Component-Based Vehicular Distributed Embedded Systems: End-to-end Timing Models Extraction at Various Abstraction Levels

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

Abstract

In order to perform the end-to-end response-time and delay analyses of a system, its end-to-end timing model should be available. The majority of existing model- and component-based development approaches for vehicular distributed embedded systems extract the end-to-end timing model at an abstraction level and development phase that is close to the system implementation. We present a method to extract the end-to-end timing models from the systems at a higher abstraction level. At the higher level, the method extracts timing information from system models that are developed with EAST-ADL and Timing Augmented Description Language (TADL2) using the TIMMO methodology. At the lower level, the method exploits the Rubus component model to extract the timing information that cannot be clearly specified at the higher level such as trigger paths in distributed chains. We also discuss challenges and issues faced during extraction of the timing models. Further, we present guidelines and solutions to address these challenges.

I. Introduction

Due to increase in the amount of advanced computer controlled functionality in vehicular distributed embedded systems, the size and complexity of embedded software has drastically increased in the past few years. For example, the embedded software in heavy vehicle architectures such as a truck may consist of as many as 2000 software functions that may be distributed over 45 ECUs (Electronic Control Units) [1]. In order to deal with the software complexity, the research community proposed model- and component-based development of embedded real-time systems by using the principles of Model-Based Software Engineering (MBSE) and Component-Based Software Engineering (CBSE) [2], [3]. This approach is intended to capture requirements early during the development, lower development cost, enable faster turn-around times in early design phases, increase reusability, support modeling at higher abstraction levels, and to 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. Whereas, CBSE supports the development of large software systems by integration of software components. It raises the level of abstraction for software development and aims to reuse software components and their architectures. Within the segment of construction-equipment vehicles and similar segments for heavy special-purpose vehicles, model-based development of software architectures for embedded real-time systems has had a surge the last few years.

A. Problem Statement and Paper Contributions

Most of the vehicular functions are developed as distributed embedded systems with real-time requirements. Hence, the providers of these systems are required to ensure that the actions by the systems will be taken at a time that is appropriate to their environment. One way to guarantee that the system will meet all its deadlines is to perform the end-to-end response-time and delay analyses [4], [5]. These analyses can validate timing requirements, specified on the system, without performing exhaustive testing. In

order to perform the timing analyses of the system, an end-to-end timing model needs to be available. The end-to-end timing model consists of the information containing timing properties, requirements and dependencies concerning all tasks, messages, task chains and distributed transactions in the system. Based on this information, execution behavior of the system, with respect to timing, can be predicted by means of certain type of timing analysis such as end-to-end response-time and delay analysis.

The majority of existing approaches for component-based vehicular distributed embedded systems support the extraction of the timing models at an abstraction level and development phase that is close to their implementation [6], [7], [8], [9], [10]. Furthermore, high-level timing analysis provided by existing approaches does not support high precision and is often not based on actual implementation of the system. As a result, a high-precision end-to-end timing analysis cannot be performed at higher abstraction levels. In the past few years, one of the focuses of several large EU research projects, that involve both academia and industry, has been on supporting the timing analysis at various abstraction levels and parts of the development process [11], [12], [13].

Extraction of the timing model at higher abstraction levels is challenging mainly because not all timing information that is required to be captured in the timing model is available. Moreover, mismatch and incompatibilities among various methodologies, languages and tools that are used in different parts of the development process also add to the complexity of extracting the timing model. Since complete timing information may not be available at higher levels, the timing analysis results may not represent accurate timing behavior of the final system. However, these results can provide useful information that can guide further model refinement and implementation.

We envision the extraction of end-to-end timing model and performing high-precision end-to-end timing analysis at higher levels of abstraction to be state of the practice in the future. We believe, timing information will be formally modeled at higher abstraction levels in the vehicular industry. In that case, we need to extract the specified timing information at 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 system implementation.

We have experienced that timing information is modeled at higher abstraction levels in the vehicular industry. This may be carried out using SysML language [14]. However, it is done mostly in an informal and textual way; which cannot be used for any formal timing analysis. Today, Timing Augmented Description Language (TADL2) [15] provides the only viable formal method for modeling the timing information using timing constraints at various abstraction levels. In order to extract a complete end-to-end timing model and perform a high-precision timing analysis, TADL2 has to be combined with a lower abstraction level execution modeling approach such as the Rubus Component Model (RCM) [16]. We hope the industry will start using TADL2. If they do so, we can reuse that information to perform high-precision end-to-end response-time and delay analyses at a higher abstraction level.

In our previous work [8], we presented a method to extract end-to-end timing models at lower abstraction levels. In this paper, we extend our previous method to extract the timing models at a higher abstraction level. At the higher level, the method extracts timing information from system models that are developed with EAST-ADL [17] using the TIMMO methodology [18]; and annotated with timing information using TADL2. At the lower level, the method exploits RCM and its tool suite Rubus-ICE [19] to extract the timing information that cannot be clearly specified at the higher level, e.g., trigger paths in distributed chains. However, it is not straightforward to combine TADL2 with RCM due to several challenges such as unambiguous transformation of TADL2 timing constraints in RCM; and unambiguous extraction of control and data paths at the higher level. The main focus of this paper is to attack these challenges. As a contribution, we provide an interpretation of TADL2 timing constraints in RCM. Moreover, we propose extensions in RCM for unambiguous transformation of TADL2 timing

constraints.

We choose RCM instead of AUTOSAR [6] at the lower abstraction level. This is because AUTOSAR lacks a complete timing model, e.g., control flow is not specified in an unambiguous way. The work in this paper is a step towards a bigger goal, i.e., development of a seamless tool-chain for model-based development of vehicular distributed real-time systems; and support for inter-operating various modeling and analysis tools, including the AUTOSAR-based tool chain [13].

B. Paper Layout

The rest of the paper is organized as follows. In Section II, we discuss background and related work. In Section III, we discuss an interpretation of TADL2 constraints in RCM. In Section IV we discuss other challenges and corresponding solutions. Finally, Section V summarizes the paper and presents the future work.

II. Background and Related Works

A. Abstraction Levels Considered by Various Methodologies

There are several frameworks that support timing modeling such as AADL [20], SCADE [21], MARTE [22], MAST [23], SysML, CHESS [24], [25]. However, the focus in the vehicle industry, especially in the segment of construction-equipment vehicles and other heavy road vehicle architectures, today is on EAST-ADL and AUTOSAR; Rubus is also being used. In this work, we focus only on the vehicular domain. Various models and methodologies used for the development of vehicular distributed embedded systems [17], [15], [6], [26] consider four *abstraction levels* shown in Figure 1.

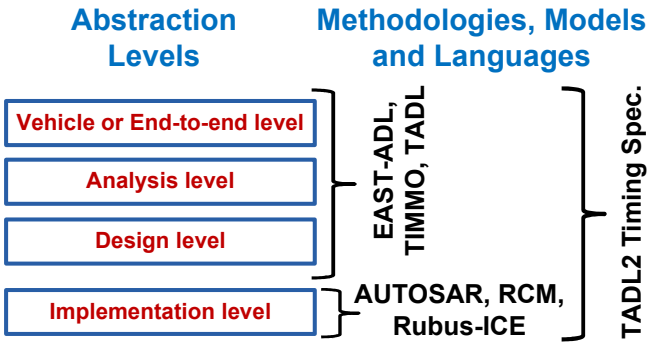

Figure 1. Abstraction levels considered during the development

1) Vehicle or end-to-end level: At the vehicle level, requirements, functionality and features of the vehicle are captured in an informal (often textual) and solution-independent way. This level captures the information regarding what the system should do [27]. In the segment of construction-equipment vehicles, this abstraction level is better known as the end-to-end level because features and requirements on the end-to-end functionality of the machine or vehicle are captured in an informal way.

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

3) Design level: The artifacts at this level are developed in an implementation-independent way. These artifacts are refined from the analysis level artifacts; and also contains middleware abstraction and hardware architecture. In addition, software functions to hardware allocation may be present.

4) Implementation level: At the implementation level, the design-level artifact is refined to software-based implementation of the system functionality. The EAST-ADL methodology defines a system at this level in terms of AUTOSAR elements. However, in this work, our focus is on using RCM and its development environment Rubus-ICE at the implementation level. Hence, the artifact at this level consists of software architecture of the system defined in terms of Rubus components and their interactions.

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

B. Models and Development Methodologies

We focus on some of the component technologies that are used for the development of distributed embedded systems in the vehicular domain.

