown in Fig. 4c. In such a chain, multiple response occurrences due to each consecutive stimulus occurrence are differentiated by means of colors. For example, assume that the current occurrence of the stimulus is at time 0 in Fig. 4c. All the occurrences of the response event that occur after the current occurrence but before the next occurrence of the stimulus event are represented with the same color (black) as that of the color of the current occurrence of the stimulus. We use a similar approach to associate colors to the event occurrences when there is a single occurrence of the response event correspond-

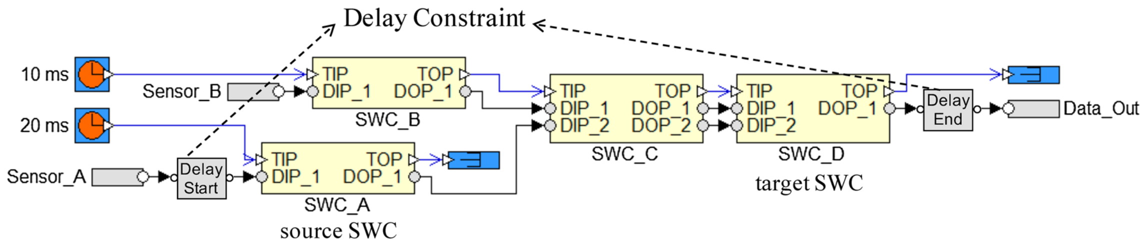


Fig. 5 Proposed objects to specify the `Delay` constraint in RCM

ing to several occurrences of the stimulus events as shown in Fig. 4d–f.

4.2 Delay constraint

4.2.1 TADL2 description

This constraint constrains the distance between occurrences of the `source` and `target` events. It does not matter if the matching `target` occurrence is caused by the corresponding `source` occurrence or not.

4.2.2 Semantics

A system behavior satisfies the specified `DelayConstraint DC` iff for every occurrence `s` of `DC.source`, there is an occurrence `t` of `DC.target` such that

$$\texttt{DC.lower} \leq (\texttt{t} - \texttt{s}) \leq \texttt{DC.upper} \qquad (2)$$

4.2.3 Interpretation in RCM

RCM does not offer any support for the specification of this constraint.

We propose the addition of a new timing constraint with the above semantics, denoted by `Delay`, in RCM. Since this constraint corresponds to the distance between occurrences of the `source` and `target` events, we associate two objects with it, namely `Delay Start` and `Delay End` as shown in Fig. 5. The `Delay Start` object can be specified at the data input port (DIP) of the source SWC. The triggering of trigger input port (TIP) of the source SWC corresponds to a new occurrence of the `source` event. The triggering can be done by a clock or an event in RCM. The `Delay End` object can be specified at the data output port (DOP) of the target SWC. A trigger produced at the trigger output port (TOP) of the target SWC corresponds to a new occurrence of the `target` event. In order to express the `lower` and `upper` values of the constraint, we associate two parameters with the same names to the `Delay End` object.

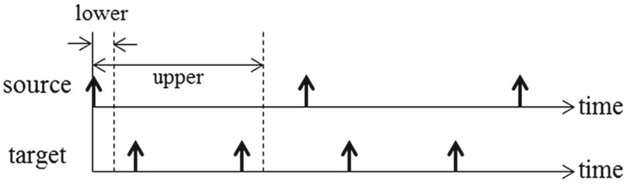


Fig. 6 Event sequence satisfying a `Delay` constraint

The occurrences of the `target` event (data in DOP_1 of SWC_D) may correspond to the input data at DIP_1 of SWC_A or DIP_1 of SWC_B or both depending upon how the SWCs are triggered. In the example shown in Figs. 5 and 6, the occurrences of the `target` event correspond to the input data either from SWC_B or from both SWC_A and SWC_B. The upward arrows in Fig. 6 symbolize occurrences of the events. The `lower` and `upper` attributes for the `Delay` constraint are also identified in Fig. 6. Assuming the priority of the task corresponding to SWC_A to be higher than the priority of SWC_B, the first occurrence of the `target` event matches the first occurrences of both SWC_B and the `source` event, whereas the second occurrence of the `target` event is due to only SWC_B. As discussed earlier, the matching occurrence of the `target` event with respect to the occurrences of the `source` event does not matter in this constraint. This implicitly implies that the activation periods of the source and target events may or may not be equal as shown in Fig. 5.

4.3 Strong delay constraint

4.3.1 TADL2 description

This constraint constrains the distance between each indexed occurrence of the `source` event and the corresponding identically indexed occurrence of the `target` event. Matching of the `target` occurrence caused by the corresponding `source` event occurrence is vital for this constraint.

4.3.2 Semantics

A system behavior satisfies the specified `StrongDelayConstraint SDC` iff the number of occurrences of `SDC.`

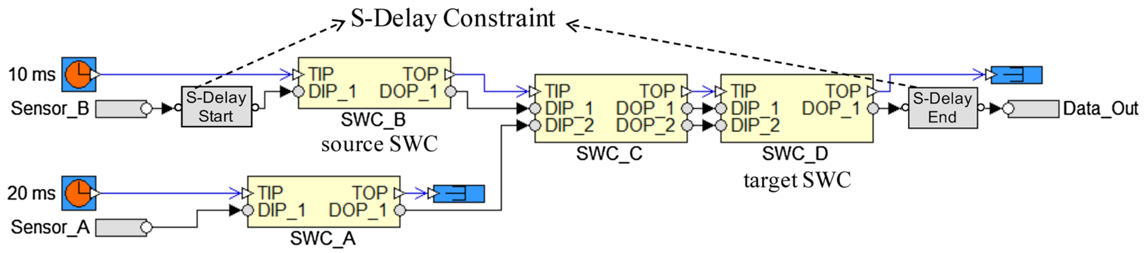


Fig. 7 Proposed objects to specify the `Strong Delay` constraint in RCM

`source` and `SDC.target` events is equal; and for each index i, if there is an ith occurrence of `SDC.source` at time `s` there also is an ith occurrence of `SDC.target` at time `t` such that

$$\texttt{SDC.lower} \leq (\texttt{t - s}) \leq \texttt{SDC.upper} \tag{3}$$

4.3.3 Interpretation in RCM

RCM does not offer any support for the specification of this constraint.

We propose the addition of a new timing constraint with the above semantics, denoted by `S-Delay`, in RCM. Since this constraint corresponds to the distance between two matching occurrences of the `source` and `target` events, we associate two objects with it, namely `S-Delay Start` and `S-Delay End` as shown in Fig. 7. As the number of occurrences of the `source` and `target` events for each index is not equal in the example in Fig. 5, `S-Delay` constraint cannot be used in place of the `Delay` constraint. However, it can be used on the same system if the source SWC is changed as shown in Fig. 7. The `S-Delay Start` object can be specified at the DIP of the source SWC. The triggering of the TIP of the source SWC corresponds to a new occurrence of the `source` event. The `S-Delay End` object can be specified at the DOP of the target SWC. The production of a trigger at the TOP of the target SWC corresponds to the new occurrence of the `target` event. In order to express the `lower` and `upper` values of the constraint, we associate two parameters with the same names to the `S-Delay End` object. These values are identified in Fig. 8. The figure also shows that the occurrences of the `target` event match with the occurrences of the `source` event. This implicitly implies that the activation periods of the source and target events must be equal as shown in Fig. 7.

4.4 Order constraint

This constraint is a special case of the `Strong Delay` constraint (see Sect. 4.3). It constrains an order between the occurrences of any two events. The order constraint is equivalent to the `Strong Delay` constraint after the following three variations:

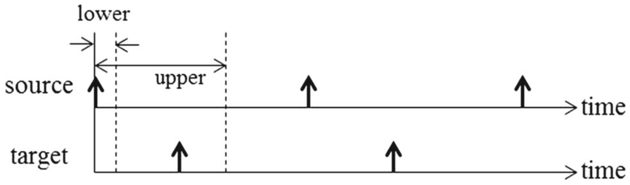


Fig. 8 Event sequence satisfying a `Strong Delay` constraint

1. `SDC.lower` in Eq. 3 is set to zero,
2. `SDC.upper` in Eq. 3 is set to infinity,
3. the matching occurrences of the `source` and `target` events in Eq. 3 denoted by `s` and `t`, respectively, cannot coincide.

4.5 Reaction constraint

4.5.1 TADL2 description

This constraint constrains the occurrence of a `response` event after the occurrence of a corresponding `stimulus` event in an event chain. Basically, the constraint specifies “how long after the occurrence of a stimulus the corresponding response must occur” [2]. This constraint differs from the `Delay` constraint in a way that it can only be applied to event chains and not to individual events. In the multi-rate event chains, multiple response occurrences due to each consecutive stimulus occurrence are differentiated by means of colors. In order to satisfy this constraint, the earliest occurrence of the `response` with the same color as that of the `stimulus` must take place within the limits specified by this constraint as shown in Fig. 9.

4.5.2 Semantics

A system behavior satisfies the `ReactionConstraint ReaC` if and only if for each occurrence of `ReaC.stimulus` at time `s`, there is an occurrence of `ReaC.response` at time `r` such that

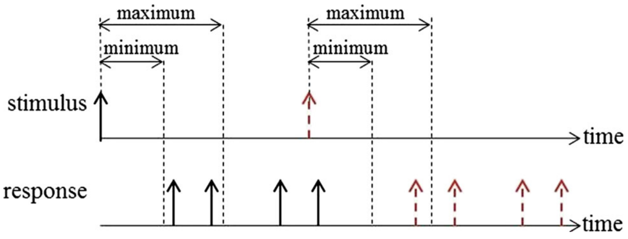


Fig. 9 Event sequence satisfying a `Reaction` constraint

$$
\begin{gathered}
(\texttt{r.color} = \texttt{s.color}) \\
\text{and} \\
(\texttt{r} \text{ is time of the earliest occurrence of } \texttt{ReaC.response} \\
\text{with color } \texttt{s.color}) \\
\text{and} \\
(\text{ReaC.minimum} \leq (\texttt{r} - \texttt{s}) \leq \text{ReaC.maximum})
\end{gathered}
$$

4.5.3 Interpretation in RCM

RCM offers the support to specify the reaction constraint. This constraint is denoted by `DataReaction` (`DR` for short). This constraint can be specified on an event chain, an event chain segment or a distributed event chain (distributed over more than one node) by means of the `DR Start` and `DR End` objects as shown in Fig. 10. The `DR End` object supports the specification of a maximum value by means of a deadline parameter associated with it. However, the minimum parameter is considered to be zero. In order to be consistent with the TADL2 `Reaction` constraint, we associate a new parameter with the `DR End` object to specify the nonzero minimum value of the constraint.

The analysis engines [13] provided by Rubus-ICE support the calculations for the corresponding `Reaction` delay. Consider the example of an event chain in a multi-rate system in Fig. 10. In Fig. 11, we show the time line when this chain is executed (assuming each SWC corresponds to a task denoted by τ at run-time). It should be noted that task τ_B is deliberately given an *offset* of 15 ms to maximize the delays. An offset is an externally imposed time interval between the arrival of the activating event and release of the task for execution. Often, an offset is used to specify temporal dependency among the releases of a set of tasks. The reaction delay is equal to the time elapsed between the previous non-overwritten release of task τ_A (input of the chain) and the first response of task τ_C (output of the chain) corresponding to the current non-overwritten release of task τ_A. Assume that a new value of the input is available in the input buffer of task τ_A "just after" the release of the second instance of task τ_A (at time 8 ms). Hence, the second instance of task τ_A "just misses" the read of the new value from its input buffer. This new value has to wait for the next instance of task τ_A to travel toward the output of the chain. Therefore, the new value is read by the third and fourth instances of task τ_A. The first output corresponding to the new value (arriving just after 8 ms) appears at the output of the chain at 34 ms. This results in the delay of 26 ms as shown in Fig. 11. This phenomenon is more obvious in the case of distributed embedded systems where a task in the receiving node may just miss to read fresh signals from the message arriving from the network. The analysis engines calculate the `Reaction` delay as shown in Fig. 11 and compare it with the specified constraint parameters.

4.6 Age constraint

4.6.1 TADL2 description

This constraint constrains the occurrence of a `stimulus` from the occurrence of the corresponding `response` looking back through the event chain. Basically, the constraint specifies "how long before each response the corresponding stimulus must have occurred" [2]. In order to satisfy this constraint, the latest occurrence of the `stimulus` with the same color as that of the `response` must lie within the limits specified by this constraint as shown in Fig. 12. This constraint differs from the `Delay` constraint in a way that it can only be applied to event chains and not to individual events.

