Software Qual J (2016) 24:835–876
DOI 10.1007/s11219-015-9290-6

Timing consistency checking for UML/MARTE behavioral models

Jinho Choi[1] · **Eunkyoung Jee**[2] · **Doo-Hwan Bae**[2]

Published online: 10 September 2015

Abstract UML/MARTE model-driven development approaches are gaining attention in developing real-time embedded software (RTES). UML behavioral models with MARTE annotations are used to describe timing behaviors and timing characteristics of RTES. Particularly, state machine, sequence, and timing diagrams with MARTE annotations are appropriate to understand and analyze timing behaviors of RTES. However, to guarantee software correctness and safety, timing inconsistencies in UML/MARTE should be identified in the design phase of RTES. UML/MARTE timing inconsistencies are related to modeling errors and can be hazards throughout the lifecycle of RTES. We propose a systematic approach to check timing consistency of state machine, sequence, and timing diagrams with MARTE annotations for RTES. First, we present how state machine, sequence, and timing diagrams with MARTE annotations specify the behaviors of RTES. To overcome informal semantics of UML/MARTE models, we provide formal definitions of state machine, sequence, and timing diagrams with MARTE annotations. Second, we present the timing consistency checking approach that consists of a rule-based and a model checking-based timing consistency checking. In the rule-based timing consistency checking, we validate well formedness of UML/MARTE behavioral models in timing aspects. In the model checking-based timing consistency checking, we verify whether timing behaviors of sequence and timing diagrams with MARTE annotations are consistent with the timing behaviors of state machine diagrams with MARTE annotations. We

✉ Jinho Choi
jhchoi93@gmail.com

Eunkyoung Jee
ekjee@se.kaist.ac.kr

Doo-Hwan Bae
bae@se.kaist.ac.kr

[1] The 1st R&D Institute, Agency for Defense Development (ADD), Yuseong P.O. Box 35, Daejeon 34188, Republic of Korea

[2] School of Computing, KAIST, 291 Daehak-ro, Yuseong-gu, Daejeon 34141, Republic of Korea

support an automated timing consistency checking tool UML/MARTE timing Consistency Analyzer for a seamless approach. We demonstrate the effectiveness and the practicality of the proposed approach by two case studies using cruise control system software and guidance and control unit software.

Keywords UML · State machine diagram · Sequence diagram · Timing diagram · MARTE · Timing consistency checking

1 Introduction

Real-time embedded software (RTES) for safety-critical systems (e.g., avionics, military, automobile, and medical equipment, etc.) are becoming larger and more complex (Millett et al. 2007; Leveson 2011). Timing is still a major concern in the development of RTES (Peraldi-Frati et al. 2012; Leveson 2011; Lavagno et al. 2003; Gomes et al. 2010). Thus, timing behavioral modeling for RTES is indispensable to manage software complexity and safety. Timing behavioral modeling has advantages as follows: First, through timing behavioral modeling, RTES designers can clearly capture required functionalities and timing constraints of RTES (Lavagno et al. 2003; Gomes et al. 2010; Sgroi et al. 2000). Second, timing behavioral modeling provides a common understanding on software behaviors to diverse stakeholders.

The Unified Modeling Language (UML), as a de facto modeling language in practice, provides a set of graphical notations to create the visual models of software structures and behaviors (OMG 2011b). Since UML provides models with multiple viewpoints, UML facilitates communication among stakeholders. Among UML behavioral models, state machine, sequence, and timing diagrams are appropriate to understand and analyze RTES behaviors. A state machine diagram describes the overall behaviors of a single object. A sequence diagram shows the flow of logic by exchanging messages among lifelines (i.e., objects). A timing diagram represents state behaviors of objects focusing on timing constraints. In UML modeling, state machine diagrams and sequence diagrams have been used usually to specify the behaviors of software. Timing diagrams are very useful for effective communication among different stakeholders in the development of RTES because timing diagrams provide intuitive specifications for timing constraints and have been used widely in electronic domain (Fowler 2004; Choi et al. 2012b). However, UML 1.x or 2.0 notations are hard to fully describe properties of RTES such as timing and resource usage. To overcome such limitation of UML, the Object Modeling Group (OMG) has adopted the profile for Modeling and Analysis of real-time embedded systems, in short MARTE (OMG 2011a). With stereotypes and tagged values predefined for specification and design, the MARTE profile provides licit interpretation for UML models to stakeholders. Therefore, SMDs/MARTE (i.e., state machine diagrams with MARTE annotations), SDs/MARTE (i.e., sequence diagrams with MARTE annotations), and TDs/MARTE (i.e., timing diagrams with MARTE annotations) can be used to describe the behaviors and timing characteristics of RTES.

Due to different UML behavioral models with MARTE annotations, timing inconsistencies can exist naturally in a UML model with MARTE annotations and between different types of UML/MARTE models. Timing inconsistency problems in SMDs/MARTE, SDs/MARTE, and TDs/MARTE are as follows: First, timing inconsistencies are related to

modeling errors and can be propagated to the code implementation. If timing inconsistencies are not detected in testing, then it can lead to catastrophic system failures. Suppose that the same timing behavior is specified in an SD/MARTE and a TD/MARTE model, but these two models are not consistent due to human errors in modeling. Without timing consistency checking, if functions of RTES are implemented from a timing scenario in the SD/MARTE model and a timing scenario in the TD/MARTE model is used in testing, then the inconsistencies can be detected in testing. As a result, the timing inconsistencies will lead to a development cost increase because late checking is required to identify the inconsistency points between the SD/MARTE and the TD/MARTE. Second, it is hard to detect timing inconsistencies by human review. In particular, it is hard to check whether the timing scenarios of SDs/MARTE and TDs/MARTE are consistent with the timing behaviors of SMDs/MARTE through human review. Thus, an automated tool is needed to effectively detect timing inconsistency points in UML/MARTE models. Last, most UML tools only provide a standard representation of UML models. To specify UML/MARTE models in the RTES development, a systematic timing consistency checking approach should be provided to RTES designers.

Our research questions regarding the specification of SMDs/MARTE, SDs/MARTE, and TDs/MARTE models for RTES are as follows:

- What types of inconsistencies are in SMDs/MARTE, SDs/MARTE, and TDs/MARTE models for RTES?
- How can we detect timing inconsistencies in SMDs/MARTE, SDs/MARTE, and TDs/MARTE models for RTES?
- Is our approach useful and practical in the RTES domain?

To tackle the research questions, we propose a systematic approach to check timing consistency of SMDs/MARTE, SDs/MARTE, and TDs/MARTE models for RTES. The proposed approach consists of a rule-based timing consistency checking and a model checking-based timing consistency checking. The rule-based timing consistency checking checks intra-model and inter-model consistency of UML/MARTE models. The model checking-based timing consistency checking checks whether timing scenarios of SDs/MARTE and TDs/MARTE are consistent with the behaviors of SMDs/MARTE. To this end, timed automata models are transformed from SMDs/MARTE and timing properties to be verified are extracted from SDs/MARTE and TDs/MARTE. We present how UML/MARTE models specify the behaviors of RTES and provide formal definitions for SMDs/MARTE, SDs/MARTE, and TDs/MARTE models. We categorize UML/MARTE consistency types into timing and non-timing consistency, and intra-model and inter-model consistency. We support an automated tool, UMCA (UML/MARTE timing Consistency Analyzer), for a seamless timing consistency checking approach (Choi 2015). Using the UMCA, we demonstrate the proposed approach with two case studies, cruise control system (CCS) software in automotive systems and guidance and control unit (GCU) software in avionics systems.