1) Rubus Component Model (RCM) and Rubus-ICE: Rubus [28] is a collection of methods and tools for model- and component-based development of dependable embedded real-time systems. It is developed by Arcticus Systems [1] in close collaboration with several industrial partners. Rubus is today mainly used for the development of control functionality in vehicles by several international companies, e.g., BAE Systems Hägglunds[2], Volvo Construction Equipment[3], Knorr-bremse[4], Mecel[5] and Hoerbiger[6]. The Rubus concept is based around RCM and its development environment Rubus-ICE which includes modeling tools, code generators, analysis tools and run-time infrastructure. The overall goal of Rubus is to be aggressively resource efficient and to provide means for developing predictable, timing analyzable and synthesizable control functions in resource-constrained embedded systems. The timing analysis supported by Rubus-ICE includes distributed end-to-end response-time and delay analyses [5]. Rubus methods and tools mostly focus at the implementation level in Figure 1.

The lowest-level hierarchical component in RCM is called Software Circuit (SWC). Its purpose is to encapsulate basic functions. Figure 2 shows an example of a software architecture in RCM composed of SWCs; interconnections between SWCs; and their interactions with external events and actuators with regard to both data and triggering.

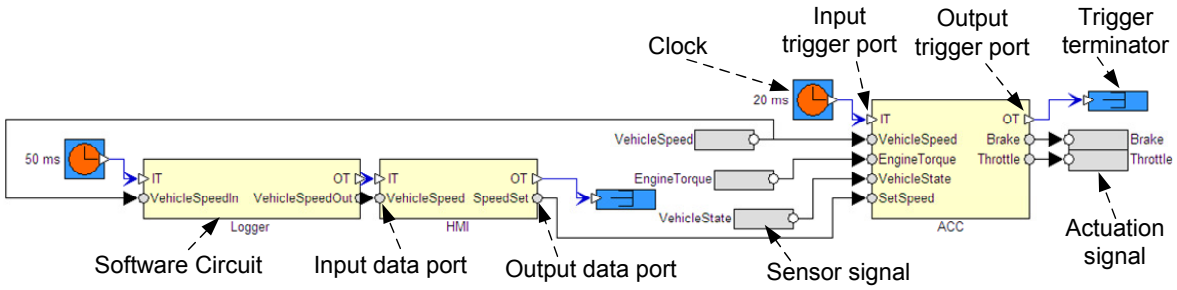

Figure 2. Example of software architecture of a system modeled in RCM.

2) AUTOSAR: AUTOSAR [29] is an industrial initiative to provide standardized software architecture for the development of embedded software. It is used at the implementation level in Figure 1. It describes software development at a higher level of abstraction compared to RCM. Unlike RCM, it does not

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

separate control and data flows among components within a node. It does not differentiate between the modeling of intra- and inter-node communication which is unlike RCM. The timing model in AUTOSAR is introduced fairly recently compared to that of Rubus. There are some similarities between AUTOSAR and RCM, e.g., the sender receiver communication in AUTOSAR resembles the pipe-and-filter communication in RCM. AUTOSAR is more focussed on the functional and structural abstractions, hiding the implementation details about execution and communication. AUTOSAR hides the details that RCM highlights.

3) EAST-ADL, MARTE, TIMMO, TIMMO-2-USE, TADL and TADL2: TIMMO [11] is an initiative to provide AUTOSAR with a timing model [30]. It is based around a methodology and TADL [26] language. TADL is used to express timing requirements and constraints. It is inspired by MARTE [22] which is a UML profile for model-driven development of real-time and embedded systems. TIMMO methodology uses EAST-ADL language [17] for structural modeling and AUTOSAR for the implementation. TIMMO and EAST-ADL focus on the top three levels in Figure 1. However, TIMMO methodology uses AUTOSAR at the implementation level.

In TIMMO-2-USE project [12], a major redefinition of TADL is done and released in TADL2 specification [15]. TADL2 can specify timing related information at all abstraction levels shown in Figure 1. Most of these initiatives lack the support on expressing low-level details at the higher levels such as linking information in distributed chains. These details are necessary to extract the end-to-end timing model from the architecture. Furthermore, there is no support on how to extract this information from the model or perform timing analysis. In our view, the end-to-end timing model includes enough information from the systems to be able to perform certain type of timing analysis, e.g., end-to-end response-time analysis.

To the best of our knowledge, none of the existing approaches are able to extract the end-to-end timing model and perform corresponding pre run-time analysis at higher levels of abstraction. This is one of the objectives in an ongoing EU research project [13].

4) COMDES and ProCom: COMDES-II [9] is a two-level component-based framework for the development of distributed embedded control systems. Unlike RCM, COMDES-II employs signal-based communication for both intra- and inter-node interactions. COMDES-II does not include explicit components to model network communication.

ProCom [7] is a two-layered component model for the development of distributed embedded systems. It is inspired by RCM and there are a number of similarities between the two. For example, both have passive components, both separate control flow from the data flow, and both use a pipe-and-filter style of communication mechanism for components interconnection. However, ProCom does not differentiate between intra- and inter-node communication which is unlike RCM. ProCom hides communication details, whereas RCM lifts them up to the modeling level.

ProCom supports timing analysis [31], however, the analysis is not performed with such a high precision as it is done in Rubus-ICE. Moreover, the timing analysis in ProCom does not include execution models. It will be very hard in ProCom and COMDES-II to extract the timing model and perform end-to-end timing analysis with the precision that is done in RCM.

5) Previous Works: In our previous works [8], [32], we presented a method to automatically extract the end-to-end timing models only at the lowest abstraction level shown in Figure 1. In this paper, we extend our previous method to raise the extraction of end-to-end timing models at the design level. Since, we aim to extract the timing information from a higher abstraction level, we need to explicitly capture timing requirements and constraints. For this purpose we provide unambiguous interpretation and transformation of TADL2 timing constraints in RCM.

III. Interpretation of TADL2 Timing Constraints in RCM

In the first subsection, we define some terms and notations. 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 [15]. Moreover, we interpret and transform these timing constraints in RCM.

A. Definitions and Notations

In TADL2, timing requirements are specified by means of timing constraints on events and event chains [18]. Constraints are used to put restriction on, e.g., delays between a pair of events; repetition of an event; 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. An event is used to trigger an analysis- or design-level function. When the function is triggered, input data is 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. In order to clarify the notations used to define timing constraints in the following subsections, consider the following example.

Example: Assume a timing constraint is denoted by TC. It can be specified on some events. Let us denote two such events by source and target. We use object-oriented notation to define the attributes of the constraint. For example, TC.source refers to the source event on which TC is specified. There can be any number of occurrences of an event. Let us denote an occurrence of event TC.source by an attribute s. 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 lie 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 iff[7] for every occurrence s of TC.source, there is an occurrence t of TC.target such that

$$\text{TC.lower} \leq (\text{t- s}) \leq \text{TC.upper}$$

This means, the timing constraint TC is satisfied 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 is smaller than or equal to the time distance specified by the higher attribute.

B. Delay Constraint

1) TADL2 Description: It constrains the distance between occurrences of source and target events. It does not matter if matching target occurrence is caused by the corresponding source occurrence or not.

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

$$\text{DC.lower} \leq (\text{t- s}) \leq \text{DC.upper}$$

[7]if and only if

3) Interpretation in RCM: There is no existing support in RCM to specify this constraint.

We propose the addition of a new timing constraint with the above-mentioned semantics, denoted by Delay, in RCM. Since this constraint corresponds to the distance between occurrences of source and target events, we associate two objects with it, namely "Delay Start" and "Delay End" as shown in Figure 3. The Delay Start object can be specified at the Data Input Port (DIP) of source SWC. The triggering of Trigger Input Port (TIP) of source SWC corresponds to a new occurrence of 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 target SWC. Production of a trigger at the Trigger Output Port (TOP) of target SWC corresponds to a new occurrence of target event. The lower and upper values of the constraint can be specified on Delay End object.

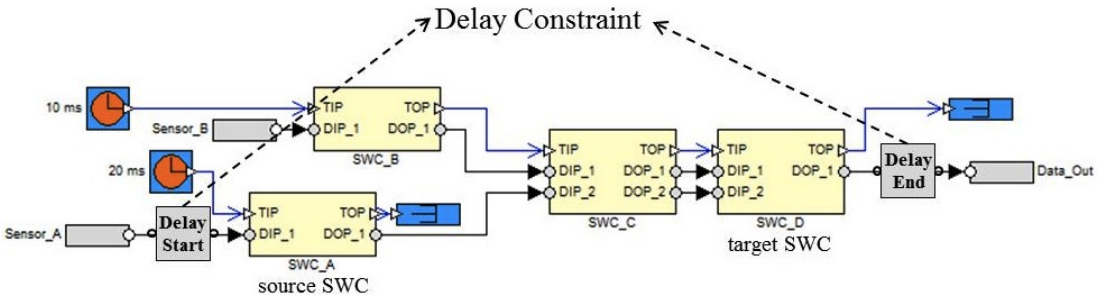

Figure 3. Proposed objects to specify Delay constraint in RCM

The occurrences of target event (data on 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 Figure 3 and Figure 4, the occurrences of target event corresponds to input data either from SWC_B or from both SWC_A and SWC_B. The upward arrows in Figure 4 symbolize occurrence of events. The lower and upper attributes for the Delay constraint are also identified in Figure 4. Assuming the priority of the task corresponding to SWC_A to be higher than that of SWC_B, the first occurrence of target matches the first occurrences of both SWC_B and the source. Whereas, the second occurrence of target is due to only SWC_B. As discussed earlier, matching occurrence of target with respect to occurrences of the source does not matter in this constraint.