4.6.2 Semantics

A system behavior satisfies the specified `AgeConstraint AgeC` if and only if for each occurrence of `AgeC.response` at time `r`, there is an occurrence of `AgeC.stimulus` at time `s` such that

$$
\begin{gathered}
(\texttt{s.color} = \texttt{r.color}) \\
\text{and} \\
(\texttt{s} \text{ is time of the latest occurrence of } \texttt{AgeC.stimulus} \\
\text{with color } \texttt{r.color}) \\
\text{and} \\
(\text{AgeC.minimum} \leq (\texttt{r} - \texttt{s}) \leq \text{AgeC.maximum})
\end{gathered}
$$

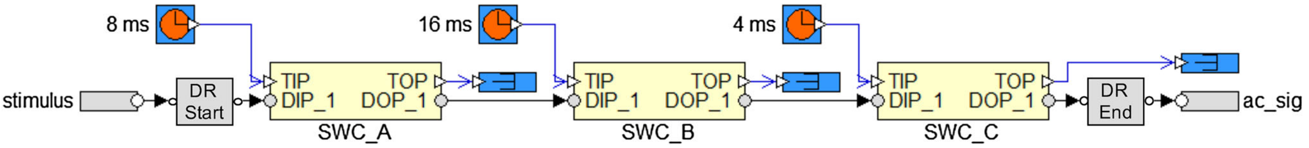


Fig. 10 Existing objects in RCM that are used to specify the `Reaction` constraint

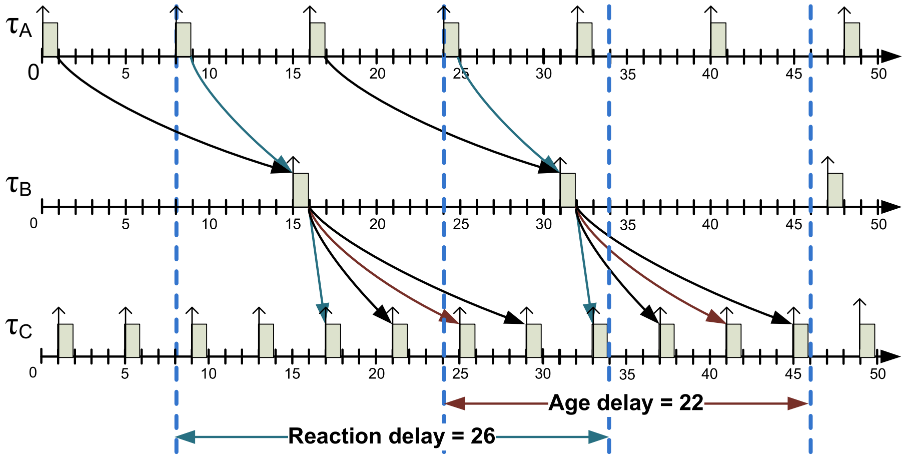


Fig. 11 Demonstration of the Reaction and Age delay calculations by analysis engines. Note that the time is expressed in ms

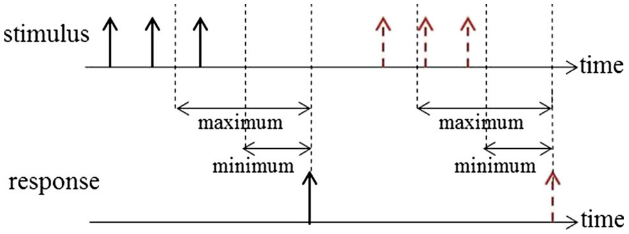


Fig. 12 Event sequence satisfying an Age constraint

4.6.3 Interpretation in RCM

RCM supports the specification of the Age constraint denoted by DataAge. This constraint can be specified on an event chain, an event chain segment or a distributed event chain by means of the Age Start and Age End objects as shown in Fig. 13. The Age End object supports the specification of a maximum value by means of a deadline parameter associated with it. However, the minimum parameter is considered to be zero. In order to be consistent with the TADL2 Age constraint, we associate a new parameter with the Age End object to specify the nonzero minimum value of the constraint.

The analysis engines support the calculations for the corresponding Age delay. Consider the example of an event chain in a multi-rate system shown in Fig. 13. Figure 11 shows the time line when this chain is executed. The analysis engines calculate the Age delay as shown in Fig. 11 and compare it with the specified constraint parameters.

4.7 Repetition constraint

4.7.1 TADL2 description

This constraint constrains the distribution of occurrences of a single event that may also experience *jitter* before its activation. Jitter represents the maximum variation in time with which the event can be delayed. The span attribute associated with this constraint determines which repeated occurrence will be constrained.

4.7.2 Semantics

A system behavior satisfies the specified RepetitionConstraint RC iff the following two are simultaneously satisfied for each subsequence X of RC.event:

1. if X contains span + 1 occurrences, then d is the distance between the outer- and inner-most occurrences in X and

$$\mathtt{RC.lower} \leq \mathtt{d} \leq \mathtt{RC.upper}$$

2. for each index *i*, if there is an *i*th occurrence of X at time s, there also is an ith occurrence of RC.event at time t such that

$$0 \leq (\mathtt{t} - \mathtt{s}) \leq \mathtt{RC.jitter}$$

If the span attribute is equal to one, jitter is equal to zero and the upper attribute is equal to the lower attribute, then the behavior becomes strictly periodic. Figure 14 graphically illustrates this constraint.

4.7.3 Interpretation in RCM

In RCM, an SWC can be time triggered or event triggered by means of the TrigClockTT or TrigClockET objects, respectively. The TrigClockTT object generates periodic trigger signals with a period specified on it, whereas the TrigClockET object generates sporadic trigger signals with a minimum inter-arrival time between any two consecutive occurrences. These two objects are shown in Fig. 15. Another object in RCM, denoted by TrigJitterPeriod,

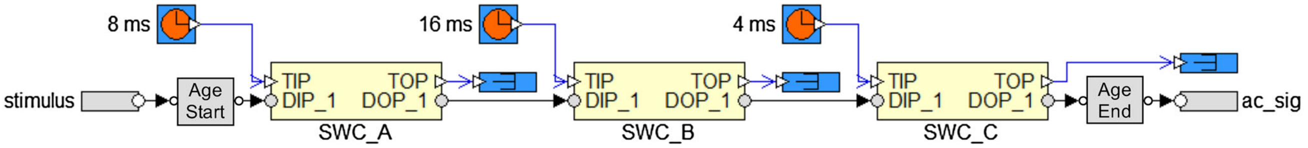


Fig. 13 Existing objects in RCM that are used to specify the `Age` constraint

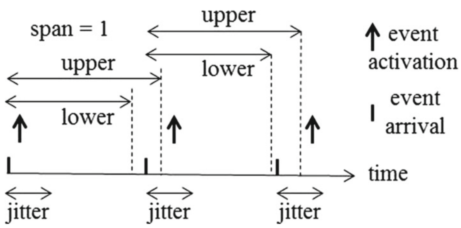


Fig. 14 Event sequence satisfying a `Repetition` constraint

provides the allowance for jitter to the trigger generating objects. Figure 15 contains two of these objects with jitter values equal to 1 millisecond and 100 microseconds.

Note that we associate a new parameter, denoted by the maximum inter-arrival time, with the `TrigClockET` object. This attribute specifies the maximum amount of time that can elapse between the occurrence of any two consecutive arrivals of the sporadic activation events. With this extension, any two consecutive triggers produced by the `TrigClockET` object cannot happen in less than the minimum inter-arrival time and more than the maximum inter-arrival time.

The `TrigClockTT` or `TrigClockET` objects can be combined with the `TrigJitterPeriod` object to represent the TADL2 `Repetition` constraint. In order to be consistent with the TADL2 Repetition constraint, we add the `span` parameter to the `TrigClockTT` and `TrigClockET` objects. When the `TrigClockTT` object is combined with the `TrigJitterPeriod` object, it represents the TADL2 `Repetition` constraint that has the `upper` attribute equal to the `lower` attribute. When the `TrigClockET` object is combined with the `TrigJitter Period` object, it represents the TADL2 `Repetition` constraint with its `lower` and `upper` values assigned to the minimum and maximum inter-arrival time attributes, respectively.

4.8 Repeat constraint

This constraint is a special case of the `Repetition` constraint (see Sect. 4.7). It constrains the distribution of the occurrences of a single event that does not experience any jitter. It is similar to the `Repetition` constraint without allowance for any jitter. Hence, the semantics and refinement for the `Repeat` constraint are the same as that of the `Repetition` constraint with jitter set to zero.

4.9 Sporadic constraint

4.9.1 TADL2 description

This constraint constrains the occurrence of a sporadic event.

4.9.2 Semantics

This constraint is a special type of the `Repetition` constraint whose `span` is equal to 1. Moreover, any two subsequent activations of the event in this constraint must be separated by the minimum inter-arrival time (MIT). This constraint is graphically illustrated in Fig. 16.

4.9.3 Interpretation in RCM

The `TrigClockET` object can be combined with the `TrigJitterPeriod` object to represent the TADL2 `Sporadic` constraint as shown in Fig. 17. In order to consistently interpret this constraint, we set the `span` parameter to 1 and the MIT value equal to the period associated with the `TrigClockET` object. The `lower` and `upper` values can be assigned to the minimum and maximum inter-arrival times. If the maximum inter-arrival time is not specified, it can be considered equal to infinity.

4.10 Burst constraint

4.10.1 TADL2 description

The `BurstConstraint` constrains an event with bursty occurrences.

4.10.2 Semantics

This constraint is a special type of the `Sporadic` constraint with the following extensions.

1. There is no allowance for jitter.
2. There is a maximum number of occurrences of the event, denoted by `MaxOccurrences`, in an interval. The size of the interval is denoted by `length`.

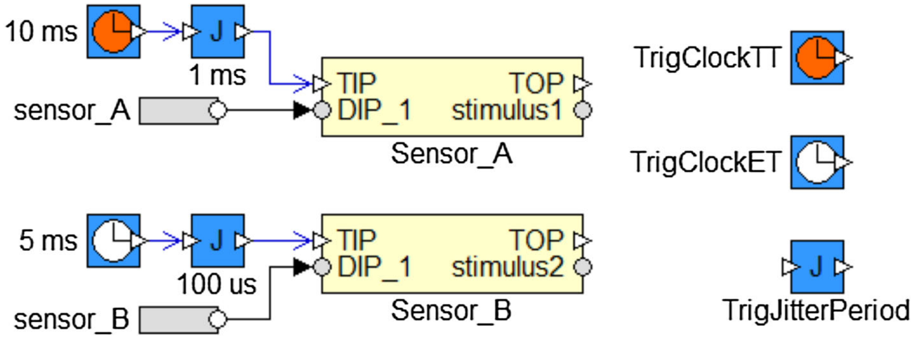


Fig. 15 Existing objects in RCM that are used to specify triggers and jitter

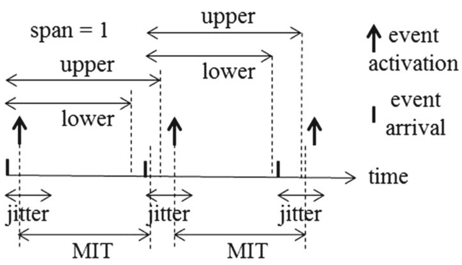


Fig. 16 Event sequence satisfying a `Sporadic` constraint

3. Two subsequent activations in the interval must be separated by the minimum inter-arrival time (MIT).

Two event sequences satisfying the same `BurstConstraint` are shown in Fig. 18.

4.10.3 Interpretation in RCM

The `Sporadic` constraint in RCM is extended to represent the TADL2 `Burst` constraint by setting the `TrigJitterPeriod` to zero and associating the `length` and `MaxOccurrences` attributes to the `TrigClockET` object shown in Fig. 17.

4.11 Periodic constraint

4.11.1 TADL2 description

This constraint constrains the occurrence of a periodic event.

4.11.2 Semantics

This constraint is a special type of `Sporadic` constraint whose `lower` and `upper` attributes are equal. These attributes are assigned the value of the period. This constraint is graphically illustrated in Fig. 19.

4.11.3 Interpretation in RCM

The `TrigClockTT` object can be combined with the `TrigJitterPeriod` object to represent the TADL2 `PeriodicConstraint` as shown in Fig. 20. In order to consistently interpret this constraint, we set the `span` parameter to 1. The `upper` and `lower` parameters are equal and are assigned the value of the period. The MIT value is assigned to the period associated with the `TrigClockTT` object unless specified otherwise.

4.12 Pattern constraint

4.12.1 TADL2 description

This constraint constrains the occurrences of an event that follows a certain pattern with respect to some periodic temporal points.