The contributions of the proposed approach are as follows:

- We establish a basis for timing consistency checking for UML/MARTE models by categorizing consistency types to be checked.
- We detect timing inconsistencies in SMDs/MARTE, SDs/MARTE, and TDs/MARTE with the automated tool.

The remainder of this paper is organized as follows: Sect. 2 explains background knowledge about UML, MARTE, timed automata, and model checking. Section 3 discusses the related works. Section 4 presents how SMDs/MARTE, SDs/MARTE, and

TDs/MARTE specify the behaviors of RTES. Section 5 describes the proposed approach to check timing consistency for SMDs/MARTE, SDs/MARTE, and TDs/MARTE models. Section 6 presents the design and implementation of an automated timing consistency checking tool, and describes the correctness validation of rules and model transformation using the tool. Section 7 describes the results of a CCS software case study. Section 8 describes the results of a GCU software case study. Section 9 discusses the proposed approach. Finally, Section 10 provides the conclusion and future work.

2 Background

2.1 UML

UML is an open standard which is controlled by OMG. UML is a general-purpose modeling language for the visualization and understanding of software structures and behaviors. Moreover, UML helps communicate among stakeholders and supports many structural and behavioral diagrams with diverse viewpoints. At present, UML 2.4.1 is released formally. It consists of two parts: superstructure and infrastructure. The superstructure defines the notation and semantics for UML diagrams and their model elements, while the infrastructure defines the core metamodel used by the superstructure. Figure 1 shows the taxonomy of UML structure and behavior diagrams (OMG 2011b). Fourteen UML diagrams are defined in UML 2.4.1. They are classified into structure diagrams, behavior diagrams, and interaction diagrams as shown in Fig. 1. Structure diagrams represent the static structures of objects. Behavior diagrams show the dynamic behaviors of objects. Interaction diagrams represent the flow of control and data among objects. In this paper, we use state machine diagrams, sequence diagrams, and timing diagrams to specify the behaviors of RTES. State machine diagrams are behavior diagrams. Sequence diagrams and timing diagrams are interaction diagrams.

2.2 MARTE

MARTE is a profile for strengthening the expressive power of UML. It supports modeling and analysis in real-time embedded systems. At present, MARTE 1.1 is released formally. MARTE provides predefined stereotypes and tagged values to specify non-functional

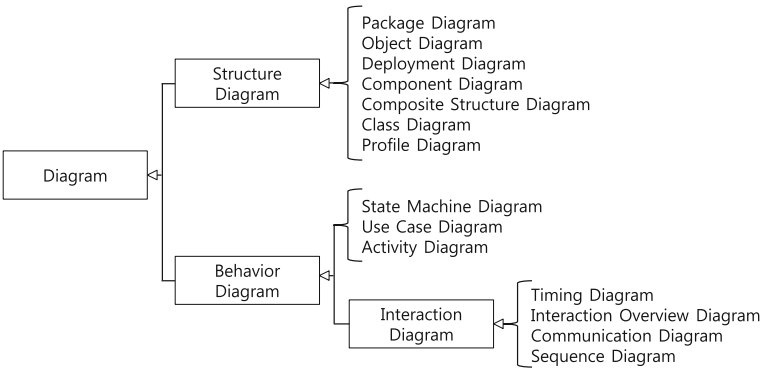


Fig. 1 The taxonomy of UML structure and behavior diagrams (OMG 2011b)

Table 1 Example of stereotypes and tagged values in the MARTE profile

Characteristics	Stereotypes	Tagged values
Task/thread	<<swScheduableResource>>	isPreemptable
		deadlineElements
Timing analysis	<<SaStep>>	Deadline
		execTime
Hardware	<<InterruptResource>>	durationElements
		vectorElements

properties, timing, general resources, and allocation. MARTE is organized into four main packages: MARTE foundations, MARTE design model, MARTE analysis model, and MARTE annexes. MARTE foundations define concepts for real-time and embedded systems. MARTE design model provides concepts required from specification to detailed design of real-time embedded systems. MARTE analysis model offers analysis modeling concepts such as schedulability and performance analysis. MARTE annexes provide a predefined MARTE model library and value specification languages such as the Value Specification Language (VSL), the Clocked Valued Specification Language (CVSL), and the Clock Constraint Specification Language (CCSL). Table 1 shows an example of stereotypes and tagged values defined in the MARTE profile. In this paper, we follow the MARTE profile to specify UML/MARTE models. To specify timing constraints information such as a deadline and an execution time, <<SaStep>> stereotype is used in MARTE annotation.

2.3 Timed automata

Timed automata were proposed by Alur and Dill (1994). Timed automata are used to specify and analyze timeliness properties of real-time systems. It is specified with two fundamental elements, a finite automaton and clocks (Bérard et al. 2010). A finite automaton describes locations (i.e., states or nodes) and transitions between locations. Clocks are used to model the quantitative timing constraints. Figure 2 shows an example of a timed automata model. As shown in Fig. 2, a transition is composed of guard, synchronization, and assignment. Clock values or flag values are used in guards. Receiving and sending messages are specified in synchronization. A message with ? represents a receiving message and a message with ! represents a sending message. Assignment is used to set clock values or flag values. An invariant represents the amount of time that may be spent in a location. In Fig. 2, a, current location is StateA as long as clock x is less than or equal to 5. If clock x is greater than or equal to 5, then the current location is changed to StateB by sending event message and setting clock to 0. A location transition is occurred when clock x is equal to 5. If the invariant of StateA specifies that clock x is less than 5, there is no location transition when clock is equal to 5. Fersman et al. (2002) proposed timed automata with tasks. A task is an executable program triggered by events and is specified in a location of timed automata. It has two parameters such as a worst-case execution time and a deadline. Timed automata with tasks can be used to check schedulability in the design phase of RTES (Choi et al. 2011a, b).

2.4 Model checking

Model checking verifies automatically whether the specified models satisfy desired properties (Clarke et al. 1999; Bérard et al. 2010). Figure 3 shows a model checking process.

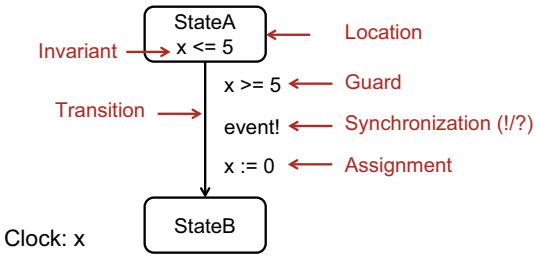


Fig. 2 Example of timed automata

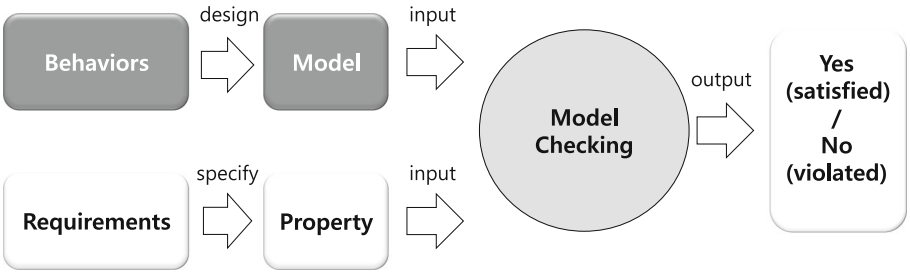


Fig. 3 Model checking process

Software behaviors are represented as semantic models using a formal language like timed automata. On the other hand, software requirements are specified as properties that are composed of temporal logic formula. Models and properties are used as inputs to a model checker (e.g., UPPAAL or TIMES). A model checker enumerates and explores all reachable states of the given model to check whether all reachable states in the models satisfy the given property. When the specified models satisfy the property, the model checker announces the result as YES. Otherwise, the model checker reports the result as NO. It means that the models violate the property. In addition, a counterexample is provided to show the reasons and locations. A counterexample shows the trace from the initial state to the violating state. In this paper, model checking is used to check whether timing behaviors of SDs/MARTE and TDs/MARTE are consistent with timing behaviors of SMDs/MARTE.