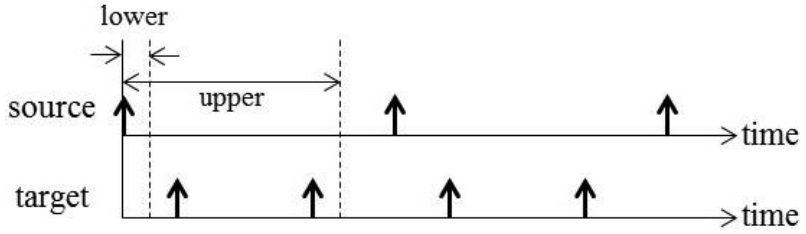

Figure 4. Graphical illustration of Delay constraint

C. Strong Delay Constraint

1) TADL2 Description: It constrains the distance between each indexed occurrence of source event and corresponding identically indexed occurrence of target event. Matching of target occurrence caused by the corresponding source occurrence is vital for this constraint.

2) Semantics: A system behavior satisfies the specified StrongDelayConstraint SDC iff the number of occurrences of SDC.source and SDC.target events is equal; and for each index i, if there is an i^{th} occurrence of SDC.source at time s these is also an i^{th} occurrence of SDC.target at time t such that

$$\text{SDC.lower} \leq (\text{t- s}) \leq \text{SDC.upper}$$

3) Interpretation in RCM: There is no existing support in RCM to specify this constraint.

We propose the addition of a new timing constraint with the above-mentioned semantics, denoted by S-Delay, in RCM. Since this constraint corresponds to the distance between two matching occurrences of source and target events, we associate two objects with it, namely "S-Delay Start" and "S-Delay End" as shown in Figure 5. As the number of occurrences of source and target events for each index are not equal in the example in Figure 3, S-Delay constraint cannot be used in place of the Delay constraint. However, it can be used on the same system if source SWC is changed as shown in Figure 5. The S-Delay Start object can be specified at the DIP of source SWC. The triggering of TIP of source SWC corresponds to a new occurrence of source event. The S-Delay End object can be specified at DOP of target SWC. The production of a trigger at the TOP of target SWC corresponds to a new occurrence of target event. The lower and upper values of the constraint can be specified on S-Delay End object. These values are identified in Figure 6. The figure also shows that occurrences of the target match with the occurrences of the source.

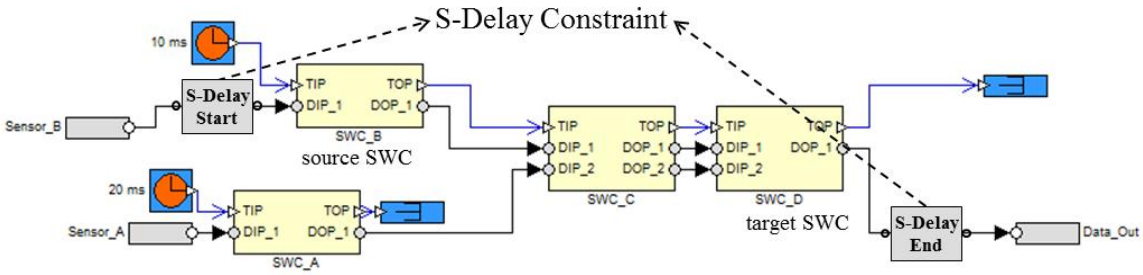

Figure 5. Proposed objects to specify Strong Delay constraint in RCM

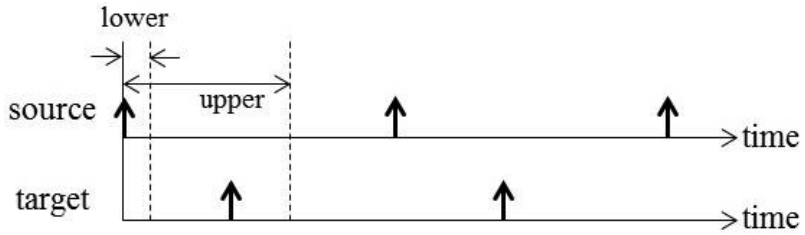

Figure 6. Graphical illustration of Strong Delay constraint

D. Reaction Constraint

1) TADL2 Description: It constrains the occurrence of a response event after the occurrence of a corresponding stimulus event in an event chain. Basically it specifies "how long after the occurrence of a stimulus a corresponding response must occur" [15]. 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 multi-rate systems, components in an event chain can be triggered with independent clocks. Hence, there can be multiple response occurrences to a single occurrence of stimulus in an event chain. In these 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 same color as that of stimulus must take place within the limits specified by this constraint as shown in Figure 7.

2) Semantics: A system behavior satisfies the specified ReactionConstraint ReaC iff for each occurrence s in ReaC.stimulus, there is an occurrence r in ReaC.response such that

$$\begin{gathered}(\text{r.color} = \text{s.color}) \\ \text{and} \\ (\text{r is minimal in ReaC.response with that color}) \\ \text{and} \\ (\text{minimum} \leq (\text{r- s}) \leq \text{maximum})\end{gathered}$$

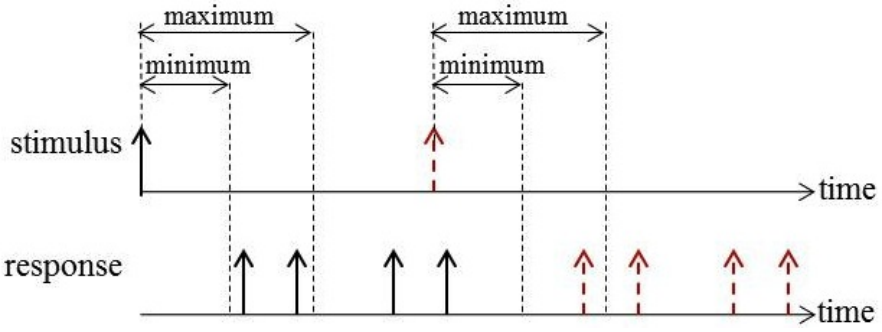

Figure 7. Graphical illustration of Reaction constraint

3) Interpretation in RCM: There is an existing support in RCM to specify the reaction constraint denoted by DataReaction (DR for short). This constraint can be specified on an event chain, event chain segment and distributed event chain (distributed over more than one node) by means of DR Start and DR End objects as shown in Figure 8. The DR End object supports the specification of maximum value by means of a deadline parameter associated to it. However, the minimum parameter is considered to be zero. In order to be consistent with TADL2 Reaction constraint, we propose to associate a parameter with DR End object to specify non-zero minimum value of the constraint.

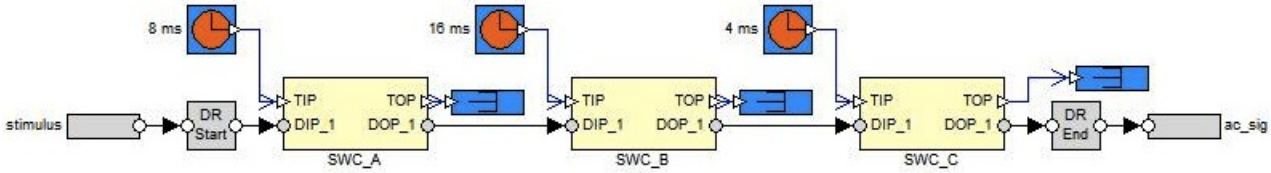

Figure 8. Existing objects in RCM to specify Reaction constraint

The analysis engines 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 Figure 8. Figure 9 shows 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_B is deliberately given an offset of 15 time units to maximize the delays. This delay is equal to the time elapsed between the previous non-overwritten release of task τ_A (input of the chain) and 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 $8ms$). 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 towards the output of the chain. Therefore, the new value is read by the third and forth instances of task τ_A. The first output corresponding to the new value (arriving just after $8ms$) appears at the output of the chain at $34ms$. This results in the delay of $26ms$ as shown in Figure 9. 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 a message that is received from the network. The analysis engines calculate the Reaction delay as shown in Figure 9 and compare it with the specified constraint parameters.