4.12.2 Semantics

A system behavior satisfies the specified `PatternConstraint PC` iff there is a set of times `X` such that the same system behavior simultaneously satisfies the following conditions:

1. `PeriodicConstraint` with its period equal to `PC.period`. This constraint corresponds to the periodic repetition of the pattern shown in Fig. 21.
2. For each `PC.offset` index i, there is an occurrence x_i of `X` such that

$$\mathtt{PC.offset}_i \leq x_i \leq (\mathtt{PC.offset}_i + \mathtt{PC.jitter})$$

3. If `X` contains two occurrences, then `d` is the distance between the outer- and inner-most occurrences in `X` and

$$\mathtt{PC.minimum} \leq \mathtt{d}$$

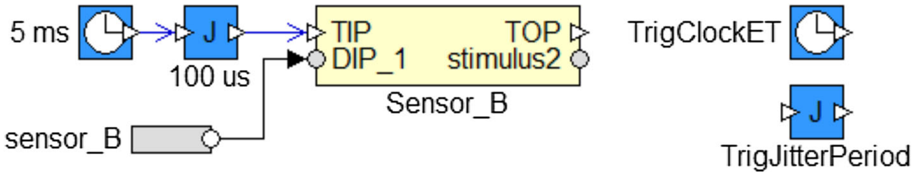


Fig. 17 Equivalent of the `Sporadic` constraint specified in RCM

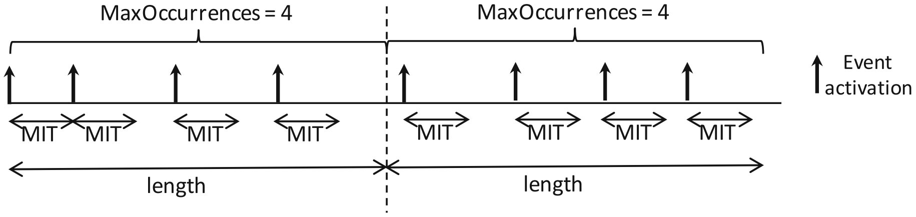


Fig. 18 Event sequences satisfying the `BurstConstraint`

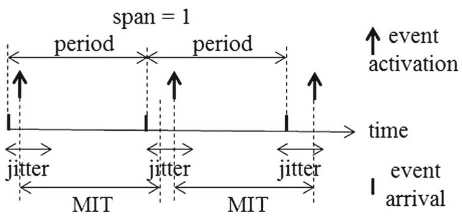


Fig. 19 Event sequence satisfying a `Periodic` constraint

Note that $\mathtt{x}_i$ represents all the occurrences of the event within each `period` shown in Fig. 21.

The `Pattern` constraint is graphically illustrated in Fig. 21. In each period of event patterns, the event occurrences happen at the predefined temporal points, called offsets, with respect to the starting reference point in that period. Each occurrence of the event can be influenced by the specified jitter.

4.12.3 Interpretation in RCM

This constraint is similar to the transactional model of tasks with offsets which is inherent to the time-triggered execution in RCM. At run-time, all time-triggered tasks (assuming an SWC corresponds to a task at run-time) from a node are combined into one big periodic transaction. The tasks within the transaction have offsets and jitter. The period of the transaction is the least common multiple of the periods of all tasks in the transaction.

We propose the addition of a new timing constraint with the above semantics, denoted by the `Pattern` constraint, in RCM as shown in Fig. 22. The parameters associated with this object are period, minimum inter-arrival time, jitter, number of event occurrences during the period time and a set of offsets. The analysis engines are responsible for checking this constraint by comparing the specified parameters with the corresponding parameters in the transactional model.

4.13 Arbitrary constraint

4.13.1 TADL2 description

This constraint constrains an event that occurs irregularly. The constraint contains a set of pairs consisting of a minimum inter-arrival time (denoted by `min`) and a maximum inter-arrival time (denoted by `max`).

4.13.2 Semantics

A system behavior satisfies the specified `ArbitraryConstraint AC` iff for each `AC.min` index i, the same system behavior satisfies, for each subsequence `X` of `AC.event`, if `X` contains i + 1 occurrences then `d` is the distance between the outer- and inner-most occurrences in `X` and

$$\mathtt{AC.min}_i \leq \mathtt{d} \leq \mathtt{AC.max}_i \tag{4}$$

The constraint is graphically illustrated in Fig. 23. In this figure, `min1`, `min2` and `min3` represent the minimum inter-arrival time between/among two, three and four subsequent occurrences of the event, respectively. Similarly, `max1`, `max2` and `max3` represent the maximum inter-arrival time between/among two, three and four subsequent occurrences of the event, respectively. Although three pairs of min and max parameters are plotted for the first two occurrences of the event, these parameters continue in a similar fashion for the rest of the occurrences of the event.

4.13.3 Interpretation in RCM

There is no existing support to specify the arbitrary constraint in RCM. We propose the addition of a new timing con-

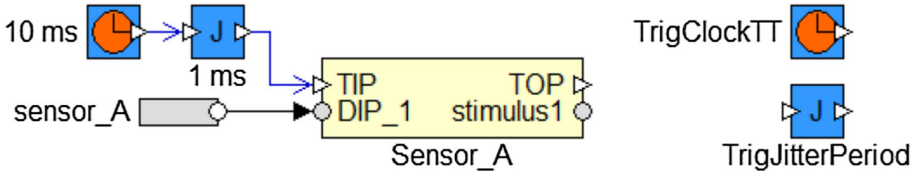


Fig. 20 Equivalent of the `Periodic` constraint specified in RCM

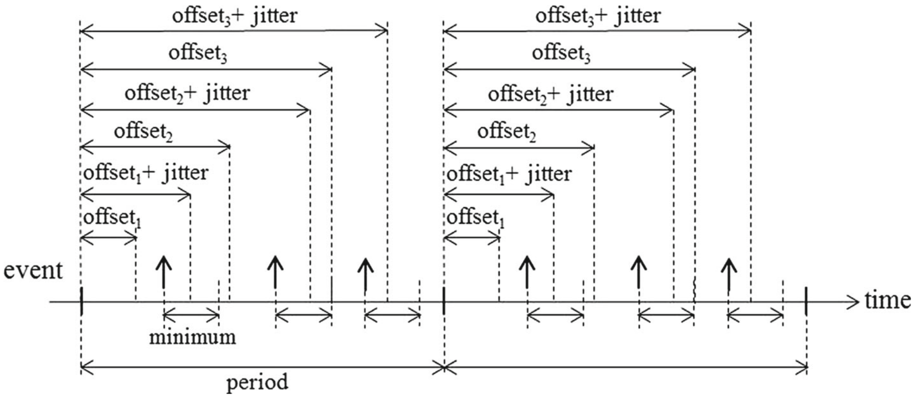


Fig. 21 Event sequence satisfying a `Pattern` constraint

Constraint -> Pattern
PatternEventConstraint

Fig. 22 Proposed object in RCM to specify the `Pattern` constraint

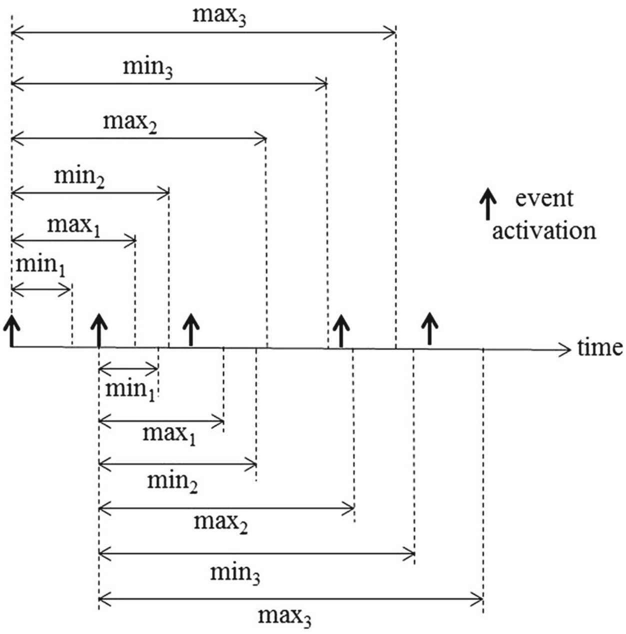


Fig. 23 Event sequence satisfying an `Arbitrary` constraint

straint with the above semantics, denoted by `Arbitrary` constraint, in RCM as shown in Fig. 24. The constraint is able to specify any number of pairs of `min` and `max` values.

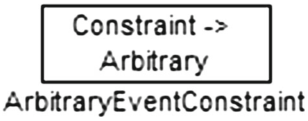


Fig. 24 Proposed object in RCM to specify the `Arbitrary` constraint

4.14 Execution time constraint

4.14.1 TADL2 description

This constraint constrains the time between activation and completion of the execution of a function (executable entity). However, the intervals, when the execution of the function is interrupted due to preemptions and blocking, are not considered in this constraint.

4.14.2 Semantics

A system behavior satisfies the specified `ExecutionTime Constraint ETC` iff for each occurrence `x` of the event `ETC.activate`, ET$_i$ is the set of times between `x` and the next `ETC.completion` while excluding the times due to `ETC.preemtion` and `ETC.blocking`, and that

$$\texttt{ETC.lower} \leq \text{sum of all continuous intervals in } \texttt{ET}_i \leq \texttt{ETC.upper} \quad (5)$$

This constraint is graphically illustrated in Fig. 25.

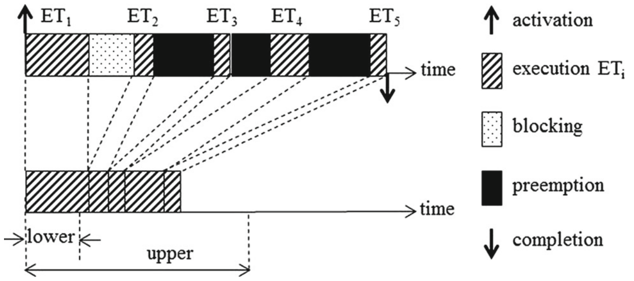


Fig. 25 Event sequence satisfying an execution time constraint

4.14.3 Interpretation in RCM

RCM supports the specification of the execution time constraint for an SWC. Each SWC has one or more behaviors, whereas each behavior represents a function. When an SWC is triggered, its state and data (from all of its DIPs) are passed to it. The states are updated and the newly calculated data are placed on the DOPs while a trigger is produced at the TOP upon completion of the behavior. RCM supports the specification of three types of execution times on the behavior of SWC, namely best-case execution time (BCET), worst-case execution time (WCET) and average-case execution time (ACET) as shown in Fig. 26. In order to unambiguously interpret this constraint in RCM, the `lower` and `upper` values of this constraint (see Fig. 25) can be assigned to the BCET and WCET parameters, respectively, in Fig. 26.

4.15 Synchronization constraint

4.15.1 TADL2 description

This constraint constrains the closeness of the occurrences of a group of events.

4.15.2 Semantics

A system behavior satisfies the specified `SynchronizationConstraint` on a given set of events and given the occurrence of any event in this set, then the rest of the events in the set must occur at least once within a certain time window called `tolerance`.

This constraint is graphically illustrated in Fig. 27. It is applied on the two events data_A and data_B. In this constraint, more than one instance of the events may exist in a time window, provided the above conditions are met. Moreover, the windows may overlap and they may share occurrences of the events.

4.15.3 Interpretation in RCM

There is an existing support in RCM to synchronize multiple triggers by means of a synchronization object denoted by `TrigSync` as shown in Fig. 28. This object has two or more TIPs and only one TOP. The synchronization condition can use either AND or OR semantics. In the case of the AND condition, the TOP is triggered only when trigger signals have arrived at all TIPs. In the case of the OR condition, the TOP is triggered as soon as there is a trigger signal at one of the TIPs. In order to make this constraint consistent with the TADL2 `Synchronization` constraint, we add the `tolerance` parameter to this object. The analysis engines are responsible for checking the constraint by determining if the triggering events occur within the `tolerance` window or not.

4.16 Strong synchronization constraint

4.16.1 TADL2 description

This constraint constrains the closeness of the occurrences of a group of events.

4.16.2 Semantics

The semantics of the `StrongSynchronizationConstraint` differs from the semantics of the `SynchronizationConstraint` in a way that the occurrences of the events in a window must have same indices. Therefore, at most one instance of the events can exist in the time window. Moreover, the windows cannot overlap and they may share occurrences of the events.

This constraint is graphically illustrated in Fig. 29. It is applied on the two events data_A and data_B.