3 Related work

Many studies have been proposed to tackle UML inconsistency problems. Lucas et al. (2009) and Usman et al. (2008) analyzed existing studies on UML consistency. From their works, we found that most UML consistency studies focused on state machine diagrams and sequence diagrams. To the best of our knowledge, the proposed approach is the first study to identify inconsistencies in timing diagrams. We have classified the existing works into three categories: UML consistency concepts, UML consistency checking using a rule-based method, and UML consistency checking using a model checking technique.

3.1 UML consistency concepts

Küster et al. (2001) provided consistency concepts such as syntactical, semantic, and temporal consistency in UML-RT sequence diagrams and state machine diagrams for real-

time embedded systems. UML-RT (Selic 1998) is an extension of UML to use the notion of capsules that describe concurrent and active objects. Capsules have each their own thread of execution and communicate with other capsules via interfaces called ports. In their approach, consistency concepts are defined as follows: A sequence diagram and a state machine diagram are syntactically consistent if messages of the sequence diagram are a subset of messages in the state machine diagram. A sequence diagram and state machine diagrams are semantically consistent if a sending message corresponds to a sending action in a state machine diagram and a receiving message corresponds to an event in the state machine diagram. A sequence diagram and a state machine diagrams are temporally consistent if tasks can be allocated and scheduled. They also proposed a method for checking temporal consistency that consists of the following steps: task graph building, tasks allocation, a scheduling analysis, and the sequence diagrams rescaling. However, their work for checking temporal consistency has two limitations. First, additional time information [i.e., worst-case execution times and communication costs) is needed to generate task graphs because sufficient information for temporal consistency checking are not specified in sequence and state machine diagrams. Second, their approach cannot detect wrong temporal scenarios of sequence diagrams because only sequence diagrams have timing information. Thus, wrong task graphs can be generated from incorrect sequence diagrams.Gherbi and Khendek (2007)] focused on consistencies in UML state machine diagrams and sequence diagrams that are extended with SPT (schedulability, performance, and time specification) profile. SPT (OMG 2005) is an OMG standard UML profile for the modeling and analysis of real-time systems. It is superseded by MARTE profile. In their approach, consistencies are classified into syntactic, behavioral, concurrency-related, and time consistency. Syntactic consistency checks the well formedness of each diagram. Behavioral, concurrency-related, and time consistency are semantic consistency. Behavioral consistency defines that behaviors of sequence diagrams should be consistent with behaviors of state machine diagrams. Concurrency-related consistency defines that concurrency in UML/SPT models should be satisfied. Time concurrency defines that state machine diagrams should be consistent with a set of sequence diagrams in scheduling aspects. They provided a method to check time consistency using a task model. However, making a task model is not easy, since the task model is transformed from a UML/SPT-based schedulability model. The UML/SPT-based schedulability model should be generated from sequence diagrams and state machine diagrams.

3.2 UML consistency checking using a rule-based method

Egyed (2006, 2007, 2011) proposed an approach to detect and track inconsistencies quickly. Class, state machine, and sequence diagrams were considered to check consistency. In this study, the previous evaluation information was used to avoid the unnecessary reevaluations when UML models change. UML/Analyzer tool was developed to support the proposed approach. Nentwich et al. (2002) proposed xlinkit framework to check the consistency of distributed Web content. Also, the rule language was presented to express consistency constraints between documents. Xlinkit can check UML models because xlinkit evaluates XML-based documents. In addition, xlinkit can check consistency of changed documents like Egyed's work. Gogolla et al. (2007) proposed USE (UML-based Specification Environment) tool. USE tool validates consistency for class diagrams using Object Constraint Language (OCL) (OMG 2006). OCL specifies invariants, preconditions, and postconditions for attributes and operations of class diagrams. USE checks whether attributes and operations of class diagrams conform to object diagrams and sequence

diagrams. Chiorean et al. (2004) suggested OCLE (Object Constraint Language Environment) tool. The OCLE is similar to the USE tool in that they also use OCL to check UML consistency. However, OCL cannot express some rules due to insufficient semantics of OCL (Sourrouille and Caplat 2002). Above four studies can check syntactic well formedness of UML models effectively, but their approaches cannot detect timing inconsistencies in UML models.

3.3 UML consistency checking using a model checking technique

Zhao et al. (2006) proposed an approach to check consistency between UML state machine diagrams and sequence diagrams. For this purpose, they used a SPIN model checker (Holzmann 2003). To verify whether a sequence diagram is consistent with state machine diagrams, Promela models are transformed from state machine diagrams and a sequence diagram. Promela is a formal language for modeling finite-state concurrent processes and is an input language of SPIN (Holzmann 2003). Since their work focuses that the invocation between lifelines in a sequence diagram should conform to the transitions of the corresponding state machine diagrams, only event sequences of sequence diagrams can be checked in this approach. Thus, their approach cannot check consistency for timing behaviors of RTES. Engels et al. (2001) presented a general methodology to check consistency for UML-RT state machine diagrams. UML-RT state machine diagrams were translated into CSP models. They focused on semantic consistency for the protocol between state machine diagrams. However, they did not provide guidelines for the transformation of state machine diagrams into CSP models. Thus, inconsistencies between state machine diagrams and CSP models can exist. Laleau et. al. (2008) provided an approach for checking consistency of class diagrams, collaboration diagrams, and state machine diagrams for information systems. They defined formal UML models at the metamodel level. Using the formalization of UML metamodels, they checked the syntax of intra- and inter-models. For semantic consistency checking, UML models are translated into B model. However, they did not demonstrate the proposed approach with a case study to show the effectiveness of the approach.

4 UML/MARTE behavioral modeling for real-time embedded software

We present how an SMD/MARTE, an SD/MARTE, and a TD/MARTE specify the behaviors of RTES. Due to the informal semantics of UML/MARTE models, we provide the formal definition of an SMD/MARTE, an SD/MARTE, and a TD/MARTE. Objectives of formalizing UML/MARTE models are as follows:

- To make more precise both the syntax and semantics of UML/MARTE behavioral models.
- To identify timing consistency checking points among SMDs/MARTE, SDs/MARTE, and TDs/MARTE in the rule-based timing consistency checking.
- To transform systematically from SMDs/MARTE into timed automata models in the model checking-based timing consistency checking.

In addition, the UML/MARTE modeling guidelines were established by the needs of ADD (Agency Defense Development) in the Republic of Korea. The guidelines are based on literature (OMG 2011a, b; Gomaa 2001; Rumbaugh et al. 2004; Pont 2001) and domain

experience (Choi et al. 2005, 2011a, b, 2012b). The guidelines have the following assumption.

Assumption (*Timing behaviors are performed under synchrony hypothesis.*) Message exchange time is not specified in UML/MARTE behavioral models. If a message exchange time needs to be considered, then the message exchange time can be added to an execution time of an object.