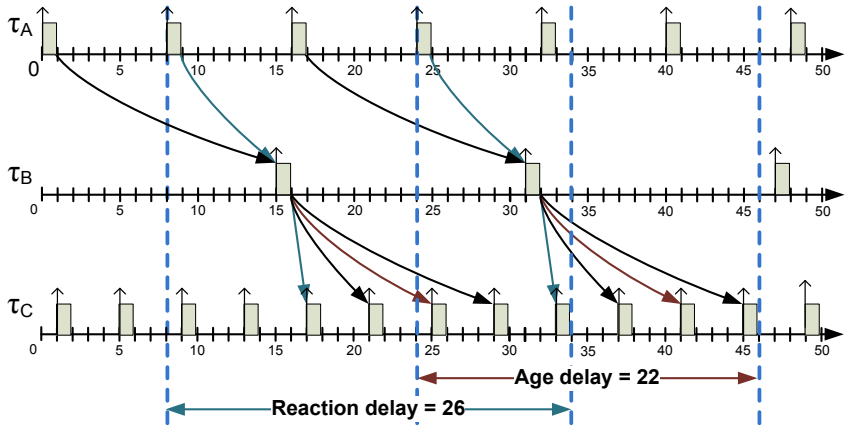

Figure 9. Demonstration of Reaction and Age delay calculations by analysis engines

E. Age Constraint

1) TADL2 Description: It constrains the occurrence of a stimulus from the occurrence of the corresponding response looking back through the event chain. Basically it specifies "how long before each response a corresponding stimulus must have occurred" [15]. In order to satisfy this constraint, the latest occurrence of the stimulus with same color as that of the response must lie within the limits specified by this constraint as shown in Figure 10. This constraint differs from the Delay constraint in a way that it can only be applied to event chains and not to individual events.

2) Semantics: A system behavior satisfies the specified AgeConstraint AgeC iff for each occurrence r in AgeC.response, there is an occurrence s in AgeC.stimulus such that

(s.color = r.color)
and
(s is maximal in AgeC.stimulus with that color)
and
(minimum $\leq$ (r- s) $\leq$ maximum)

3) Interpretation in RCM: There is an existing support in RCM to specify the age constraint denoted by DataAge. This constraint can be specified on an event chain, event chain segment and distributed event chain by means of Age Start and Age End objects as shown in Figure 11. The Age End object supports the specification of maximum value by means of a deadline parameter associated to it. However, the minimum parameter is considered to be zero. In order to be consistent with TADL2 Age constraint,

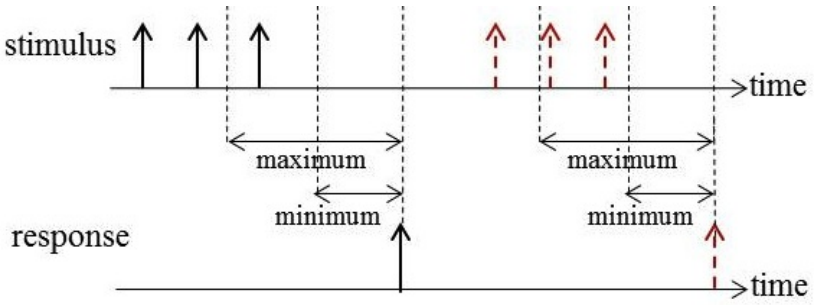

Figure 10. Graphical illustration of Age constraint

we propose to associate a parameter with Age End object to specify non-zero 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 Figure 11. Figure 9 shows the time line when this chain is executed. The analysis engines calculate the Age delay as shown in Figure 9 and compare it with the specified constraint parameters.

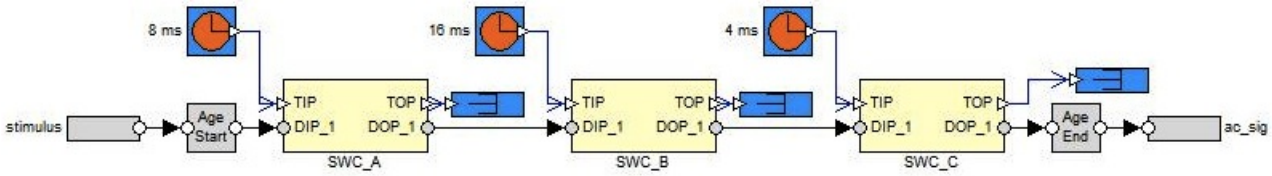

Figure 11. Existing objects in RCM to specify Age constraint

F. Repetition Constraint

1) TADL2 Description: It 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.

2) Semantics: A system behavior satisfies the specified RepetitionConstraint RC iff the following two are concurrently satisfied:

1) For each subsequence X of RC.event, if X contains span + 1 occurrences then d is the distance between the outer- and inner-most occurrences in X and

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

2) The number of occurrences of X and RC.event events is equal; and for each index i, if there is an i^{th} occurrence of X at time s these is also an i^{th} occurrence of RC.event at time t such that

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

If the span attribute is equal to one; jitter is equal to zero; and the upper and lower values are equal then the behavior becomes strictly periodic. Figure 12 graphically illustrates this constraint.

3) Interpretation in RCM: In RCM, an SWC can be time triggered or event triggered by means of TrigClockTT and TrigClockET objects. The TrigClockTT object generates periodic trigger signals with a period specified on it. Whereas, the TrigClockET object generates sporadic trigger signals with a

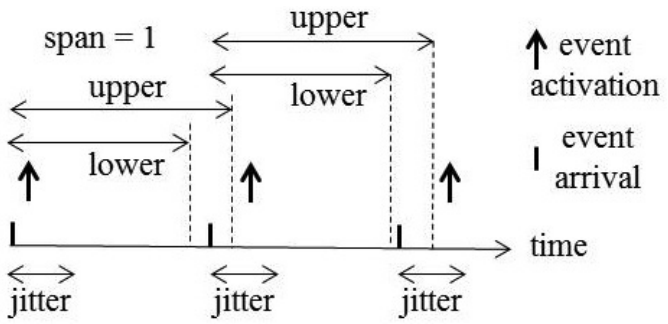

Figure 12. Graphical illustration of Repetition constraint

minimum inter-arrival time specified on it. These two objects are shown in Figure 13. There is another object in RCM called TrigJitterPeriod that provides the allowance for jitter to the trigger generating objects. Figure 13 contains two of these objects with jitter values equal to 1 millisecond and 100 microseconds.

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

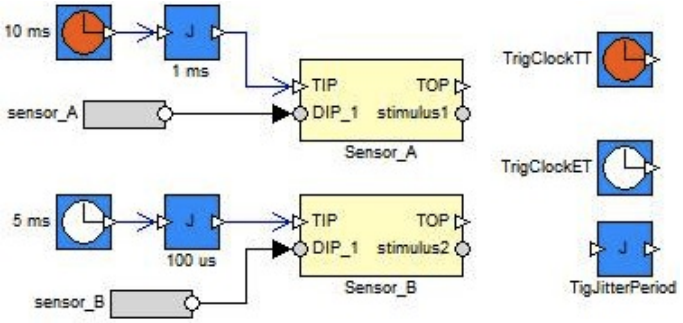

Figure 13. Existing objects in RCM to specify triggers and jitter

G. Sporadic Constraint

1) TADL2 Description: It constrains the occurrence of a sporadic event.

2) Semantics: It is a special type of Repetition constraint whose Span is equal to 1; and two subsequent activations must be separated by Minimum Inter-arrival Time (MIT). This constraint is graphically illustrated in Figure 14.

3) Interpretation in RCM: The TrigClockET object can be combined with TrigJitterPeriod to represent TADL2 Sporadic constraint as shown in Figure 15. In order to consistently interpret this constraint, we set the span parameter to 1 and assign MIT value to the period associated with the TrigClockET object. The lower and upper values can be assigned to minimum and maximum inter-arrival times. If the maximum inter-arrival time is not specified, it can be considered equal to infinity.

H. Periodic Constraint

1) TADL2 Description: It constrains the occurrence of a periodic event.

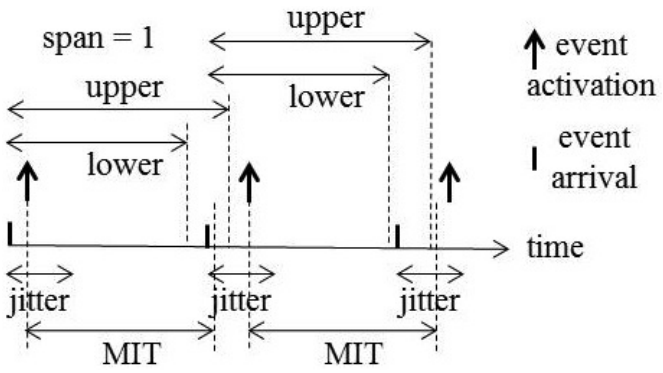

Figure 14. Graphical illustration of Sporadic constraint

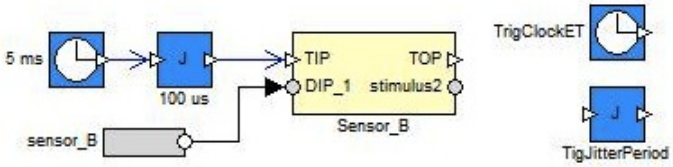

Figure 15. Equivalent to Sporadic constraint specified in RCM

2) Semantics: It is a special type of Sporadic constraint whose lower and upper attributed are equal. These attributes are assigned the value of the period. This constraint is graphically illustrated in Figure 16.