Fig. 26 Equivalent to the execution time constraint specified in RCM

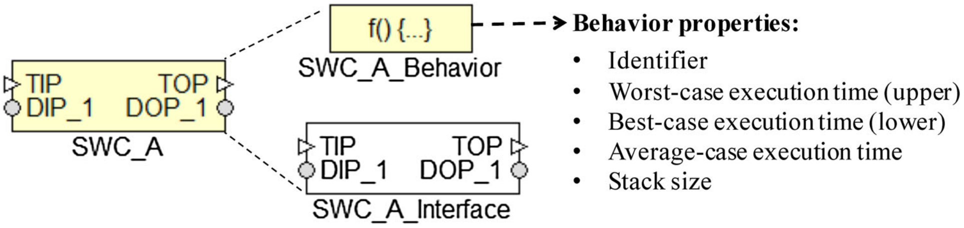
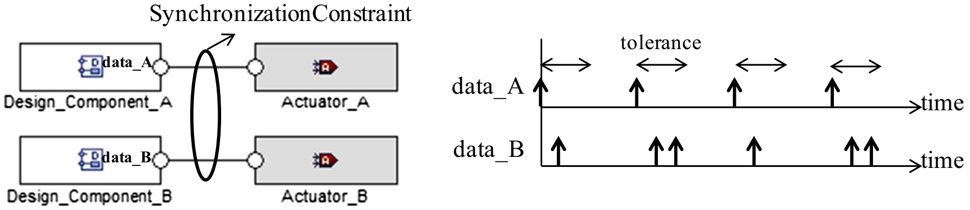


Fig. 27 Event sequences satisfying a `Synchronization` constraint



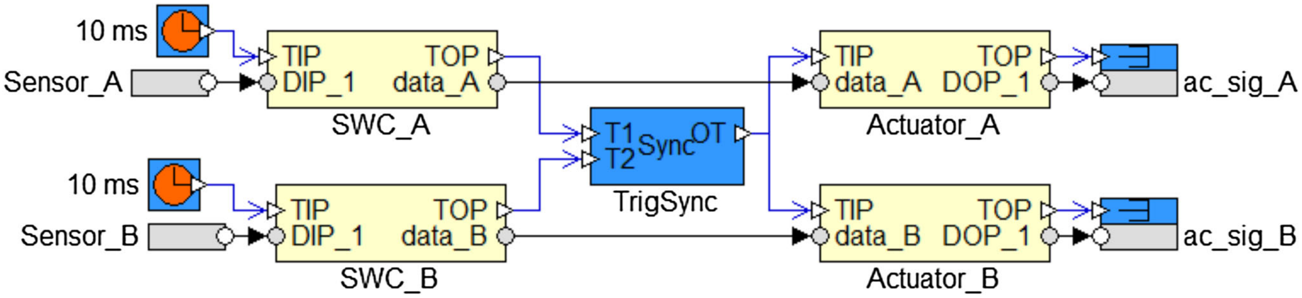
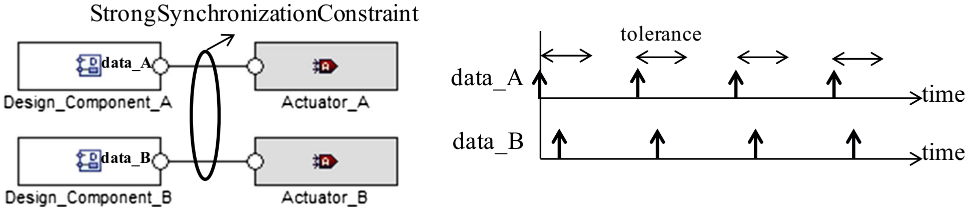


Fig. 28 `Synchronization` constraint in RCM

Fig. 29 Event sequences satisfying a `Strong Synchronization` constraint



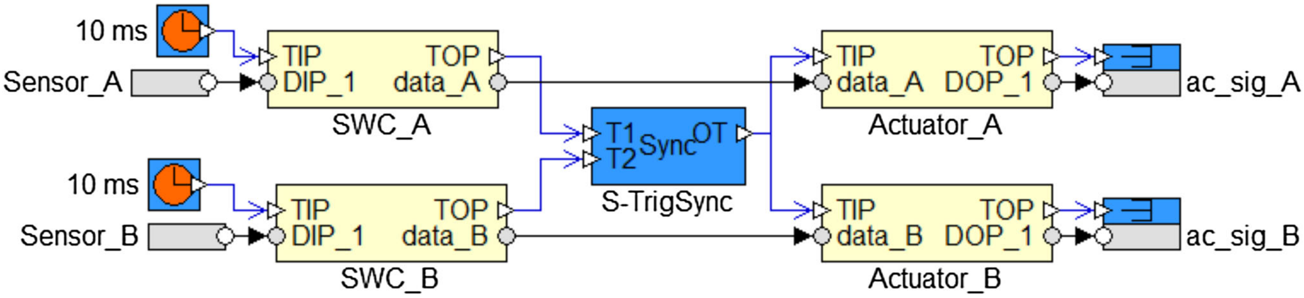


Fig. 30 Proposed object in RCM to specify the `Strong Synchronization` constraint

4.16.3 Interpretation in RCM

There is an existing support in RCM to synchronize multiple triggers by means of a synchronization object denoted by `TrigSync`. In order to differentiate the `Strong Synchronization` constraint from this object, we add a similar object denoted by `S-TrigSync` as shown in Fig. 30. This object has two or more TIPs and only one TOP. The synchronization condition can use either AND or OR semantics. In order to make this constraint consistent with the TADL2 `Strong Synchronization` constraint, we add the tolerance parameter to this object.

4.17 Output synchronization constraint

4.17.1 TADL2 description

This constraint constrains the closeness of the occurrences of responses to a certain stimulus. Basically, the constraint defines how far apart the responses to a certain stimulus can occur. This constraint differs from the `SynchronizationConstraint` in a way that it can only be applied to a set of event chains such that there are multiple responses to a single stimulus as shown in Figs. 31 and 32. The `tolerance` parameter constrains the latest of these response occurrences for each chain. The system in Fig. 31 is modeled with two event chains. They have common stimulus but different responses denoted by response1 and response2.

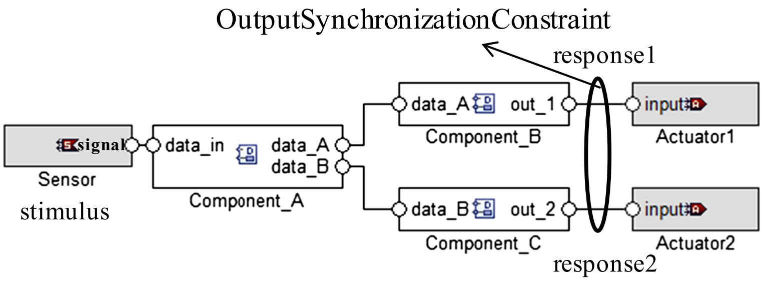


Fig. 31 Usage of the `Output Synchronization` constraint at the design level

4.17.2 Semantics

A system behavior satisfies the `OutputSynchronizationConstraint OSC` iff for each occurrence of `OSC.stimulus` at time `s`, there is a time `t` such that for each index i, there is an occurrence of $\texttt{OSC.response}_i$ at time `r` such that

$$(\texttt{r.color = s.color})$$
$$\text{and}$$
$$(\texttt{r} \text{ is time of the earliest occurrence of } \texttt{OSC.response}_i \text{ with color } \texttt{s.color})$$
$$\text{and}$$
$$(0 \le (\texttt{r} - \texttt{t}) \le \texttt{OSC.tolerance})$$

4.17.3 Interpretation in RCM

There is an existing support in RCM to synchronize multiple triggers by using the `TrigSync` object. We add a similar object, denoted by `Out-TrigSync`, in RCM. This object has two or more TIPs and only one TOP. The synchronization condition can use either AND or OR semantics. In order to make this constraint consistent with the TADL2 `Output Synchronization` constraint, we add the `tolerance` parameter to it. The analysis engines must ensure that this constraint is satisfied within the `tolerance` window. The example in Fig. 33 depicts a single-rate system. Hence, there cannot be more than one occurrences of each response corresponding to single occurrence of the stimulus. However, the `Out-TrigSync` is equally applicable to multi-rate systems where the components are triggered with independent clocks. It is important to note that the `Output Synchronization` constraint can also be specified in distributed systems. For example, the common stimulus of any two chains can be on one node, while their responses can be on two different nodes (other than the stimulus node). In such a case, two `TrigSync` objects are specified on the two response nodes. However, the usage name of these objects is the same. The run-time environment must consider any two or more `TrigSync` objects with the same usage name as one object.

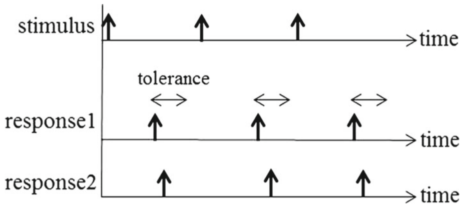


Fig. 32 Event sequences satisfying the `Output Synchronization` constraint

4.18 Input synchronization constraint

4.18.1 TADL2 description

This constraint constrains the closeness of the occurrences of stimuli corresponding to a certain response. Basically, the constraint defines how far apart the stimuli corresponding to a certain response can occur. This constraint differs from the `Synchronization` constraint in a way that it can only be applied to a set of event chains such that there are multiple stimuli and a single corresponding response as shown in Figs. 34 and 35. The `tolerance` parameter constrains the latest of these stimuli occurrences for each chain. This means that once one of the stimuli has been acquired, the others should be acquired within the time window equal to the `tolerance` parameter. The system in Fig. 34 is modeled with two event chains. They are initiated by separate stimuli but they have one common response.

4.18.2 Semantics

A system behavior satisfies the `InputSynchronization Constraint ISC` iff for each occurrence of `ISC.response` at time `r`, there is a time `t` such that for each index i, there is an occurrence of $\texttt{ISC.stimulus}_i$ at time `s` such that

$$(\texttt{r.color = s.color})$$

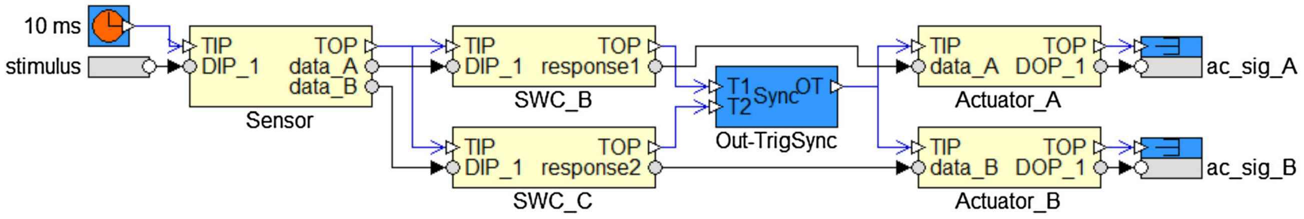
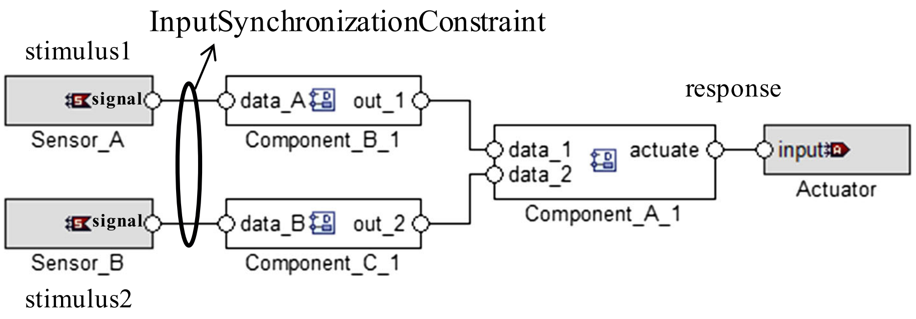


Fig. 33 Proposed object to specify the `Output Synchronization` constraint in RCM

Fig. 34 Usage of the `Input Synchronization` constraint at the design level



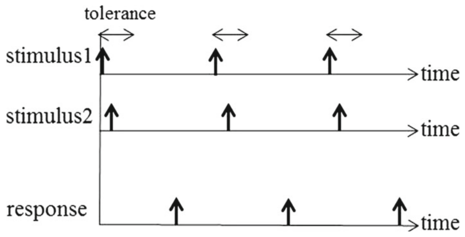


Fig. 35 Event sequences satisfying the `Input Synchronization` constraint

and
(s is time of the earliest occurrence of `ISC.stimulus`$_i$
with color `r.color`)
and
(0 ≤ (`s` − `t`) ≤ `ISC.tolerance`)

4.18.3 Interpretation in RCM

There is an existing support in RCM to synchronize multiple triggers by using the `TrigSync` object. We add a similar object, denoted by `In-TrigSync`, in RCM. This object has two or more TIPs and only one TOP. The synchronization condition can use either AND or OR semantics. In order to make this constraint consistent with the TADL2 `Input Synchronization` constraint, we add the `tolerance` parameter to it. The example in Fig. 36 depicts a single-rate system. Hence, there cannot be more than one occurrences of each response corresponding to single occurrence of the stimulus. However, the `In-TrigSync` is equally applicable to multi-rate systems where the components are triggered with independent clocks.