4.1 An illustrative example

Insulin Pump Control Software (IPCS) is selected to show how UML/MARTE behavioral models are specified and to illustrate how the proposed approach works. IPCS is safety-critical real-time embedded software that is operated on an automated insulin pump device. The objective of IPCS is to check a diabetic's blood sugar level and to deliver insulin at regular times. A software failure in an insulin pump device can do serious harm to a diabetic's health. An example of software requirements for IPCS (Sommerville 2011) is shown in Table 2. The first requirement R1 describes that IPCS should determine an insulin injection every 10 min. The second requirement R2 represents that current states of a diabetic and a device should be displayed every 5 s. The last requirement R3 describes that the insulin injection function has higher priority than the displaying function. Figure 4 shows a class diagram of IPCS. IPCS consists of four classes: Clock, ClockManager, Controller, and Display. Clock calls ClockManager every 100 ms (milliseconds). ClockManager calls Controller or Display according to current timing information. Controller determines an insulin injection and Display shows current states of a diabetic and a device.

4.2 A state machine diagram with MARTE annotations

An SMD/MARTE specifies the overall behaviors of an object. SMDs/MARTE specify the overall behaviors of RTES. The definition of an SMD/MARTE is as follows:

Definition 1 (*SMD/MARTE*) A state machine diagram with MARTE annotations is a structure $(N_{smd}, S_{smd}, E_{smd}, G_{smd}, A_{smd}, MA_{smd}, TR_{smd})$ where:

Table 2 Example of software requirements for IPCS

Requirements number	Contents
R1	Every 10 min, an insulin injection is determined
R2	Every 5 s, states of a diabetic and an insulin pump device are displayed
R3	An insulin injection is more important than displaying

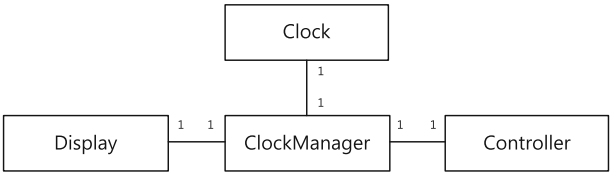


Fig. 4 Class diagram of IPCS

- N_{smd} is a name of a state machine diagram.
- S_{smd} is a set of states.
- E_{smd} is a set of events. An event is a receiving message.
- G_{smd} is a set of guards. A guard is composed of a set of time conditions and a set of flag conditions.
- A_{smd} is a set of actions. An action consists of a sending message, a set of time resets, and a set of flag resets.
- MA_{smd} is a set of MARTE annotations. A MARTE annotation is composed of a set of stereotypes, a set of tagged values, and an owner name. An owner name is N_{smd} or is included in S_{smd}.
- $TR_{smd} : S_{smd} \times E_{smd} \times G_{smd} \times A_{smd} \rightarrow S_{smd}$ is a transition relationship.

4.2.1 *Modeling guideline*

A state shows a running or blocked state of an object. A transition represents a movement between states and is composed of three optional parts: an event, a guard, and an action. An event is a receiving message to trigger a movement between states. A guard has a Boolean condition that should be satisfied for the transition. An action is executed during the transition. A sending message, time resets, and flag resets are specified in an action. MARTE annotations are composed of predefined stereotypes and tagged values. MARTE annotations linked to states specify the duration (i.e., worst-case execution times or blocking times) of each state. The MARTE annotation of SMD/MARTE represents timing constraints information (e.g., a deadline and a worst-case execution time) of the object.

Figure 5 shows an SMD/MARTE of ClockManager class that consists of five states and eight transitions. Three time variables (i.e., t1, `milliseconds_x100`, seconds) are used to check time conditions. t1 represents the clock of ClockManager and is used to check time conditions for all state transitions of ClockManager. Waiting state receives ClockMsg every 100 ms from Clock class. Also, `milliseconds_x100` is incremented every 100 ms by Clock class. TimeChecking state checks whether the sum of `milliseconds_x100` is 1 s or not. If the sum is one second then the JobSelecting state is activated. JobSelecting state moves to the Displaying state per 5 s or moves to the InsulinChecking

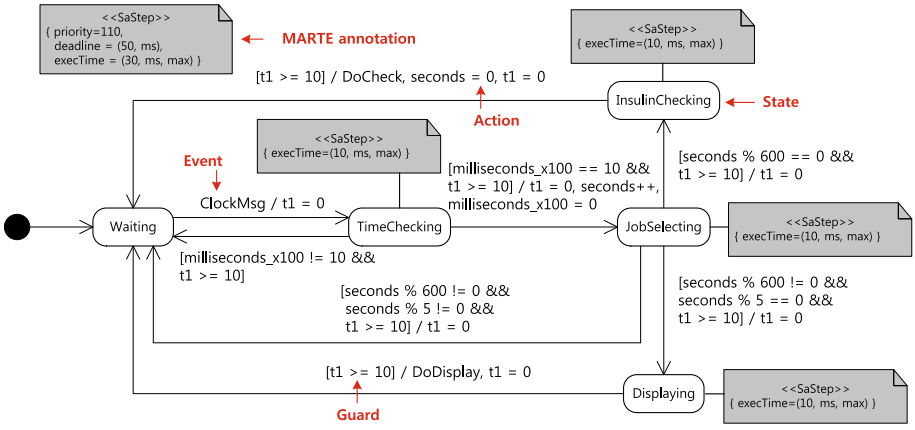


Fig. 5 SMD/MARTE for ClockManager

state per 600 s. The MARTE annotation linked to Displaying state shows that the Displaying state is executed during 10 ms in worst-case execution time. The MARTE annotation of ClockManager represents timing constraints information of ClockManager.

4.3 A sequence diagram with MARTE annotations

An SD/MARTE describes a partial behavior and interaction among objects. An SD/MARTE is defined as follows:

Definition 2 (*SD/MARTE*) A sequence diagram with MARTE annotations is a structure $(N_{sd}, L_{sd}, O_{sd}, E_{sd}, M_{sd}, T_{sd}, MA_{sd}, <_{sd}, LOC_{sd})$ where:

- N_{sd} is a name of a sequence diagram.
- L_{sd} is a set of lifelines.
- O_{sd} is a set of occurrence specifications. An occurrence specification represents a syntactic point at the starts or ends of messages or at the beginning or end of an execution specification (OMG 2011b).
- E_{sd} is a set of execution specifications. An execution specification consists of a start occurrence specification and a finish occurrence specification.
- M_{sd} is a set of messages. A message is composed of a sending occurrence specification, a receiving occurrence specification, a message name, and a message type.
- T_{sd} is a set of time observations. A time observation is composed of a time value and an occurrence specification.
- MA_{sd} is a set of MARTE annotations. A MARTE annotation consists of a set of stereotypes, a set of tagged values, and an owner name. An owner name is included in L_{sd} or E_{sd}.
- $<_{sd} \subseteq O_{sd} \times O_{sd}$ is a set of total ordering functions. The orders describe an order relationship between two adjacent occurrence specifications.
- $LOC_{sd} : O_{sd} \rightarrow L_{sd}$ is a function by which the location of an occurrence specification is defined.

4.3.1 Modeling guideline

MARTE annotations, execution specifications, and time observations are used to express timing constraints of RTES. MARTE annotation linked to a lifeline shows the timing constraints of the lifeline (i.e., object) such as deadline, worst-case execution time, and priority. MARTE annotations linked to execution specifications specify execution times of lifelines. Execution specification represents the execution of an object within the lifeline. Time observation is a reference to a time instant during an execution (OMG 2011b).

Figure 6 shows an example of an SD/MARTE which represents a scenario for displaying. White vertical rectangles in Fig. 6 represent execution specifications. Clock sends ClockMsg to ClockManager at 590,000 ms. ClockManager sends DoDisplay message to Displayer at 590,030 ms. Displayer works for 30 ms. @t = 590,000 is a time observation. The MARTE annotation linked to ClockManager shows the timing constraints of ClockManager. The MARTE annotation linked to the execution specification of ClockManager specifies that ClockManager executes during 30 ms in worst-case execution time.