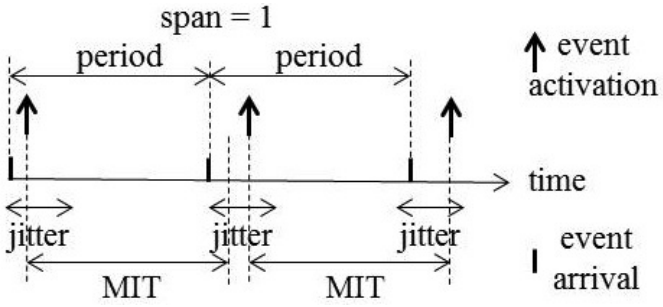

Figure 16. Graphical illustration of Periodic constraint

3) Interpretation in RCM: The TrigClockTT object can be combined with TrigJitterPeriod to represent TADL2 SporadicConstraint as shown in Figure 17. In order to consistently interpret this constraint, we set the span parameter to 1; upper and lower parameters are equal and are assigned the value of period; and assign MIT value to the period associated with the TrigClockTT object unless specified.

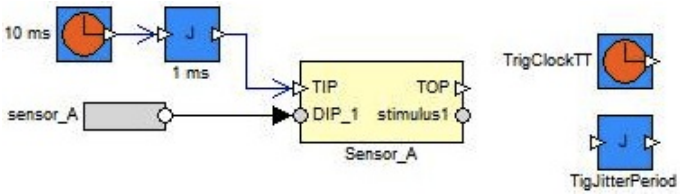

Figure 17. Equivalent to Periodic constraint specified in RCM

I. Pattern Constraint

1) TADL2 Description: It constrains the occurrences of an event that follow a certain pattern with respect to some periodic temporal points.

2) Semantics: A system behavior satisfies the specified PatternConstraint PC iff there is a set of times X such that the same system behavior concurrently satisfies the following conditions:

1) PeriodicConstraint with a period equal to PC.period
2) For each PC.offset index i: for every occurrence x of X, there is an occurrence t of the PC.event such that

$$\text{PC.offset}_i \leq (\text{t- x}) \leq (\text{PC.offset}_i + \text{PC.jitter})$$

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

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

This constraint is graphically illustrated in Figure 18. In each period of event patterns, the event occurrences happen at 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.

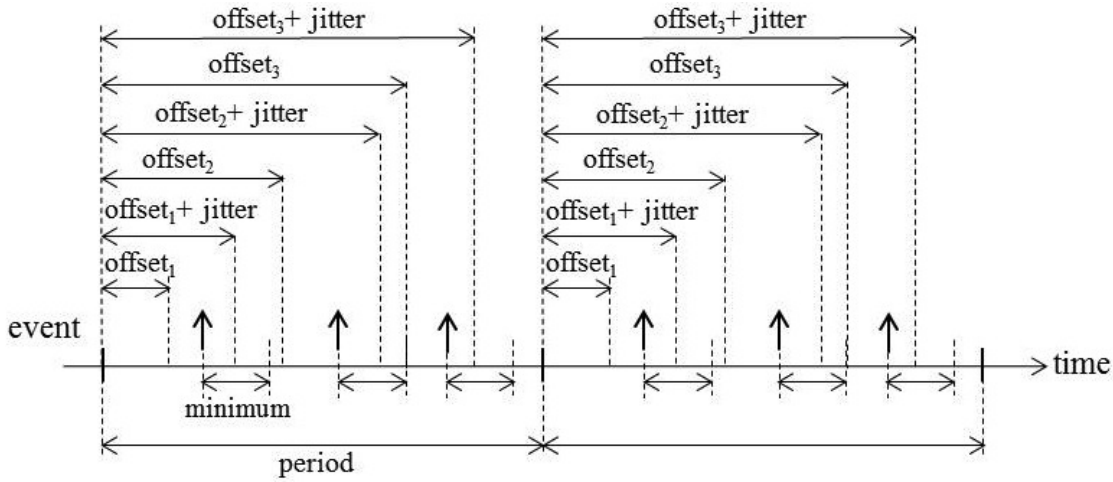

Figure 18. Graphical illustration of Pattern constraint

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 transaction with a period while the tasks 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-mentioned semantics, denoted by Pattern constraint, in RCM as shown in Figure 19. The parameters associated to 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 to satisfy this constraint by comparing the specified parameters with the corresponding parameters in the transactional model.

Figure 19. Proposed object in RCM to specify Pattern constraint

J. Arbitrary Constraint

1) TADL2 Description: It constrains an event that occurs irregularly. It contains a set of pairs consisting of minimum inter-arrival time (denoted by min) and maximum inter-arrival time (denoted by max).

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

$$\text{AC.min}_i \leq \text{d} \leq \text{AC.max}_i$$

The constraint is graphically illustrated in Figure 20. In the figure, min1, min2 and min3 represent minimum inter-arrival time between/among two, three and four subsequent occurrences of the event respectively. Similarly, max1, max2 and max3 represent 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 first two occurrences of the event, these parameters continue in a similar fashion for the rest of occurrences of the event.

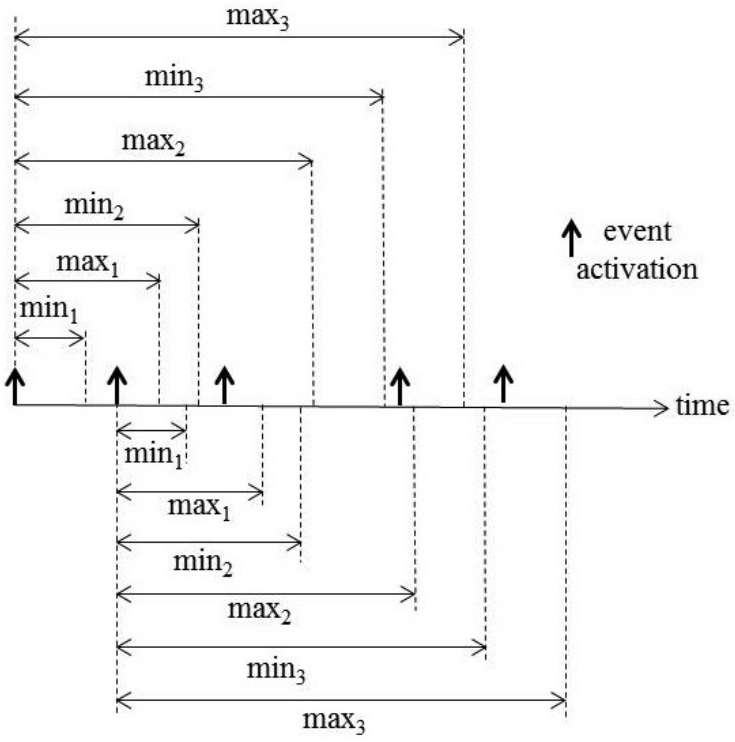

Figure 20. Graphical illustration of Arbitrary constraint

3) Interpretation in RCM: There is no existing support to specify the arbitrary constraint in RCM.

We propose the addition of a new timing constraint with the above-mentioned semantics, denoted by Arbitrary constraint, in RCM as shown in Figure 21. It is able to specify any number of pairs of min and max values.

Figure 21. Proposed object in RCM to specify Arbitrary constraint

K. Execution Time Constraint

1) TADL2 Description: It 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.

2) Semantics: A system behavior satisfies the specified ExecutionTimeConstraint ETC iff for each occurrence x of 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

$$\text{ETC.lower} \leq \text{sum of all continuous intervals in } ET_i \leq \text{ETC.upper}$$

This constraint is graphically illustrated in Figure 22.