4.19 Comparison constraint

This constraint is not a timing constraint. In fact, it is used to represent the comparison between the value of a specified constraint and the values of the variables that have arithmetic relations between/among them. For example, consider a distributed chain that consists of three sub-chains. Also assume that the delay of each sub-chain is calculated separately. The distributed chain is considered schedulable if the sum of the three delays is less than or equal to the `Delay` constraint specified on the distributed chain. Since the `Comparison` constraint is not a timing constraint, it does not require any refinement. The Rubus tool suite automatically compares each specified constraint with the corresponding calculated value. The comparison results are presented to the user. Moreover, the results are back-propagated to the models at the higher abstraction levels.

5 Challenges in the representation of the end-to-end timing model at the design level

The models and approaches that are used at the implementation level such as RCM and AUTOSAR allow to represent the end-to-end timing models. However, the modeling approaches used at the design or higher levels such as EAST-ADL, TIMMO and TADL2 do not support complete and unambiguous representation of the timing models. Due to unavailability of the end-to-end timing models at the higher abstraction levels, it is not possible to perform the end-to-end timing analysis [13,38]. As discussed earlier in Sect. 3.7, there are few works that support the end-to-

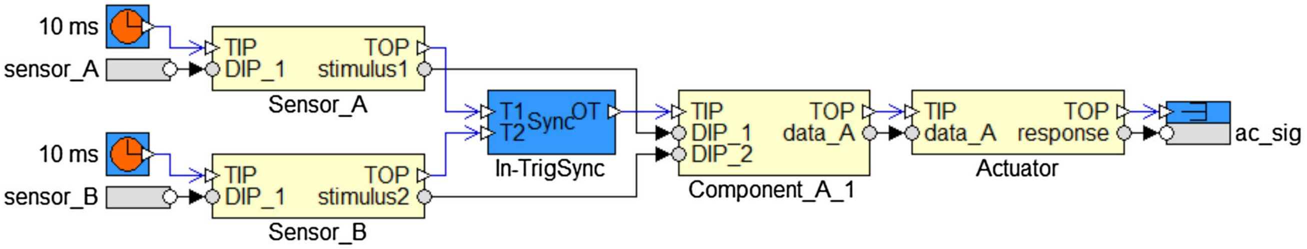


Fig. 36 Proposed object to specify the `Input Synchronization` constraint in RCM

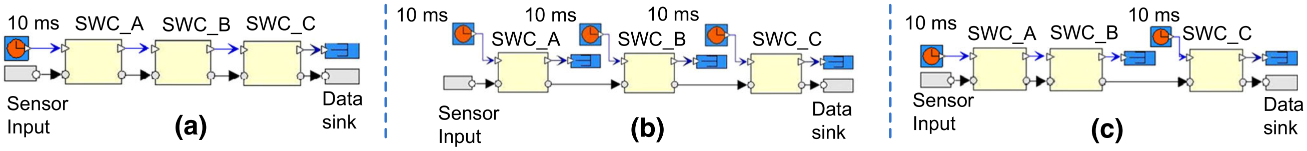


Fig. 37 Example of **a** trigger chain, **b** data chain and **c** mixed chain

end timing analysis at the higher levels of abstraction such as [18,19]. However, the analysis supported by these works is of low precision. It has already been shown in [18] that the analyses in [18,19] can be highly pessimistic (overestimated) as compared to the analyses in [13,38]. The analyses in [18,19] heavily rely on the reuse of software architectures from legacy systems. Hence, these analyses are not applicable when the system is developed from the scratch. On the other hand, our work aims to support the high-precision end-to-end timing analysis [13,38] at the higher abstraction levels. We focus mainly on the design level within the context of this problem. We consider the modeling support of EAST-ADL, TIMMO and TADL2 at the design level, whereas the modeling support of RCM is considered at the implementation level. We discuss some of the challenges that hinder the representation of the end-to-end timing model. We propose guidelines and solutions to deal with these challenges. We also discuss the implementation of these solutions in RCM.

5.1 Representation of control and data paths

Unambiguous representation of control (trigger) and data paths from the system is vital for performing its end-to-end timing analysis. A trigger path captures the flow of triggers along a chain of components (tasks at run-time). For example, the trigger path in the chain shown in Fig. 37c can be expressed as {{SWC_A → SWC_B}, {SWC_C}} because SWC_B is triggered by SWC_A, while SWC_C is triggered independently. Similarly, the trigger paths in the chains shown in Fig. 37a, b can be expressed as {{SWC_A → SWC_B → SWC_C} and {{SWC_A}, {SWC_B}, {SWC_C}}, respectively.

One of the main challenges in the representation of an end-to-end timing model at the design level is the lack of a clear separation between the trigger and data paths. At the implementation level, e.g., in RCM, these paths are clearly separated from each other by means of trigger and data ports as shown in Fig. 38b. A TOP of an SWC can only be connected to the TIP(s) of other SWC(s). Similarly, a DOP of an SWC can only be connected to the DIP(s) of other SWC(s). Hence, the trigger and data paths can be clearly identified.

On the other hand, the components at the design level communicate via the *flow ports* as shown in Fig. 38a. A flow port is an EAST-ADL object that is used to transfer data between components. It is single buffer, non-consumable and over-writable. Without any explicit information, it can be interpreted as a data or trigger port at the implementation level. There is no support to specify explicit trigger paths at the design level. Moreover, a component can be triggered via specified timing constraints on event, modes or internal behavior of the component. The two types of 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 not meaningful to compute the age delay of a trigger chain shown in Fig. 38a [13]. Since the age delay in a trigger chain is always equal to its response time, the calculations for the age delay in this case will produce redundant results.

In order to clearly identify the trigger and data paths at the design level, we make the following assumptions.

1. We assume a one-to-one mapping between each design- and implementation-level component. In general, there can be an n-to-m mapping between a design- and an implementation-level component. Our assumption is quite practical because most of the existing works, such as [44], consider a one-to-one mapping between the design-level components (developed using EAST-ADL) and the implementation-level components (developed using AUTOSAR). In addition, our assumption is based on the

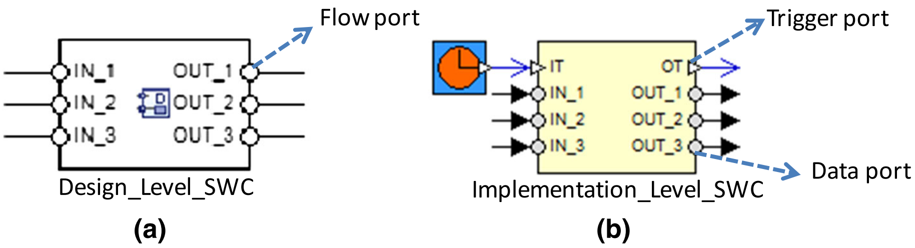


Fig. 38 Model of the SWC at **a** design level and **b** implementation level

common practice that is used in the vehicle industry, especially in the segment of construction equipment vehicles domain.

2. A flow port of a software component can be triggered either by an independent source such as a clock or by a dependent source such as another software component. If the components in a chain are triggered independently, then the resulting end-to-end delays in the chain are higher as compared to the case when the components along the chain are triggered dependently [13,38]. If there is no trigger information available for a flow port of a software component on which a timing constraint is specified, we assume that the component is triggered independently. The type of triggering is judged by the type of the constraint. This assumption is pessimistic but safe because we are interested in the worst-case end-to-end timing analysis.
3. If the `Age` or `Reaction` are the only constraints that are specified on a chain, we assume that the first and last components in the chain are triggered independently. This is because more than one independent trigger in a chain makes it a multi-rate chain. Otherwise, the chain becomes a single-rate chain. In a single-rate chain, the age delay is equal to its response time, while the reaction delay is a slight variation of its response time. Hence, the schedulability of a single-rate chain can be determined by response-time analysis [12] without performing the end-to-end delay analysis [13,38]. Therefore, the single-rate chains are constrained by the deadline constraints instead of the age and reaction constraints. It is more meaningful to specify the Age and Reaction constraints on the multi-rate chains as compared to the single-rate chains.
4. We assume that a flow port is implicitly triggered at the arrival of data. If there are more than one flow ports in a component, then the arrival of data at each port produces a trigger. For example, the component in Fig. 38a may receive three individual triggers when data are separately received at the three input flow ports. The TrigSync object in RCM can be used to deal with multiple implicit triggers (corresponding to multiple flow ports) 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 SWC (corresponding to the design-level component) at the implementation level. Figure 39 shows an implementation-level equivalent of the design-level component with three flow ports as shown in Fig. 38a.

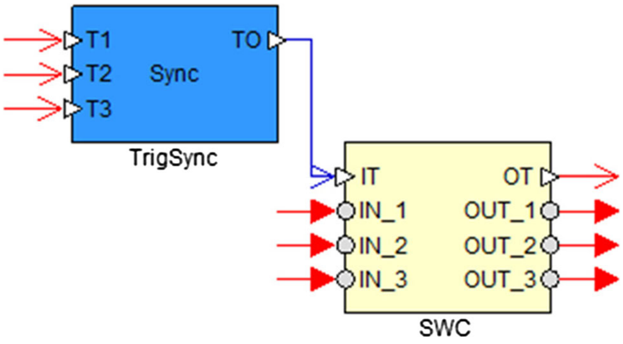


Fig. 39 Implementation-level equivalent of the design-level component in Figure 38(a)

5.2 Representation of timing parameters

The timing information expressed with the models and tools used at the design level is not enough to represent the end-to-end timing model. For example, one of the EAST-ADL based tools[13] used at the design and higher levels is able to specify only one timing parameter on components, 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 constraints and properties at the design level in EAST-ADL and AUTOSAR-based development. However, TADL2 does not allow to express some timing parameters, e.g., priority and transmission type which are needed to perform the end-to-end timing analysis. We have already discussed the interpretation of TADL2 timing constraints in RCM in the previous section.

We assume that the execution order of the design-level components in a chain is specified; otherwise, we make an implicit assumption about it. That is, each component is assumed to execute only after successful execution of its preceding component in the chain unless specified otherwise. This means a data provider component is assumed to be

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

always executed before the data receiver component. Since this assumption fixes the execution order, it is safe to assume that the priorities of the components are equal within the chain. Note that this assumption is in line with the fourth assumption in Sect. 5.1. If worst-, best- and average-case execution times are not available at the design level, they can be estimated at the implementation level either by using estimates by the experts or by reusing them from the other projects or previous releases of the vehicle.

5.3 Identification of chain types

The chain types in RCM can be easily identified because the control and data flows are clearly separated at the implementation level. Various types of chains in RCM are depicted in Fig. 37. Since there is 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 analysis cannot be performed. This is because a trigger chain is analyzed by calculating its end-to-end response time and reaction delay, whereas a data or a mixed chain is analyzed by calculating its end-to-end response time and reaction delay as well as its age delay [13]. If there are no constraints specified on a chain, we assume it to be a trigger chain. Otherwise, it can be considered as a data or a mixed chain depending upon how the constraints are specified.

5.4 Information duplication and ambiguity

At the implementation level, 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 component in EAST-ADL using TADL2. Similarly, if the data age and reaction constraints are wrongly specified, then the development environment does not complain about it. As a result, the timing model may have redundant or erroneous information. Information duplication can lead to inconsistency in the timing model. However, at the implementation level, Rubus-ICE complains about these inconsistencies and ambiguities. The analysis engines calculate the age and reaction delays only when the corresponding constraints are specified on data and mixed chains.

5.5 Implementation challenges and applicability of the approach

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 represented. Moreover, the represented models are general enough to be operated on by different models and tools. The only problem with this approach is that it requires strong 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 interpretation of the design level that depends upon the execution-level modeling technology. Such an interpretation should be general enough to be applicable to any component model which is designed for the software development at the implementation abstraction level, for example, developing a Rubus interpretation of EAST-ADL. 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 representation of the end-to-end timing models. In this paper we have advocated the second option.

The approach proposed in this paper can be generally applied to any implementation-level component model for the development of vehicle software that (1) supports a pipe-and-filter style for the interaction between/among software components, (2) differentiates between the control and data flows between/among the software components and (3) allows representation of the low-level details at the higher abstraction levels such as the linking information in distributed chains [13,14]. Moreover, the challenges and proposed solutions discussed in this paper are equally applicable to other higher-level modeling technologies that comply with the EAST-ADL methodology. Note that all the assumptions made in this paper reflect the worst-case conditions. Hence, the analysis results can be sometimes pessimistic (overestimated) but safe, i.e., the results cannot be optimistic (underestimated). The timing model representation approach is well suited to hard real-time software systems that are required to meet stringent timing requirements.