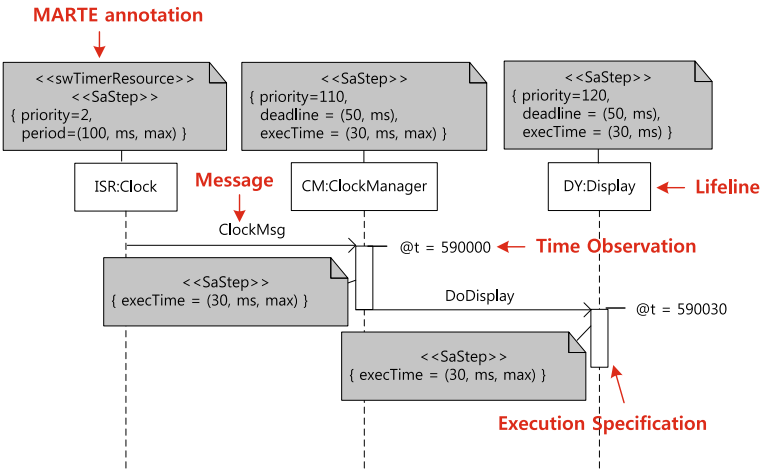


Fig. 6 SD/MARTE of IPCS

4.4 A timing diagram with MARTE annotations

A TD/MARTE specifies a partial behavior of RTES and shows state changes of objects over time. The definition of a TD/MARTE is as follows:

Definition 3 (*TD/MARTE*) A timing diagram with MARTE annotations is a structure (N_{td}, L_{td}, S_{td}, T_{td}, O_{td}, M_{td}, MA_{td}, $<_{td}$, LOC_{td}) where:

- N_{td} is a name of a timing diagram.
- L_{td} is a set of lifelines.
- S_{td} is a set of states. A state consists of a lifeline and a set of durations. Durations are time range of states.
- T_{td} is timing ruler.
- O_{td} is a set of occurrence specifications.
- M_{td} is a set of messages. A message is composed of a sending occurrence specification, a receiving occurrence specification, a message name, and a message type.
- MA_{td} is a set of MARTE annotations. MARTE annotation consists of a set of stereotypes, a set of tagged values, and an owner name. An owner name is included in L_{td}.
- $<_{td} \subseteq O_{td} \times O_{td}$ is a set of total ordering functions. The orders describe an order relationship between two adjacent occurrence specifications.
- $LOC_{td} : O_{td} \rightarrow S_{td}$ is a function by which the location of an occurrence specification is defined.

4.4.1 Modeling guideline

Lifelines, states, messages, durations, timing ruler, and MARTE annotations are used to specify a TD/MARTE. Duration shows an execution time or a blocking time of a state. Timing ruler displays passage of time from left to right on a TD/MARTE. The use of MARTE annotations is the same as the MARTE annotations in an SD/MARTE. MARTE annotations linked to lifelines represent timing constraints information of each lifeline.

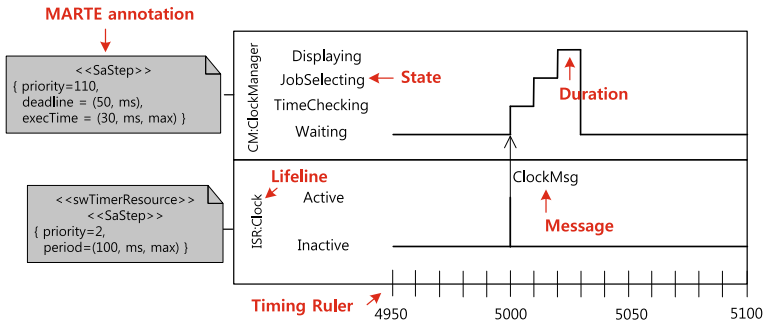


Fig. 7 TD/MARTE of IPCS

Figure 7 shows an example of a TD/MARTE which represents a timing scenario between Clock and ClockManager. Clock sends ClockMsg to ClockManager at 5000 ms (i.e., 5 s) and then ClockManager is executed with states such as TimeChecking, JobSelecting, and Displaying during 30 ms.

5 Timing consistency checking for UML/MARTE behavioral models

We propose a timing consistency checking approach to validate and verify timing specifications between a UML diagram and MARTE annotations, and among UML/MARTE models.

5.1 Classification of UML/MARTE consistency types

We first categorize UML/MARTE consistency types used in our approach. Basically, there exist timing consistency and non-timing consistency as follows:

- Timing consistency: Timing specifications should be consistent among UML/MARTE models. For example, an execution time of a state in a timing diagram should be equal to an execution time of the same state in a state machine diagram.
- Non-timing consistency: Identifiers of UML/MARTE models should be consistently defined among UML/MARTE models. For example, messages used in an SD/MARTE should be defined in SMDs/MARTE.

In addition, we define intra-model consistency and inter-model consistency as follows:

- Intra-model consistency: A UML model should be consistent with its MARTE annotations. For example, an execution time of a state in an SMD/MARTE should be less than or equal to the deadline of the SMD/MARTE.
- Inter-model consistency: Different types of UML models with MARTE annotations should be consistent with each other. For example, timing specifications of an SD/MARTE should be consistent with timing specifications of SMDs/MARTE.

5.2 Timing consistency checking process

Figure 8 shows the overall timing consistency checking process for UML/MARTE behavioral models. The specifications of SMDs/MARTE, SDs/MARTE, and TDs/MARTE

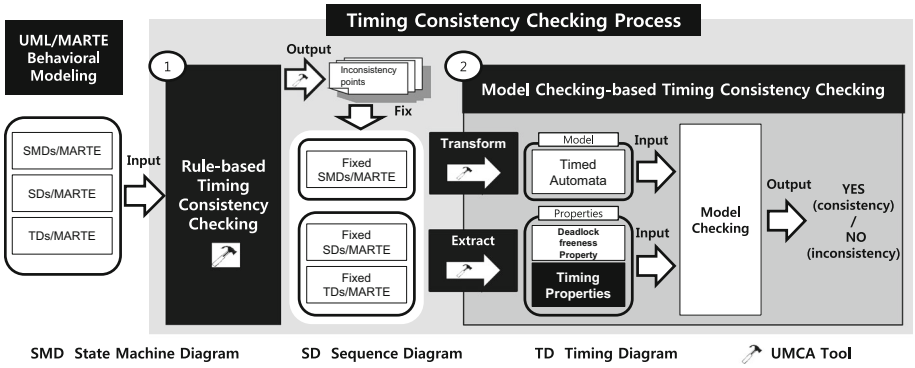


Fig. 8 Overall timing consistency checking process

should conform to the guidelines for UML/MARTE behavioral modeling. We use two UML modeling tools to describe UML/MARTE models because one tool alone cannot support our modeling guidelines. The Visual Paradigm for UML community edition (Paradigm 2012) is used to specify SMDs/MARTE and TDs/MARTE. The Papyrus tool (Papyrus 2012) is used to specify SDs/MARTE. The specified UML/MARTE behavioral models are used as inputs to the timing consistency checking process. The timing consistency checking process has two steps: a rule-based and a model checking-based timing consistency checking. In the rule-based timing consistency checking, we detect timing inconsistencies of two types: intra-models (i.e., a UML model and MARTE annotations) and inter-models (i.e., SMDs/MARTE, SDs/MARTE, and TDs/MARTE). The inconsistency points are reported to a RTES designer as output of the rule-based timing consistency checking. Using the results, we can fix the UML/MARTE behavioral models. In the model checking-based timing consistency checking, we check whether the timing behaviors of SDs/MARTE and TDs/MARTE are feasible against the timing behaviors of SMDs/MARTE. To this end, timed automata models are transformed from SMDs/MARTE, and timing properties are extracted from SDs/MARTE and TDs/MARTE for consistency checking. The transformed timed automata models and the generated timing properties are used as inputs to a model checker. If the timing property is not satisfied by the timing behaviors of timed automata models, then the model checker reports that the property is not satisfied. It means that the timing behaviors of an SD/MARTE or a TD/MARTE are not consistent with the timing behaviors of SMDs/MARTE. Otherwise, the model checker announces that the timing property satisfies the timing behaviors of timed automata models. We developed the UMCA (UML/MARTE timing Consistency Analyzer) tool to support the proposed approach. The UMCA provides the automatic capabilities of the rule-based timing consistency checking, model transformation, and timing property extraction. Model checking tools, UPPAAL (2012) and TIMES (2007), are used for the model checking-based timing consistency checking.