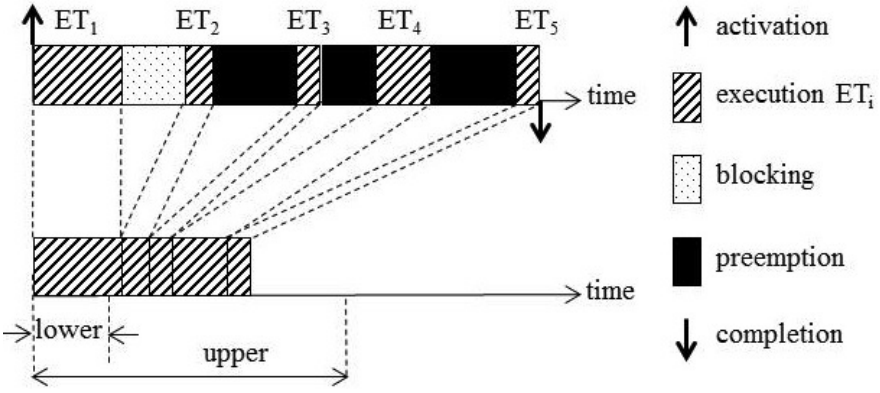

Figure 22. Graphical illustration of Execution Time constraint

3) Interpretation in RCM: There is an equivalent existing support in RCM to specify 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 newly calculated data is 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 Figure 23. In order to unambiguously interpret this constraint in RCM, the lower and upper values of this constraint (see Figure 22) can be assigned to the BCET and WCET parameters respectively in Figure 23.

L. Synchronization Constraint

1) TADL2 Description: It constrains the closeness of the occurrences of a group of events.

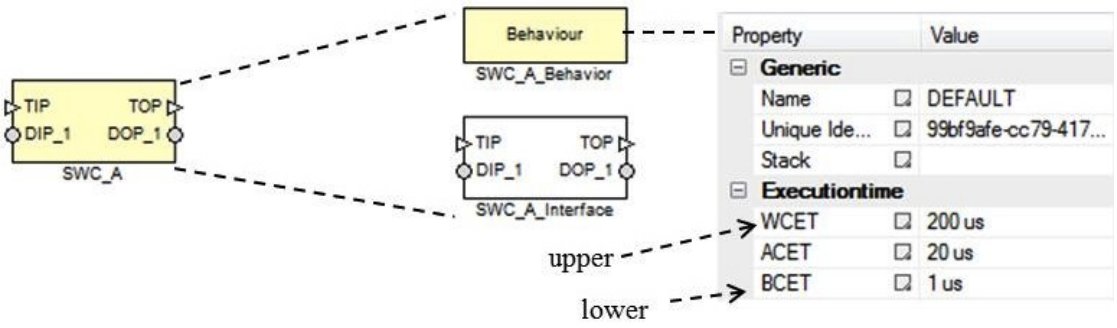

Figure 23. Equivalent to Execution Time constraint specified in RCM

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 Figure 24. 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 share occurrences of the events.

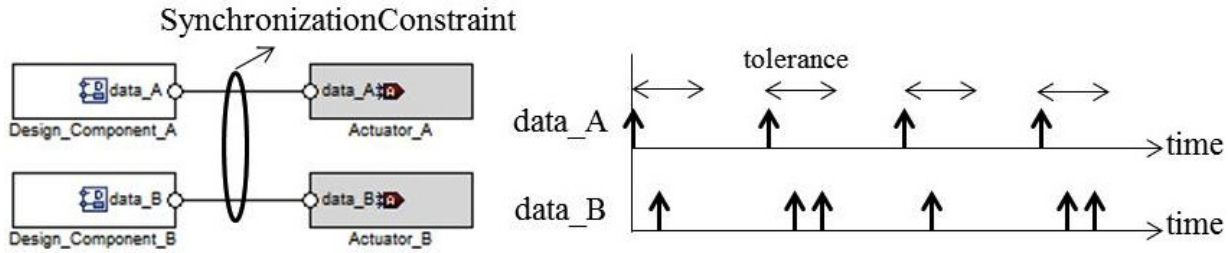

Figure 24. Graphical illustration of Synchronization constraint

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 Figure 25. 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 AND condition, the TOP is triggered only when trigger signals have arrived at all TIPs. Whereas, in the case of OR condition, the TOP is triggered as soon as there is a trigger signal at one of the TIPs. In order to make it consistent with the TADL2 Synchronization constraint, we add the tolerance parameter to this object. The analysis engine must ensure that this constraint is satisfied within the tolerance window.

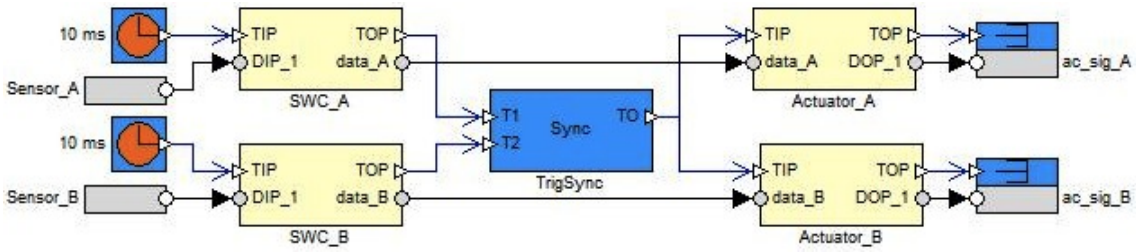

Figure 25. Synchronization constraint in RCM

M. Strong Synchronization Constraint

1) TADL2 Description: It constrains the closeness of the occurrences of a group of events.

2) Semantics: The semantics of this constraint differ from the SynchronizationConstraint in a way that the occurrences of the events in a window must have same indices. Therefore, more than one instance of the events cannot exist in a time window. Moreover, the windows cannot overlap and share occurrences of the events.

This constraint is graphically illustrated in Figure 26. It is applied on the two events data_A and data_B.

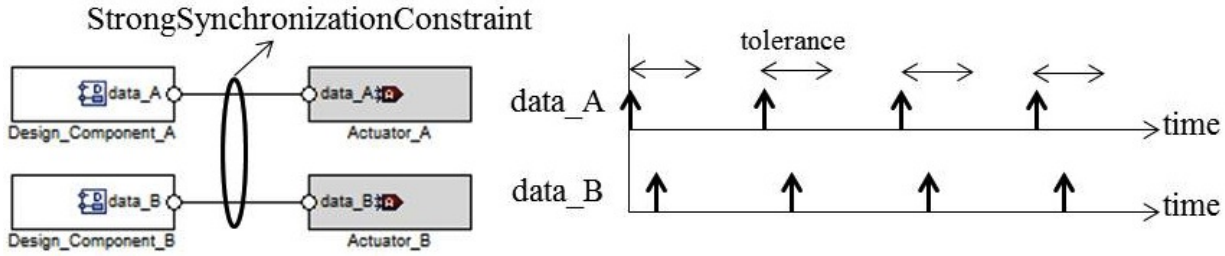

Figure 26. Graphical illustration of Strong Synchronization constraint

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 propose to add a similar object denoted by S-TrigSync as shown in Figure 27. 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 it consistent with the TADL2 Strong Synchronization constraint, we add the tolerance parameter to this object.

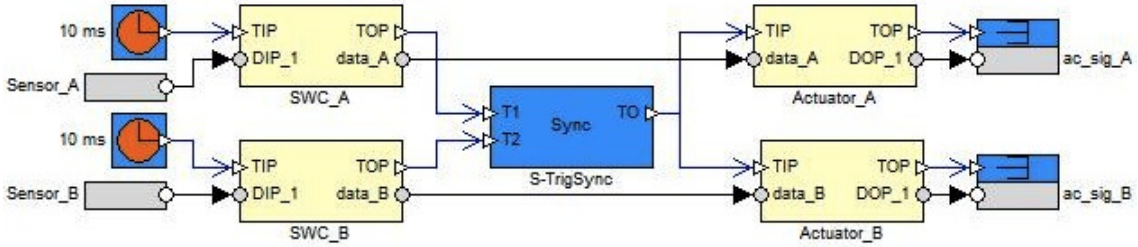

Figure 27. Proposed object in RCM to specify Strong Synchronization constraint

N. Output Synchronization Constraint

1) TADL2 Description: It constrains the closeness of the occurrences of responses to a certain stimulus. Basically, it 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 Figure 28 and Figure 29. The tolerance parameter constrains the latest of these response occurrences for each chain. The system in Figure 28 is modeled with two event chains. They have common stimulus but different responses denoted by response1 and response2.