5.6 Implementation of the refinement in Rubus-ICE

The refinement of the TADL2 timing constraints to RCM (discussed in Sect. 4) is hard coded in the refinement engine of Rubus-ICE as shown in Fig. 40. Note that all EAST-ADL editors support exchange of the design-level model in the XML format. Such a model, augmented with the TADL2 timing constraints, is read by the refinement engine. The output of this engine is the refined implementation-level model. The existing end-to-end timing analysis engines [13,38] in Rubus-ICE are extended based on the assumptions and guidelines that are discussed in this section. The end-to-end timing

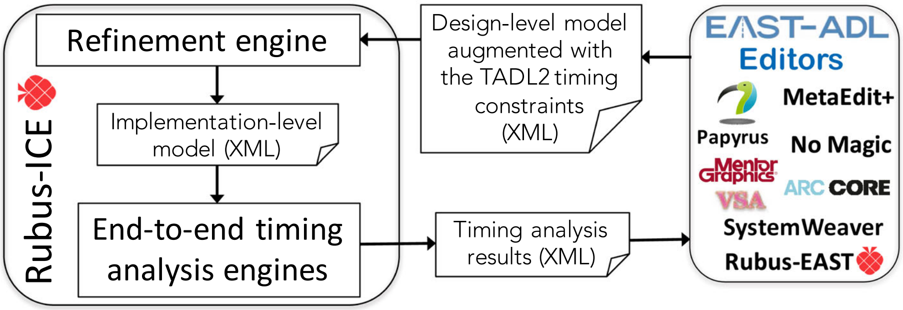

Refinement engine
Implementation-level model (XML)
End-to-end timing analysis engines
Rubus-ICE
Design-level model augmented with the TADL2 timing constraints (XML)
Timing analysis results (XML)
EAST-ADL Editors
MetaEdit+
Papyrus
No Magic
Mentor Graphics
VSA
ARC CORE
SystemWeaver
Rubus-EAST

Fig. 40 Information flow after the implementation of the refinement in Rubus-ICE

analysis results obtained from the analysis engines are back-propagated to the design-level models as shown in Fig. 40.

6 Vehicular-application case study

In this section, first we model the steer-by-wire (SBW) system with EAST-ADL at the design level. In [1], we modeled partial software architectures of only two nodes in the SBW system. This section extends the previous case study by modeling the complete software architecture of the SBW system. In the second step, we specify several timing constraints on the software architecture of the SBW system. In the third step, the design-level software architecture along with the specified timing constraints are refined to the implementation-level software architecture. In the fourth step, the analysis engines are run to verify the specified timing constraints.

6.1 Steer-by-wire (SBW) system

The SBW system provides electronic steer control to a vehicle by substituting majority of mechanical and hydraulic components with electronic components in the conventional steering system. In this system, the steering angle is converted into electrical signals. These signals are then processed to produce actuation signals that control the direction of the wheels. The SBW system consists of five nodes or electronic control units (ECUs) that are connected to a single Controller Area Network (CAN) [45] bus as shown in Fig. 41. The CAN bus is assumed to operate at the speed of 250 Kbit/s. There are four ECUs for Wheel Control (WC) and one ECU for Steer Control (SC). The WC ECUs for front-left, front-right, rear-left and rear-right wheels are denoted by FL_WC, FR_WC, RL_WC and RR_WC in Fig. 41.

The SC ECU receives inputs from three sensors that include steering angle, steering torque (applied by the driver) and vehicle speed sensors. It receives one CAN message from each WC ECU. The message includes information regarding the torque of each wheel. Based on these inputs, the SC ECU calculates the feedback steering torque and sends it to the feedback torque actuator. This actuator is responsible for producing the feeling of turning effect of the steering wheel for the driver. Such an effect corresponds to the grip of the wheels. The wheel actuators in the WC ECUs should move the wheels in accordance with the steering wheel movements. Hence, the SC ECU sends two CAN messages to all WC ECUs. One message carries the steer angle signal. Whereas the other message carries the steer torque signal.

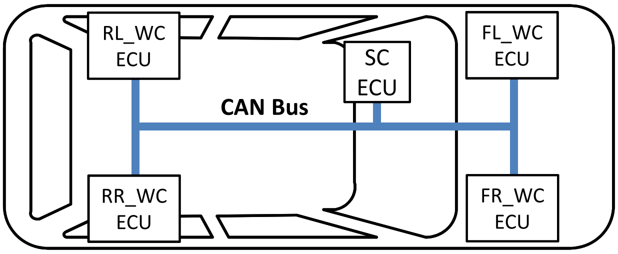


Fig. 41 Block diagram of the SBW system

Each WC ECU receives inputs from wheel angle and wheel torque sensors. Depending upon the sensor inputs and the CAN message that is received from the SC ECU, each WC ECU calculates the wheel torque and produces actuation signals for the corresponding wheel actuator. The actuator is responsible for moving the corresponding wheel in accordance with the steering wheel movements. Each WC ECU sends one CAN message to the SC ECU containing the corresponding wheel torque signals.

6.2 Modeling of the SBW system at the design level

The software architecture of the SBW system at the design level, modeled with EAST-ADL, is depicted in Fig. 42. The left-hand side of the figure shows the software architecture of the SC ECU, whereas the right-hand side of the figure shows the software architectures of the four WC ECUs. Each component in Fig. 42 is a Function Prototype which is the design-level software component in EAST-ADL. It should be noted that EAST-ADL does not provide detailed models of networks. Hence, the components that require inter-ECU communication are interconnected using direct connections, e.g., SC_Controller and FL_Controller. The detailed network

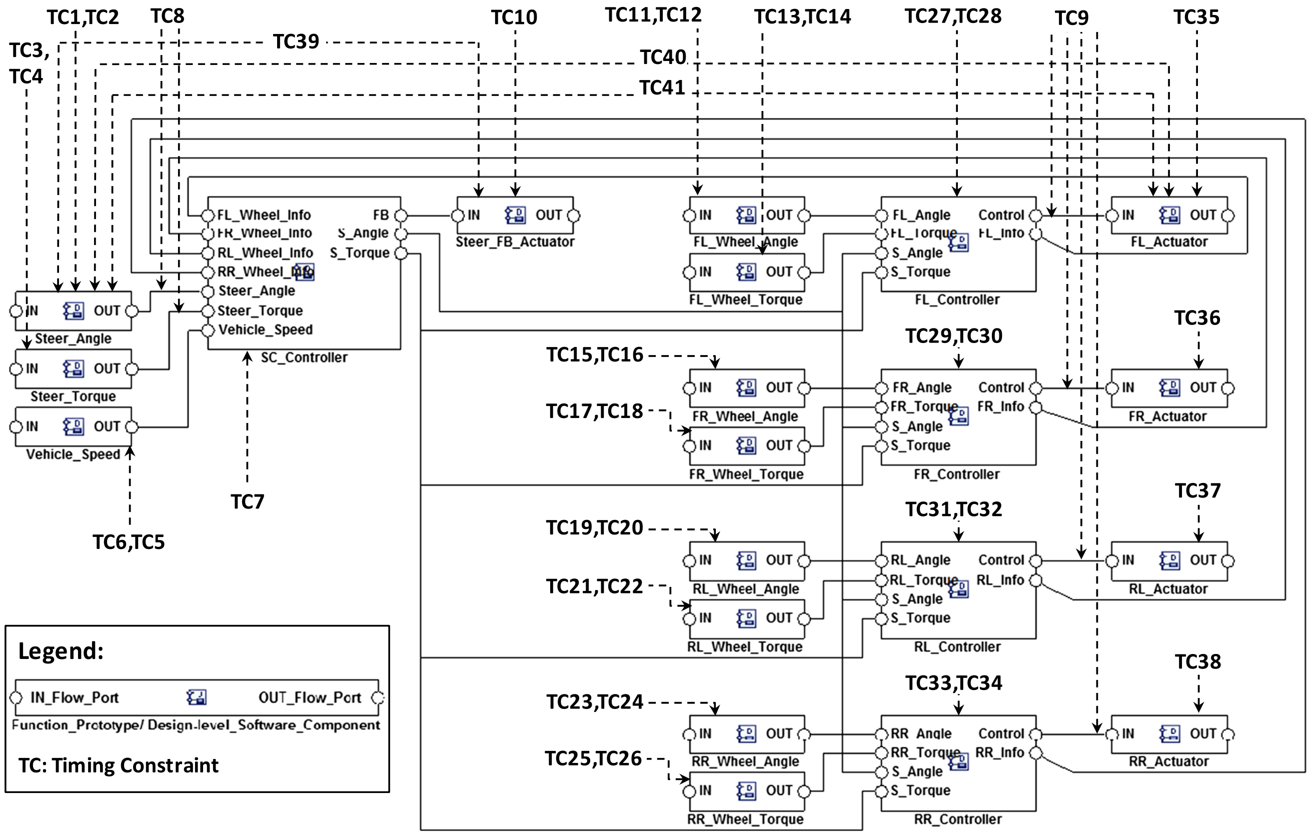


Fig. 42 Design-level software architecture of the SBW system in EAST-ADL

communication is modeled only at the implementation level. Hence, these components communicate with each other via network messages at the implementation level.

6.3 Specification of timing constraints at the design level

There are 41 Timing Constraints (TCs) that are specified on the software architecture of the SBW system shown in Fig. 42. These constraints comprise of nine different types of timing constraints including `Periodic`, `Sporadic`, `Repetition`, `Strong Delay`, `Execution Time`, `Age`, `Reaction`, `Input Synchronization` and `Output Synchronization`. Various attributes that are associated with these constraints are listed in Table. 1. Let us consider three examples to understand the specified timing constraints. `TC1` is a `Periodic` constraint that is specified on the Steer_Angle component. It requires the activation of Steer_Angle to be strictly periodic with a period of 10,000 µs and maximum allowed jitter of 10 µs. `TC9` represents `Output Synchronization` constraint among the outputs of the FL_Controller, FR_Controller, RL_Controller and RR_Controller components. It constrains the closeness of occurrences of the responses of these four components by 60 µs. `TC40` represents the `Age` constraint that constrains the data age delay between the arrival of input data at the Steer_Angle component in the SC ECU and the production of output data by the FL_Actuator component in the FL_WC_ECU. The maximum and minimum values associated with this constraint are equal to 20,000 and 30,000 µs, respectively.

6.4 Refinement of the SBW system to the implementation level

In order to refine the software architecture of the SBW system from the design level to the implementation level, we use the model representation and timing constraints refinement approach that we have discussed in Sects. 4 and 5. The refined system-level software architecture of the SBW system is shown in Fig. 43. This figure contains the models of five ECUs and one CAN bus. There are six messages (see Sect. 6.1 for details) in the network. Each message is assumed to carry a maximum amount of data, i.e., 8 bytes. The refined software architecture of the SC ECU is shown in Fig. 44, whereas the refined software architectures of the four WC ECUs are shown in Fig. 45.

Each `Periodic` constraint is refined as a pair of periodic clock and jitter objects. For example, `TC11` is refined to the periodic clock and jitter objects that are connected to the input trigger port of the FL_Wheel_Angle component in Fig. 45.