5.3 Rule-based timing consistency checking

We elicited elements related to intra-model and inter-model consistencies to make rules for consistency checking. Table 3 shows UML/MARTE elements related to intra-model consistency. To specify timing in UML/MARTE, we use the elements such as execution

Table 3 UML/MARTE elements related to intra-model consistency

Intra-model	Elements
SMD/MARTE	MARTE annotation of SMD
	MARTE annotation of state
SD/MARTE	MARTE annotation of lifeline
	MARTE annotation of execution specification
	Time observation
TD/MARTE	MARTE annotation of lifeline
	Duration
	Timing ruler

specification, time observation, duration, and timing ruler. These elements should be checked against timing constraints information of MARTE annotations.

Table 4 shows UML/MARTE elements related to inter-model consistency. A lifeline in an SD/MARTE and a TD/MARTE should have a corresponding SMD/MARTE. A receiving message and a sending message of lifelines in an SD/MARTE and a TD/MARTE should be defined in events and actions in an SMD/MARTE of the lifeline. States of a lifeline in a TD/MARTE should be defined in an SMD/MARTE of the lifeline. We should check the consistency among an execution time of an execution specification in an SD/MARTE, a duration (i.e., an execution time) of a state of a lifeline in a TD/MARTE, and an execution time of a state in an SMD/MARTE of the lifeline. MARTE annotation of a lifeline in an SD/MARTE and a TD/MARTE should be same as MARTE annotation of SMD of the lifeline.

Based on this analysis, we established consistency checking rules as shown in Tables 5 and 6. In Table 5, intra-model consistency checking rules focus on checking timing consistency between a UML model and MARTE annotations. In Table 6, inter-model consistency checking rules check not only timing consistency but also non-timing consistency among UML/MARTE models. The detailed information about each rule is described in "Appendix 1".

5.4 Model checking-based timing consistency checking

In a model checking-based timing consistency checking, we check timing behaviors of UML/MARTE models that passed a rule-based timing consistency checking. As inputs of a model checker, timed automata models are transformed from SMDs/MARTE and timing properties to be verified are extracted from SDs/MARTE and TDs/MARTE.

5.4.1 Model transformation

A timed automata model is composed of states and transitions between states (Alur and Dill 1994; Bérard et al. 2010). Also, it has a similar structure of an SMD (Knapp et al. 2002). The definition of a timed automata model is as follows:

Definition 4 (*Timed Automata*) A timed automata is a structure (N_{ta}, L_{ta}, G_{ta}, S_{ta}, A_{ta}, TR_{ta}) where:

- N_{ta} is a name of a timed automata.

Table 4 UML/MARTE elements related to inter-model consistency

SMD/MARTE	SD/MARTE	TD/MARTE
SMD name	Lifeline	Lifeline
Event, Action	Message	Message
State	–	State
MARTE annotation of state	MARTE annotation of execution specification	Duration
MARTE annotation of SMD	MARTE annotation of lifeline	MARTE annotation of lifeline

Table 5 Intra-model consistency checking rules

Intra-model consistency	Timing consistency
SMD/MARTE	SMD-MARTE ExecTime rule
	SMD-MARTE Deadline rule
SD/MARTE	SD-MARTE ExecTime rule
	SD-MARTE Deadline rule
	SD-MARTE TimeObservation rule
TD/MARTE	TD-MARTE ExecTime rule
	TD-MARTE Deadline rule
	TD-MARTE TimingRuler rule

Table 6 Inter-model consistency checking rules

Inter-model consistency	Non-timing consistency	Timing consistency
SD/MARTE-SMD/MARTE	SD-SMD Lifeline rule	SD-SMD MARTE rule
	SD-SMD Message rule	SD-SMD ExecTime rule
TD/MARTE-SMD/MARTE	TD-SMD Lifeline rule	TD-SMD MARTE rule
	TD-SMD Message rule	TD-SMD ExecTime rule
	TD-SMD State rule	
SD/MARTE-TD/MARTE	SD-TD Lifeline rule	SD-TD MARTE rule
		SD-TD ExecTime rule

- L_{ta} is a set of locations. A location has an invariant that represents a clock's duration at the location.
- G_{ta} is a set of guards. A guard is composed of a set of clock conditions and a set of flag conditions.
- S_{ta} is a set of synchronizations. A synchronization consists of a receiving message and a sending message.
- A_{ta} is a set of assignments. An assignment consists of a set of clock resets and a set of flag resets.
- $TR_{ta} : L_{ta} \times G_{ta} \times S_{ta} \times A_{ta} \rightarrow L_{ta}$ is a transition relationship.

All elements of a timed automata model are mapped into ones of an SMD/MARTE. Table 7 shows the corresponding elements of an SMD/MARTE and a timed automata

Table 7 Mapping between the elements of SMD/MARTE and timed automata

SMD/MARTE (N_{smd})	Timed automata (N_{ta})
State (S_{smd})	Location (L_{ta})
Transition (TR_{smd})	Transition (TR_{ta})
Receiving message in event (E_{smd})	Message with ? in synchronization (S_{ta})
Guard (G_{smd})	Guard (G_{ta})
Sending message in action (A_{smd})	Message with ! in synchronization (S_{ta})
Action except sending messages (A_{smd})	Assignment (A_{ta})
An execution time in MARTE linked to a state (MA_{smd})	Invariant of a location (L_{ta})

model in model transformation. Elements of an SMD/MARTE are translated into elements of a timed automata model in a straightforward manner. States, transitions, and guards in an SMD/MARTE are transformed into locations, transitions, and guards in a timed automata model, respectively. Receiving messages in event and sending messages in action of an SMD/MARTE are used in transforming synchronization of timed automata. An execution time or a blocking time of a state, which is described in MARTE annotation, is translated into an invariant of a location (i.e., a state). Also, a monitoring variable *run* is added to assignments in the transformed timed automata model. If a state has an execution time in MARTE annotation, then *run* is set to 1 in an assignment of an incoming transition of the location. Otherwise, *run* is set to 0. A monitoring variable *run* is used to extract timing properties from SDs/MARTE.

Figure 9 shows a timed automata model with a monitoring variable *run* that is translated from the SMD/MARTE of Fig. 5. Transformation of SMDs/MARTE into timed automata models is done automatically by the UMCA tool.