2) Semantics: A system behavior satisfies the specified OutputSynchronizationConstraint OSC iff for each occurrence s in OSC.stimulus, there is a time t such that for each index i, there is an occurrence r in OSC.response$_i$ such that

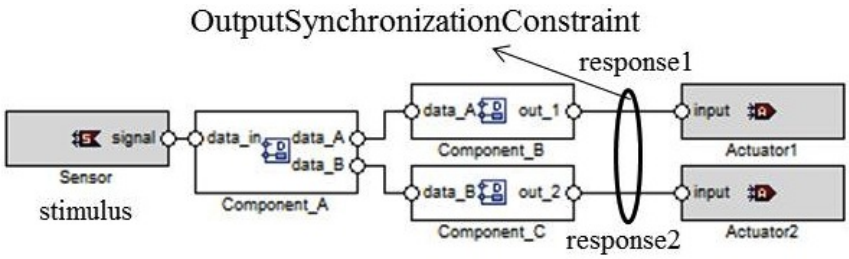

Figure 28. Usage of Output synchronization constraint at the design level

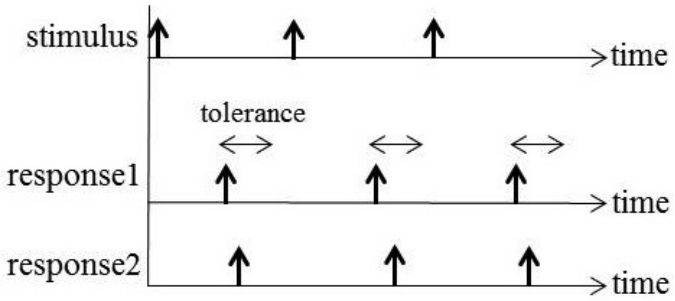

Figure 29. Graphical illustration of Output Synchronization constraint

$$\text{(r.color = s.color)}$$
$$\text{and}$$
$$\text{(r is minimal in OSC.response}_i \text{ with that color)}$$
$$\text{and}$$
$$(0 \leq \text{(r- t)} \leq \text{OSC.tolerance})$$

3) Interpretation in RCM: There is an existing support in RCM to synchronize multiple triggers by using TrigSync object. We propose to 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 it consistent with the TADL2 Output Synchronization constraint, we add the tolerance parameter to it. The analysis engine must ensure that this constraint is satisfied within the tolerance window. The example in Figure 30 depicts a single rate system; hence there cannot be more than one occurrences of each response corresponding to single occurrence of the stimulus. However, Out-TrigSync is equally applicable to multi-rate systems where the components are triggered with independent clocks.

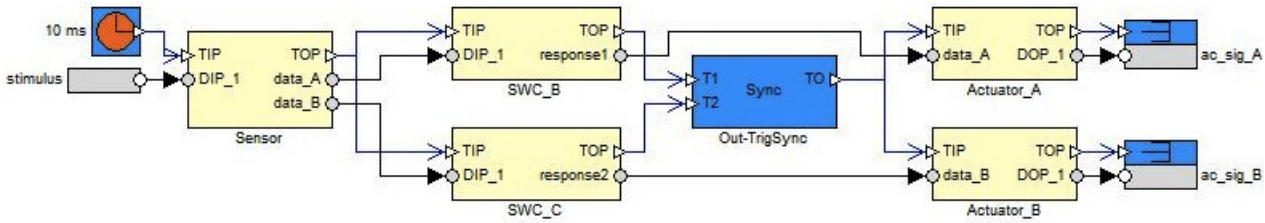

Figure 30. Proposed object to specify Output Synchronization constraint in RCM

O. Input Synchronization Constraint

1) TADL2 Description: It constrains the closeness of the occurrences of stimuli corresponding to a certain response. Basically, it 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 Figure 31 and Figure 32. 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 a time window equal to the tolerance parameter. The system in Figure 31 is modeled with two event chains. They are initiated by separate stimuli but have one common response.

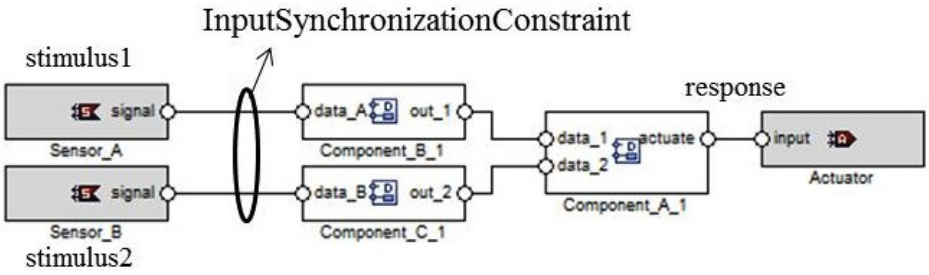

Figure 31. Usage of Input synchronization constraint at the design level

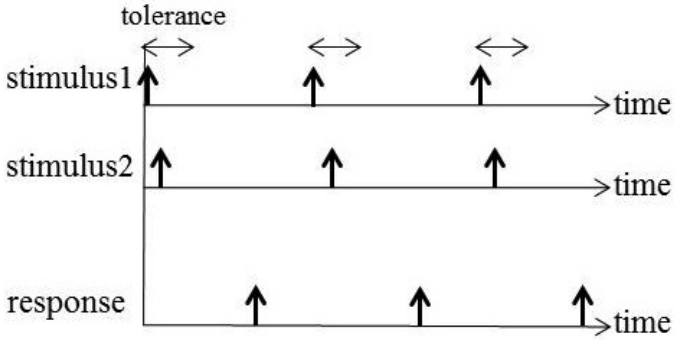

Figure 32. Graphical illustration of Input Synchronization constraint

2) Semantics: A system behavior satisfies the specified InputSynchronizationConstraint ISC iff for each occurrence r in ISC.response, there is a time t such that for each index i, there is an occurrence s in ISC.stimulus$_i$ such that

$$\begin{gathered}(\text{r.color} = \text{s.color}) \\ \text{and} \\ (\text{s is minimal in ISC.stimulus}_i \text{ with that color}) \\ \text{and} \\ (0 \leq (\text{s- t}) \leq \text{ISC.tolerance})\end{gathered}$$

3) Interpretation in RCM: There is an existing support in RCM to synchronize multiple triggers by using TrigSync object. We propose to 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 it consistent with the TADL2 Input Synchronization constraint, we add the tolerance parameter to it. The example in Figure 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, In-TrigSync is equally applicable to multi-rate systems where the components are triggered with independent clocks.

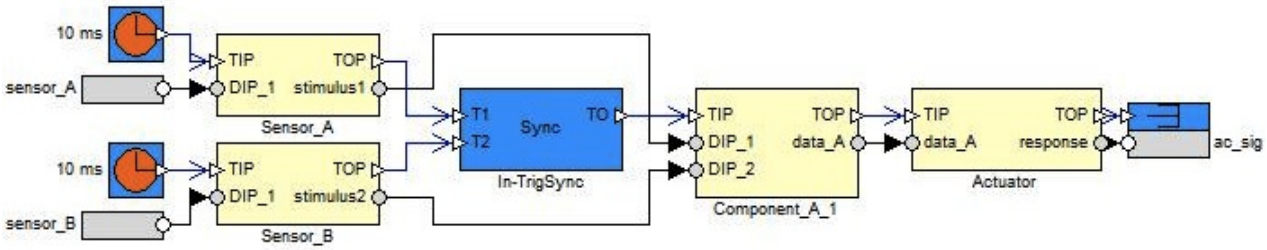

Figure 33. Proposed object to specify Input Synchronization constraint in RCM

IV. Discussion

The models and approaches that are used at the implementation level such as RCM and AUTOSAR support the extraction of 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 extraction of the timing models. Due to unavailability of the end-to-end timing model at higher abstraction levels, it may be impossible to perform the timing analysis in some cases.

We focus 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 extraction of the end-to-end timing model. We also propose guidelines and solutions to deal with these challenges. We also discuss a proposal for implementation of the solutions in RCM.

A. Extraction of Control and Data Paths

Unambiguous extraction of control (trigger) and data paths from the system are 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, trigger path of the chain shown in Figure 34(c) can be expressed as $\{\{SWC\_A \rightarrow SWC\_B\}, \{SWC\_C\}\}$ because SWC_B is triggered by SWC_A, while SWC_C is triggered independently. Similarly, trigger paths of the chains shown in Figure 34(a) and Figure 34(b) can be expressed as $\{\{SWC\_A \rightarrow SWC\_B \rightarrow SWC\_C\}$ and $\{\{SWC\_A\}, \{SWC\_B\}, \{SWC\_C\}\}$ respectively.

One of the main challenges in the extraction of the timing model at the design level is the lack of 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 Figure 35(b). 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.

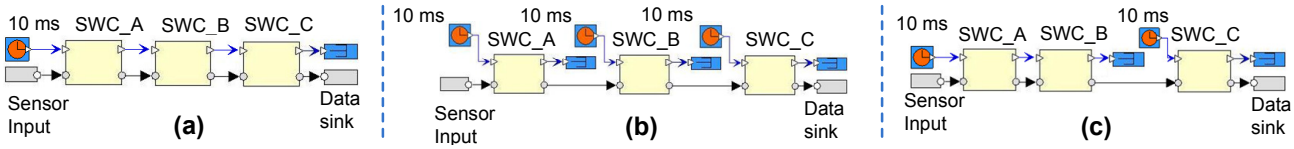

Figure 34. Example of (a) Trigger chain, (b) Data chain, and (c) Mixed chain.