Table 1 Attributes of the timing constraints specified in Fig. 42

Constraint	Constraint type	Lower/min. (us)	Upper/max. (us)	Jitter (us)	Span	Tolerance (us)
TC1	Periodic	10,000	10,000	10	1	N.A
TC2	Execution Time	100	N.A	N.A	N.A	N.A
TC3	Repetition	10,000	10,000	10	1	N.A
TC4	Execution Time	100	N.A	N.A	N.A	N.A
TC5	Sporadic	10,000	10,000	10	1	N.A
TC6	Execution Time	100	N.A	N.A	N.A	N.A
TC7	Execution Time	200	N.A	N.A	N.A	N.A
TC8	Input Synchronization	N.A	N.A	N.A	N.A	20
TC9	Output Synchronization	N.A	N.A	N.A	N.A	60
TC10	Execution Time	120	N.A	N.A	N.A	N.A
TC11	Periodic	10,000	10,000	10	1	N.A
TC12	Execution Time	100	N.A	N.A	N.A	N.A
TC13	Periodic	10,000	10,000	10	1	N.A
TC14	Execution Time	100	N.A	N.A	N.A	N.A
TC15	Periodic	10,000	10,000	10	1	N.A
TC16	Execution Time	100	N.A	N.A	N.A	N.A
TC17	Periodic	10,000	10,000	10	1	N.A
TC18	Execution Time	100	N.A	N.A	N.A	N.A
TC19	Periodic	10,000	10,000	10	1	N.A
TC20	Execution Time	100	N.A	N.A	N.A	N.A
TC21	Periodic	10,000	10,000	10	1	N.A
TC22	Execution Time	100	N.A	N.A	N.A	N.A
TC23	Periodic	10,000	10,000	10	1	N.A
TC24	Execution Time	100	N.A	N.A	N.A	N.A
TC25	Periodic	10,000	10,000	10	1	N.A
TC26	Execution Time	100	N.A	N.A	N.A	N.A
TC27	Periodic	10,000	10,000	10	1	N.A
TC28	Execution Time	200	N.A	N.A	N.A	N.A
TC29	Periodic	10,000	10,000	10	1	N.A
TC30	Execution Time	200	N.A	N.A	N.A	N.A
TC31	Periodic	10,000	10,000	10	1	N.A
TC32	Execution Time	200	N.A	N.A	N.A	N.A
TC33	Periodic	10,000	10,000	10	1	N.A
TC34	Execution Time	200	N.A	N.A	N.A	N.A
TC35	Execution Time	120	N.A	N.A	N.A	N.A
TC36	Execution Time	120	N.A	N.A	N.A	N.A
TC37	Execution Time	120	N.A	N.A	N.A	N.A
TC38	Execution Time	120	N.A	N.A	N.A	N.A
TC39	Strong Delay	10,000	20,000	N.A	N.A	N.A
TC40	Age	20,000	30,000	N.A	N.A	N.A
TC41	Reaction	20,000	40,000	N.A	N.A	N.A

N.A not available or not applicable

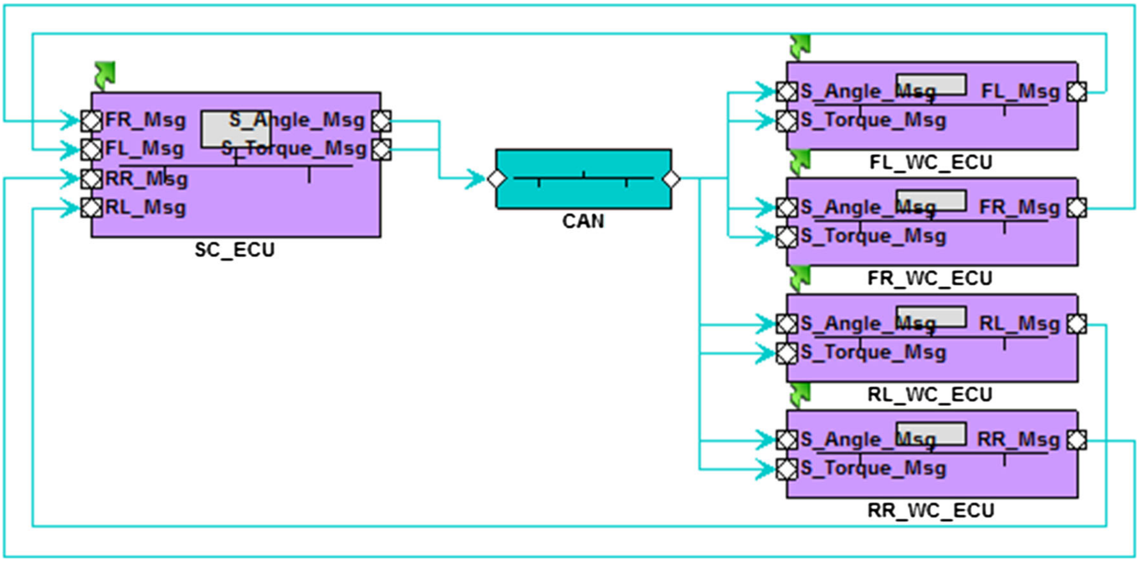


Fig. 43 Refined software architecture of the SBW system at the implementation level

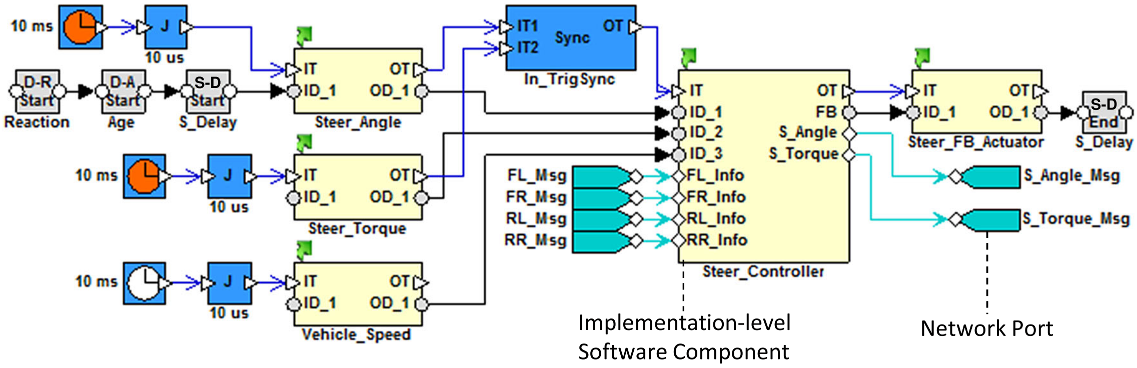


Fig. 44 Refined software architecture of the SC ECU at the implementation level

Each `Execution Time` constraint is refined by specifying it on the behavior of the corresponding component in a similar fashion as it is done in Fig. 26. The `Sporadic` constraint, `TC5`, is refined to the sporadic clock and jitter objects that are connected to the input trigger port of the Vehicle_Speed component in Fig. 44. The `Repetition` constraint, `TC3`, is refined to the periodic clock and jitter objects that are connected to the input trigger port of the Steer_Torque component in Fig. 44. The `Input Synchronization` constraint, `TC8`, is refined to the `In_TrigSync` object in Fig. 44. The `Output Synchronization` constraint, `TC9`, is refined to the `Out_TrigSync` object in Fig. 45. There are four `Out_TrigSync` objects in Fig. 45. Since we use the same usage name for these objects, they correspond to only one `Out_TrigSync` object at run-time (see Sect. 4.17.3 for details). The `Strong Delay` constraint, `TC39`, is refined to the `S_Delay Start` and `S_Delay End` objects in Fig. 44. The `Age` constraint, `TC40`, is refined to the `Age Start` and `Age End` objects in Figs. 44 and 45, respectively. The `Age Start` and `Age End` objects have the same usage name. Similarly, the `Reaction` constraint, `TC41`, is refined to the `Reaction Start` and `Reaction End` objects in Fig. 44 and 45, respectively. The `Reaction Start` and `Reaction End` objects also have the same usage name.

6.5 Verification of the timing constraints and discussion

We use the analysis engines provided by the Rubus-ICE tool suite to verify whether the specified timing constraints are satisfied or not. The periodic and sporadic activations of tasks (run-time entities corresponding to software components) can be implemented at the user or kernel level in a real-time operating system (RTOS). The Rubus RTOS supports such activations at the kernel level. This means that it guarantees strict periodic and sporadic clocks. Hence, all the `Periodic`, `Sporadic` and `Repetition` constraints, specified on the SBW system, are satisfied by construction

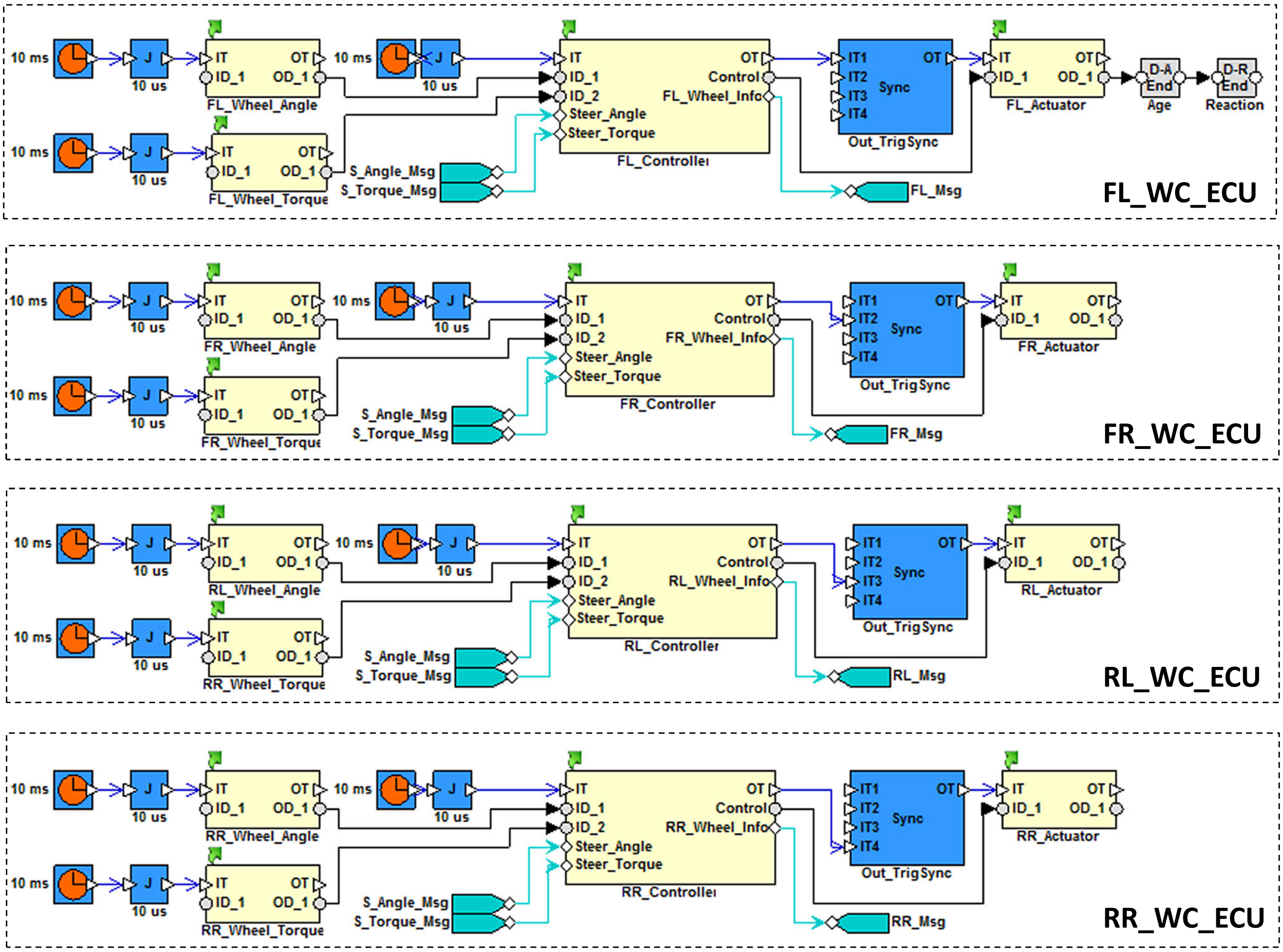


Fig. 45 Refined software architectures of the four WC ECUs at the implementation level

if the Rubus RTOS is used. RCM and its run-time framework consider both best- and worst-case execution times of the tasks. The tasks are not allowed to overrun as compared to the specified worst-case execution times. Hence, all the `Execution Time` constraints, specified on the SBW system, are satisfied by using such restrictions.

The Rubus RTOS uses offline scheduling on top of the fixed-priority scheduling [46,47]. Using the offline scheduling, all the tasks (corresponding to the components on which the `Input Synchronization` constraint is specified) are placed next to each other in the schedule. Hence, the static scheduler along with the priority assignment policy can provide guarantees for meeting the `Input Synchronization` constraint (identified as `TC8` in Fig. 42). The `Output Synchronization` constraint can be verified by performing the end-to-end delay analysis [13] on the four chains on which `TC9` is specified. According to the analysis engines, the output data are available at the data output ports of the FL_Controller, FR_Controller, RL_Controller and RR_Controller components at time 23,320 µs. Interestingly, the delay variation in the output of the four chains is 0 which is well below the tolerance parameter associated with `TC9`. The `Strong Delay`, data `Age` delay and data `Reaction` delay calculated by the end-to-end delay analysis engines are equal to 10,640, 23,440 and 33,440 µs, respectively. By comparing these delays with `TC39`, `TC40` and `TC41`, we can see that the specified timing constraints are satisfied.