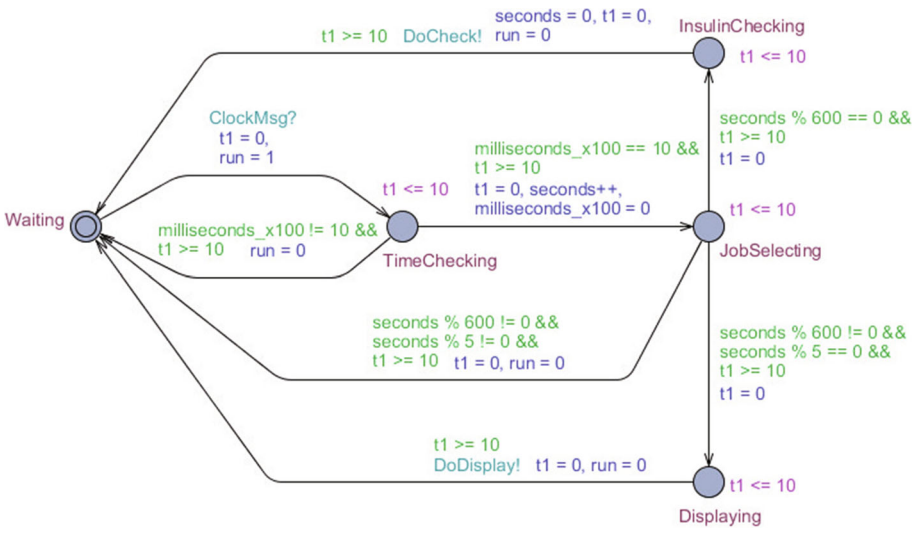


Fig. 9 Transformed timed automata model

5.4.2 Timing property extraction

Prior to timing property extraction, deadlock-freeness is checked against the transformed timed automata models to verify whether the models are executable. The deadlock freeness property is "A[ ] (not deadlock)". It means that there is no deadlock in any execution sequences. If a deadlock occurs in the timed automata models, we should modify SMDs/MARTE and conduct the rule-based timing consistency checking again.If there is no error in deadlock-freeness checking, timing properties are extracted from SDs/MARTE and TDs/MARTE. In our approach, a timing property has a form like "E< > property". It means that there exists at least one state satisfying the property. In our approach, an SD/MARTE and a TD/MARTE have the design patterns as shown in Figs. 10 and 11, respectively. Therefore, timing properties can be consistently extracted from an SD/MARTE and a TD/MARTE.

Figure 10 shows timing properties that are extracted from an SD/MARTE. Timing properties of an SD/MARTE check whether objects execute during the specified time ranges. Object names, time observation values, MARTE annotations (i.e., execution times) linked to execution specifications, and *run* variable are used to specify timing properties from an SD/MARTE. *run* variable checks whether an object is executing or not. A time range is composed of a time observation value and an execution time of an execution specification as shown in Fig. 10. A starting time is a time observation value. An ending time is the sum of a time observation value and an execution time of an execution specification. From an SD/MARTE in Fig. 10, we extract two timing properties. For example, "E< > (t > T1 and t < T1 + T_A) and A.run == 1" means that there exist at least one state satisfying the property that object A is running during a time range (i.e., t > T1 and t < T1 + T_A).

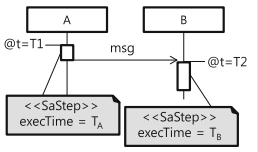
Design Pattern	Extracted Timing Properties
	• E<> (t > T1 and t < T1 + T_A) and A.run == 1 • E<> (t > T2 and t < T2 + T_B) and B.run == 1

Fig. 10 Timing properties extracted from an SD/MARTE

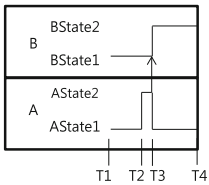
Design Pattern	Extracted Timing Properties
	• E<> (t > T1 and t < T2) and A.AState1 • E<> (t > T2 and t < T3) and A.AState2 • E<> (t > T1 and t < T3) and B.BState1 • E<> (t > T3 and t < T4) and B.BState2 • E<> (t > T3 and t < T4) and A.AState1

Fig. 11 Timing properties extracted from a TD/MARTE

Figure 11 shows timing properties that are extracted from a TD/MARTE. Object names, states, durations, and timing ruler values are used to generate timing properties as shown in Fig. 11.

Timing properties in a TD/MARTE check whether objects stay at the corresponding states during the specified time ranges. A time range is generated from a duration and timing ruler. A starting time is the beginning time of the duration in the timing ruler. An ending time is the termination time of the duration in the timing ruler. Five timing properties are extracted from a TD/MARTE in Fig. 11. For example, "E$<>$ ($t > T3$ and $t < T4$) and A.AState1" means that object A stays in AState1 state during a time range (i.e., $t > T3$ and $t < T4$).

6 Tool implementation

We describe the design and implementation of our timing consistency checking tool, UMCA (UML/MARTE timing Consistency Analyzer) (Choi 2015). The tool supports the rule-based timing consistency checking, model transformation, and timing property extraction. Also, UMCA is used to validate correctness of rules and model transformation. UMCA is built based on the Eclipse framework.

6.1 UMCA architecture

Figure 12 shows the overall architecture of the UMCA tool. Rules for the rule-based consistency checking, blocking states information, and UML/MARTE models are inputs to UMCA. Blocking states information is used to prevent wrong inconsistency detection by TD-MARTE ExecTime rule because UMCA cannot recognize whether the duration of a state in a TD/MARTE is an execution time or a blocking time. As shown in Fig. 12, UMCA consists of XML parser, rule repository, rule checker, model converter, property extractor, and model reconverter.

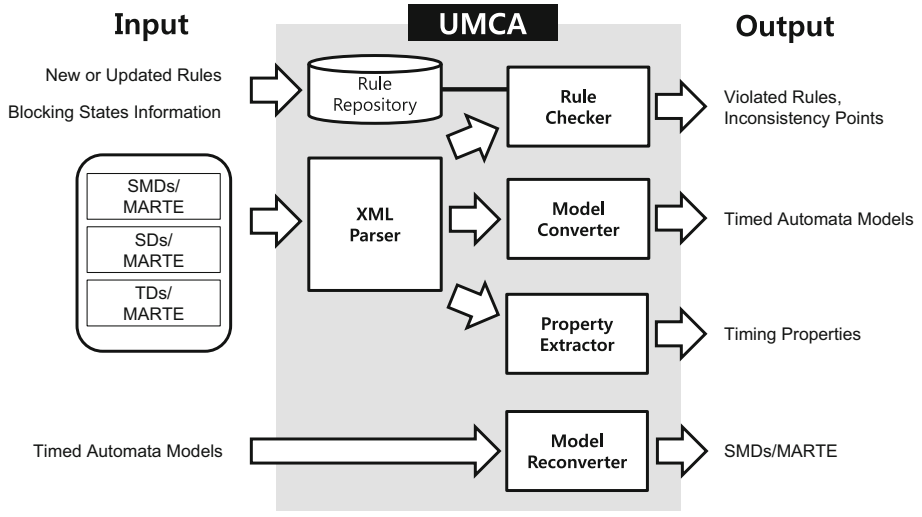


Fig. 12 UMCA architecture

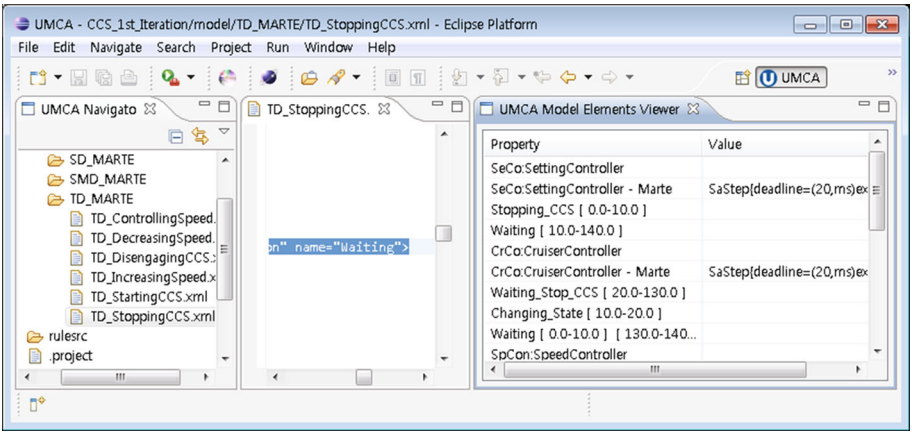


Fig. 13 UMCA navigator and model elements viewer

6.2 XML parser

XML parser extracts and displays the elements of each UML/MARTE model from XML files of UML/MARTE models. Figure 13 shows the navigator and model elements viewer of UMCA. If an XML file is selected in the UMCA navigator, then the model elements information is shown in the UMCA model elements viewer. When an element is selected in the UMCA model elements viewer, the XML file is opened automatically and the element position is highlighted as shown in Fig. 13.