On the other hand, at the design level, the components communicate via *flow ports* as shown in Figure 35(a). 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 perform end-to-end delay analysis on a trigger chain shown in Figure 35(a) [5].

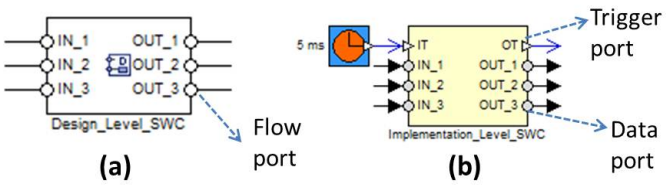

Figure 35. Model of the SWC at (a) design level, (b) implementation level

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

1) We assume a one-to-one mapping between each design- and implementation-level component. Although, a design-level component can be mapped to more than one implementation-level components; our assumption is based on common practice that is used in the industry, especially in the segment of construction-equipment vehicles domain.
2) If a timing constraint is specified on the flow port, we assume the component is triggered independently. The type of triggering is judged by the type of specified constraint.
3) If Age or Reaction constraint is specified on a chain; and no other constraint is specified, 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 either data or mixed chain. It is meaningful to specify the Age and Reaction constraints only on data and mixed 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, arrival of data at each port produces a trigger. For example, the component in Figure 35(a) may receive three individual triggers when data is separately received at 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 SWC (corresponding to the design-level component) at the implementation level. Figure 36 shows implementation-level equivalent of design-level component with three flow ports as shown in Figure 35(a).

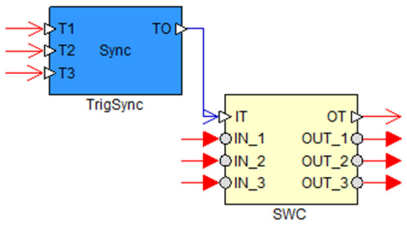

Figure 36. Implementation-level equivalent of design-level component in Figure 35(a)

B. Annotation and Extraction of Timing Parameters

The timing information expressed with the models and tools used at the design level is not enough to extract the end-to-end timing model. For example, one of the EAST-ADL based tools[8] 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, it lacks the expression of some timing parameters, e.g., priority and transmission type which are needed to perform the end-to-end timing analysis. We already discussed the interpretation of TADL2 timing constraints in RCM in the previous section.

We assume that the execution order of design-level components in a chain is specified, otherwise, we make implicit assumption about it. That is, each component is assumed to execute only after successful execution of preceding component in the chain, unless specified otherwise. This means, a data provider component is assumed to be always executed before the data receiver component. Since, this assumption fixes the execution order, it is safe to assume the priorities of the components are equal within the chain. If worst-, best- and average-case execution times are not available at the design level, they can be estimated at the implementation level either using estimations by the experts or reusing from other projects.

C. Identification of Chain Types

Since, control and data flows are clearly separated at the implementation levels, e.g., in RCM, the chain types can be easily identified as shown in Figure 34. Due to no clear separation between these flows at the design level, virtually it is not possible to identify the type of a chain. At the design level, a chain can be interpreted as a trigger or data chain. Without any explicit trigger information, the end-to-end timing analysis cannot be performed. This is because trigger chains are analyzed using end-to-end response-time analysis, whereas, data and mixed chains are analyzed using both end-to-end response-time and delay analyses [5]. 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.

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

E. Conclusion

There can be two different approaches to deal with these challenges. The first approach is to extend and improve the design-level models, languages and tools in such a way that the timing models can be completely and unambiguously extracted. Moreover, the extracted models are general enough to

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

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

V. Summary and Future Work

We extended our previous method to support the extraction of end-to-end timing models at a higher level of abstraction. The purpose is to support the end-to-end timing analysis at a higher abstraction level and early parts of the development process for component-based vehicular distributed embedded systems. At the higher level, the method extracts timing information from models of the systems that are developed with EAST-ADL and TADL2 languages using TIMMO methodology. Whereas, at the lower level, it considers Rubus Component Model (RCM) to extract the timing information that cannot be clearly specified at the higher level. As part of this method, we provided an interpretation of TADL2 timing constraints in RCM. We also proposed extensions in RCM for unambiguous transformation of these constraints. Moreover, we discussed the challenges and issues that are faced during the extraction of end-to-end timing information at a higher abstraction level. Further, we presented guidelines and solutions to deal with these challenges. These challenges and corresponding solutions may be equally applicable for other modeling technologies suitable for these abstraction levels.

In the future we plan to conduct an industrial case study to provide a proof of concept for the end-to-end timing model extraction method at various abstraction levels. 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 crank shaft. Furthermore, time can also be expressed as algebraic expressions and parameterized expressions between different time bases using the Symbolic Timing Expression [15]. It can be an interesting future work to provide support for timing expressions based on these multiple time bases in RCM.

Acknowledgement

The work in this paper is supported by the Swedish Research Council (VR) within the project SynthSoft. We thank our industrial partners Arcticus Systems and Volvo, Sweden.

References

[1] Peter Thorngren, keynote Talk: Experiences from EAST-ADL Use, EAST-ADL Open Workshop, Gothenberg, Oct., 2013.

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

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

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

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

[6] "AUTOSAR Techincal Overview, Release 4.1, Rev. 2, Ver. 1.1.0., The AUTOSAR Consortium, Oct., 2013," http://autosar.org.

[7] S. Sentilles, A. Vulgarakis, T. Bures, J. Carlson, and I. Crnkovic, "A Component Model for Control-Intensive Distributed Embedded Systems," in *11th International Symposium on Component Based Software Engineering*. Springer, Oct. 2008, pp. 310–317.

[8] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Communications-Oriented Development of Component- Based Vehicular Distributed Real-Time Embedded Systems," *Journal of Systems Architecture*, 2013.

[9] X. Ke, K. Sierszecki, and C. Angelov, "COMDES-II: A Component-Based Framework for Generative Development of Distributed Real-Time Control Systems," in *Embedded and Real-Time Computing Systems and Applications, RTCSA 2007. 13th IEEE International Conference on*, Aug. 2007, pp. 199 –208.

[10] "Catalog of Specialized CORBA Specifications. OMG Group," http://www.omg.org/technology/documents/.

[11] "TIMMO Methodology, Ver. 2," *TIMMO (TIMing MOdel), Deliverable 7*, Oct. 2009, The TIMMO Consortium.

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

[13] CRYSTAL - CRitical sYSTem engineering AcceLeration, http://www.crystal-artemis.eu, accessed May, 2014.

[14] "OMG Systems Modeling Language, version 1.3. http://www.omgsysml.org."

[15] Timing Augmented Description Language (TADL2) syntax, semantics, metamodel Ver. 2, Deliverable 11, Aug. 2012.

[16] K. Hänninen et.al., "The Rubus Component Model for Resource Constrained Real-Time Systems," in *3rd IEEE International Symposium on Industrial Embedded Systems*, Jun. 2008.

[17] "EAST-ADL Domain Model Specification, Deliverable D4.1.1," http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-06-02.pdf.

[18] TIMMO-2-USE Methodology Description, Ver. 2, Del. 13, Jul., 2012.

[19] "Rubus ICE-Integrated Development Environment," http://www.arcticus-systems.com.

[20] P. Feiler, B. Lewis, S. Vestal, and E. Colbert, "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*, ser. The International Federation for Information Processing (IFIP). Springer US, 2005, vol. 176, pp. 3–15.

[21] SCADE Suite, http://www.esterel-technologies.com/products/scade-suite, accessed May, 2014.

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

[23] "MAST–Modeling and Analysis Suite for Real-Time Applications," http://mast.unican.es.

[24] CHESS Project, CHESS consortium. Avialable at: http://www.chess-project.org, accessed May, 2014.

[25] A. Cicchetti, F. Ciccozzi, S. Mazzini, S. Puri, M. Panunzio, T. Vardanega, and A. Zovi, "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.

[26] TADL: Timing Augmented Description Language, Ver. 2, Deliverable 6, Oct., 2009.

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

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

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

[30] Mastering Timing Information for Advanced Automotive Systems Engineering. In the TIMMO-2-USE Brochure, 2012. Available at: http://www.timmo-2-use.org/pdf/T2UBrochure.pdf.

[31] J. Carlson, "Timing Analysis of Component-based Embedded Systems," in *15th International ACM SIGSOFT Symposium on Component Based Software Engineering*. ACM, Jun. 2012.

[32] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "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, Oct. 2011, pp. 1–6.