7 Conclusion and future work

We have extended our previous approach to support the representation of the end-to-end timing models at a higher abstraction level compared to the level where the software architecture is implemented. The purpose is to support the end-to-end timing analysis at the higher abstraction level and at an earlier phase during the development of component-based vehicular distributed embedded systems. At the higher level, the approach provides a representation of the timing information that is extracted from the models developed with the EAST-ADL and TADL2 languages using the TIMMO methodology, whereas at the lower level, it uses the Rubus Component Model (RCM) to represent the timing information that cannot be clearly specified at the higher level. As part of this approach, we have provided an interpretation of the TADL2 timing constraints in RCM. We have also proposed extensions to RCM for the unambiguous refinement of these constraints. Moreover, we have discussed the challenges and issues that are faced during the representation of the timing information at the higher abstraction level. We have presented the guidelines and solutions to deal with these challenges. Finally, we have modeled and analyzed the timing of a vehicular-application case study to provide a proof of concept for our approach. The challenges and corresponding solutions presented in this paper can be applied to other modeling technologies that comply with the EAST-ADL methodology at the higher abstraction levels. The proposed approach is suitable for any implementation level modeling technology that supports a pipe-and-filter style for the communication among its software components, differentiates between the control and data flows among its software components and allows representation of the low-level details at the higher abstraction levels (e.g., linking information in distributed chains).

In TADL2, time can be expressed in multiple time bases, e.g., chronometric time, angular time, revolution per minute and time expressed in distance or rotation of a crank shaft. Furthermore, time can also be expressed as algebraic expressions and parameterized expressions between different time bases using the symbolic timing expression [2]. It can be an interesting future work to extend our approach by supporting the timing expressions that are based on multiple time bases.

Acknowledgements The work in this paper is supported by the Swedish Foundation for Strategic Research, ARTEMIS and the Swedish Knowledge Foundation through the projects PRESS, CRYSTAL and PreView respectively. The authors would like to take the opportunity to thank the industrial partners Arcticus Systems, Volvo CE, Volvo GTT and BAE Systems Hägglunds, Sweden.

References

1. Mubeen, S., Mäki-Turja, J., Sjödin, M.: Translating timing constraints during vehicular distributed embedded systems development. In: 1st International Workshop on Model-Driven Engineering for Component-Based Software Systems, Sep 2014
2. Timing Augmented Description Language (TADL2) syntax, semantics, metamodel Ver. 2, Deliverable 11, Aug 2012
3. Hänninen, K., et al.: The Rubus component model for resource constrained real-time systems. In: 3rd IEEE International Symposium on Industrial Embedded Systems, June 2008
4. Thorngren, P.: Keynote Talk: Experiences from EAST-ADL Use, EAST-ADL Open Workshop, Gothenburg, Oct 2013
5. Henzinger, T.A., Sifakis, J.: The embedded systems design challenge. In: Proceedings of the 14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science (Springer, Berlin, 2006). pp. 1–15
6. Crnkovic, I., Larsson, M.: Building Reliable Component-Based Software Systems. Artech House Inc, Norwood (2002)
7. AUTOSAR Technical Overview, Release 4.1, Rev. 2, Ver. 1.1.0., The AUTOSAR Consortium, Oct 2013. http://autosar.org
8. TIMMO Methodology, Ver. 2, TIMMO (TIMing MOdel), Deliverable 7, Oct 2009, The TIMMO Consortium
9. TIMMO-2-USE. https://itea3.org/project/timmo-2-use.html
10. CRYSTAL—CRitical sYSTem engineering AcceLeration. http://www.crystal-artemis.eu. Accessed Mar 2016
11. Model-based Analysis & Engineering of Novel Architectures for Dependable Electric Vehicles (MAENAD) Project. http://www.maenad.eu. Accessed Mar 2016
12. Tindell, K., Clark, J.: Holistic schedulability analysis for distributed hard real-time systems. Microprocess. Microprogram. **40**, 117–134 (1994)
13. Mubeen, S., Mäki-Turja, J., Sjödin, M.: Support for end-to-end response-time and delay analysis in the industrial tool suite: issues, experiences and a case study. Comput. Sci. Inf. Syst. **10**(1), 1361–1384 (2013)
14. Mubeen, S., Mäki-Turja, J., Sjödin, M.: Communications-oriented development of component-based vehicular distributed real-time embedded systems. J. Syst. Archit. **60**(2), 207–220 (2014)
15. Sentilles, S., Vulgarakis, A., Bures, T., Carlson, J., Crnkovic, I.: A component model for control-intensive distributed embedded systems. In: 11th International Symposium on Component Based Software Engineering. Springer, Berlin, 2008). pp. 310–317
16. Ke, X., Sierszecki, K., Angelov, C.: COMDES-II: a component-based framework for generative development of distributed real-time control systems. In: 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007). pp. 199 –208, Aug 2007
17. Catalog of Specialized CORBA Specifications. OMG Group. http://www.omg.org/technology/documents/
18. Mubeen, S., Nolte, T., Lundbäck, J., Gålnander, M., Lundbäck, K-L.: Refining timing requirements in extended models of legacy vehicular embedded systems using early end-to-end timing analysis. In: 13th International Conference on Information Technology: New Generations (ITNG), Apr 2016
19. Mubeen, S., Sjödin, M., Nolte, T.,Lundbäck, J., Gålnander, M., Lundbäck, K.-L.: End-to-end timing analysis of black-box models in legacy vehicular distributed embedded systems. In: 21st International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Aug 2015
20. OMG Systems Modeling Language, version 1.3. http://www.omgsysml.org
21. EAST-ADL Domain Model Specification, Version V2.1.12, Version V2.1.12, Deliverable 11, Aug 2012,http://www.east-adl.info/Specification/V2.1.12/EAST-ADL-Specification_V2.1.12.pdf. Accessed Mar 2016
22. TIMMO-2-USE Methodology Description, Ver. 2, Del. 13, July 2012
23. Rubus ICE-Integrated Development Environment. http://www.arcticus-systems.com
24. Feiler, P., Lewis, B., Vestal, S., Colbert, E.: An overview of the SAE Architecture Analysis & Design Language (AADL) standard: a

basis for model-based architecture-driven embedded systems engineering. In: Architecture Description Languages. The International Federation for Information Processing (IFIP), vol. 176. (Springer, New York, 2005), pp. 3–15
25. SCADE Suite. http://www.esterel-technologies.com/products/scade-suite. Accessed Mar 2016
26. The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems. http://www.omgmarte.org/. Jan 2010
27. MAST—Modeling and Analysis Suite for Real-Time Applications. http://mast.unican.es
28. CHESS Project, CHESS consortium. http://www.chess-project.org. Accessed Mar 2016
29. Cicchetti, A., Ciccozzi, F., Mazzini, S., Puri, S., Panunzio, M., Vardanega, T., Zovi, A.: Chess: a model-driven engineering tool environment for aiding the development of complex industrial systems. In: 27th International Conference on Automated Software Engineering (ASE 2012), Sep 2012
30. Rubus models, methods and tools. http://www.arcticus-systems.com
31. Chen, D., Feng, L., Qureshi, T., Lönn, H., Hagl, F.: An architectural approach to the analysis, verification and validation of software intensive embedded systems. Computing **95**(8), 649–688 (2013)
32. ISO 26262-1:2011: Road vehicles Functional safety. http://www.iso.org/
33. Mastering Timing Information for Advanced Automotive Systems Engineering. In the TIMMO-2-USE Brochure, 2012. http://www.timmo-2-use.org/pdf/T2UBrochure
34. TADL: Timing Augmented Description Language, Ver. 2, Deliverable 6, Oct 2009
35. Ohno, A., Azumi, T., Nishio, N.: TECS components providing functionalities of OSEK specification for ITRON OS. J. Inf. Process. **22**(4), 584–594 (2014)
36. Carlson, J.: Timing analysis of component-based embedded systems. In: 15th International ACM SIGSOFT Symposium on Component Based Software Engineering. ACM, June 2012
37. Hill, H.: CUTS: a system execution modeling tool for realizing continuous system integration testing. In: 32nd ACM/IEEE International Conference on Software Engineering, May 2010
38. Feiertag, N., Richter, K., Nordlander, J., Jonsson, J.: A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. In: International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS), Dec 2008
39. EAST-ADL Tooling. http://www.east-adl.info/Tooling.html. Accessed Mar 2016
40. Mubeen, S., Mäki-Turja, J., Sjödin, M.: Extraction of end-to-end timing model from component-based distributed real-time embedded systems. In: Time Analysis and Model-Based Design, from Functional Models to Distributed Deployments (TiMoBD) Workshop located at Embedded Systems Week (Springer, Berlin, 2011), pp. 1–6
41. Mubeen, S., Nolte, T., Sjödin, M., Lundbäck, J., Gålnander, M., Lundbäck, K.-L.: Modeling of legacy distributed embedded systems at vehicle abstraction level. In: 19th International Symposium on Component Based Software Engineering, Apr 2016
42. Bucaioni, A., Cicchetti, A., Ciccozzi, F., Eramo, R., Mubeen, S., Sjödin, M.: Anticipating implementation-level timing analysis for driving design-level decisions in east-adl. In: International Workshop on Modelling in Automotive Software Engineering, Sep 2015
43. Mubeen, S., Mäki-Turja, J., Sjödin, M.: Towards translation of timing constraints during vehicular embedded systems development. In: International Conference on Component-Based Software Engineering and Software Architecture (CompArch) (Springer, Berlin, 2014)
44. Qureshi, T., Chen, D., Lönn, H., Törngren, M.: From EAST-ADL to AUTOSAR software architecture: a mapping scheme. In: Software Architecture, Lecture Notes in Computer Science, vol 6903 (2011), pp. 328–335
45. ISO 11898-1. Road vehicles interchange of digital information controller area network (CAN) for high-speed communication. ISO Standard-11898, Nov 1993
46. Audsley, N., Burns, A., Davis, R., Tindell, K., Wellings, A.: Fixed priority pre-emptive scheduling: an historic perspective. Real-Time Syst. **8**(2/3), 173–198 (1995)
47. Mäki-Turja, J., Hänninen, K., Nolin, M.: Efficient development of real-time systems using hybrid scheduling. In: International Conference on Embedded Systems and Applications, June 2005

Saad Mubeen Dr. Mubeen is a Senior Lecturer/Assistant Professor at the School of Innovation, Design and Engineering at Mälardalen University, Sweden. He is a senior member of IEEE. He received his PhD in Computer Science and Engineering from the same university in 2014. Saad has formerly worked as a software engineer and a consultant for Arcticus Systems and Volvo Construction Equipment, Sweden, respectively. His research interests include model-based development of vehicular embedded systems with a focus on timing models, end-to-end timing analysis and multicore platforms. Saad has co-authored over 85 research publications in international peer-reviewed journals, conferences, workshops and book chapters.

Thomas Nolte Professor Nolte is leading the Complex Real-time Embedded Systems research group at Mälardalen University, Sweden. He is a senior member of IEEE. He was awarded a B.Eng., an M.Sc., a Licentiate and a Ph.D. degree in Computer Engineering from the same university in 2001, 2002, 2003 and 2006, respectively. He has been a Visiting Researcher at University of California, Irvine (UCI), Los Angeles, USA, in 2002, and a Visiting Researcher at University of Catania, Italy, in 2005. He has been a Postdoctoral Researcher at University of Catania in 2006 and at Mälardalen University in 2006–2007. He has co-authored over 300 research publications in peer-reviewed conferences, workshops, books and journals.

Mikael Sjödin Dr. Sjödin is a professor of real-time system and research director for Embedded Systems at Mälardalen University, Sweden. His research is focused on the development of new methods that will make embedded software development cheaper, faster and yield software with higher quality. Concurrently, he is also pursuing research in analysis of real-time systems, where the goal is to find theoretical models for real-time systems that will allow their timing behavior and memory consumption to be calculated. He received his PhD in computer systems in 2000 from Uppsala University (Sweden). Since then he has been working in both academia and industry with embedded systems, real-time systems and embedded communications. Previous affiliations include Newline Information, Melody Interactive Solutions and CC Systems. He has co-authored over 240 research publications in peer-reviewed conferences, workshops, books and journals.

John Lundbäck He holds a Master degree from the Royal Institute of Technology (KTH), Sweden. He has been working as a manager for the tool chain development at Arcticus Systems since 1999. He is also leading the research unit at Arcticus Systems. He has co-authored over 15 research publications in international peer-reviewed journals, conferences and workshops.

Kurt-Lennart Lundbäck Dr. Lundbäck has been active in the field of computing since 1972 with broad experience in industrial environments. He is experienced in many facets of computing and computer-based systems including systems and software engineering, programming languages and compilers, operating systems as well as various real-time application domains. He founded Arcticus Systems AB, a Swedish corporation in 1985. Currently, he is the CEO of the company. The goal of Arcticus Systems is to offer methods and tools to its customers in the vehicle industry, which provide cost-effective development and support safety critical, non-safety critical and mixed embedded systems. He has acted as one of the hubs in transferring research results from academia to the Swedish software industry especially in the vehicular domain. Dr. Lundbäck has co-authored over 20 research publications in international peer-reviewed journals, conferences and workshops.