6.3 Rule repository and rule checker

The rule repository manages rules for the rule-based timing consistency checking. At present, the rule repository has 20 rules described in Sect. 5. A rule is implemented with Java code and is composed of two parts: rule identifier and rule condition. The rule identifier is implemented with Java annotation to define basic information of a rule. Rule condition checks inconsistency points and saves the inconsistency information.

For example, Fig. 14 shows TD-MARTE deadline rule. TD-MARTE deadline rule defines that deadline should be greater than or equal to an execution time of a TD/MARTE. Rule identifier is defined at line 11. Rule condition is defined from line 17 to line 28. Elements (i.e., a deadline and an execution time) to be checked are extracted from line 21 and line 22. At line 23, a checking is done about whether a TD/MARTE conforms to the rule with the elements. If an execution time is greater than a deadline, the specification of TD/MARTE violates the rule. Then, the inconsistency point (i.e., a MARTE annotation) is saved to display in the checking result viewer of UMCA at line 24.

The rule checker detects inconsistency points using the consistency checking rules in the rule repository. The detected inconsistencies are displayed in the UMCA consistency checking results viewer as shown in Fig. 15. If the detected inconsistency is selected in the consistency checking results viewer, the corresponding element position is highlighted in the XML file. The element is also highlighted in the UMCA model elements viewer.

```
1  package intrarules;
2
3  import java.util.*;
4
5  import com.kaist.se.consistencychecker.*;
6  import com.kaist.se.consistencychecker.item.*;
7  import com.kaist.se.consistencychecker.diagram.*;
8  import com.kaist.se.consistencychecker.ConsistencyRule;
9
10 // Rule Identifier
11 @Rule(name = "TD-MARTE Deadline", type = "SINGLE", Object1 = "TimingDiagram")
12
13 public class TDMarteDeadline extends ConsistencyRule {
14     public List check(Diagram diagram) {
15
16         // Rule Condition
17         List re = new ArrayList();
18             TimingDiagram td = (TimingDiagram) diagram;
19             for (TDLifeLine lifeLine : td.getLifeLineList()) {
20                 if (lifeLine.getMarte() != null) {
21                   Double execTimeLifeLine = lifeLine.getMarte().getNumberValue(Marte.EXECTIME);
22                   Double deadlineLifeLine = lifeLine.getMarte().getNumberValue(Marte.DEADLINE);
23                   if (deadlineLifeLine < execTimeLifeLine) {
24                     re.add(lifeLine.getMarte());
25                   }
26                 }
27             }
28             return re;
29     }
30 }
```

Fig. 14 Example of rule

6.4 Model converter and model reconverter

The model converter translates SMDs/MARTE into timed automata models for the model checking-based timing consistency checking. The corresponding elements of an SMD/MARTE and a timed automata model are shown in Table 7 of the Sect. 5. In addition, a monitoring variable *run* is added to assignments in a timed automata model to check running status of an object. Conversely, the model reconverter generates SMDs/MARTE from the timed automata models.

6.5 Property extractor

Property extractor generates timing properties from timing behaviors of SDs/MARTE and TDs/MARTE. Timing properties are used as inputs to the model checking-based timing consistency checking. Basically, the timing properties are saved as query file format for the UPPAAL model checker. Timing properties of SDs/MARTE are used to check whether objects execute during the specified time ranges. Timing properties of TDs/MARTE are used to check whether objects stay at the corresponding states during the specified time ranges.

6.6 Correctness test of rules and model transformation

UMCA tool is used to test the correctness of rules. To this end, an inconsistency related to each rule was injected in UML/MARTE models. Then, we checked whether the

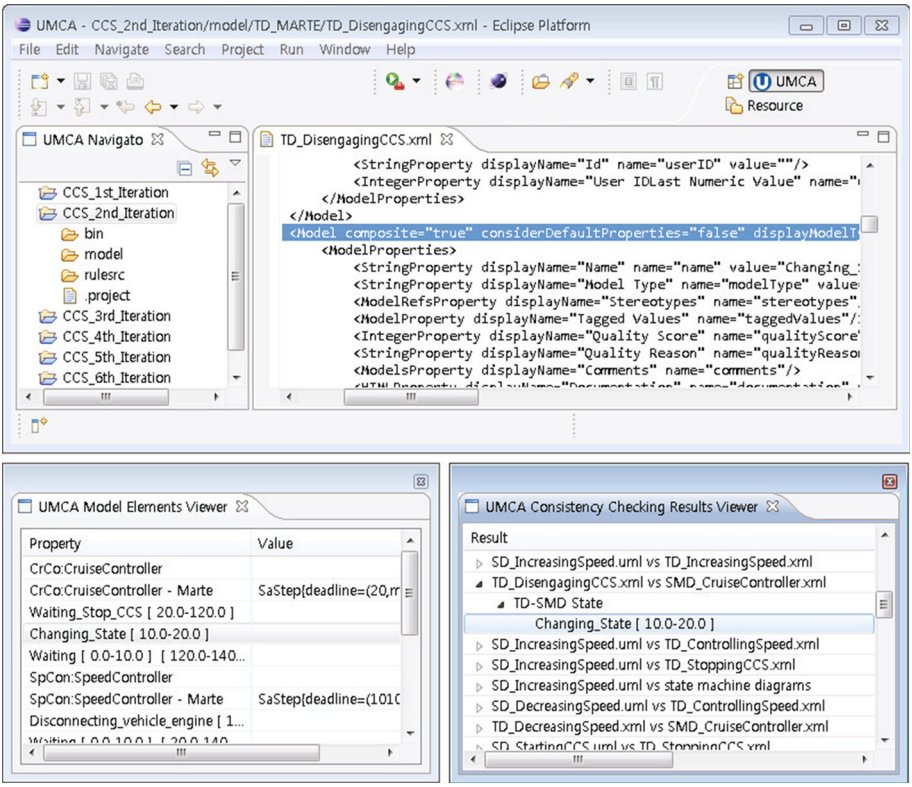


Fig. 15 UMCA consistency checking results viewer

inconsistency can be detected by the rule that is implemented in UMCA. Through this process, we tested the correctness of rules for intra-model and inter-model consistency. To test correctness of timed automata transformation, we retranslated the SMDs/MARTE from the transformed timed automata models. Then, we compared the retranslated SMDs/MARTE with the original SMDs/MARTE. Figure 16 shows an example of model transformation by UMCA. As shown in Fig. 16, except for the MARTE annotation of an SMD/MARTE, all elements are retranslated correctly from the timed automata model. MARTE annotation of an SMD/MARTE cannot be generated from the timed automata model because a timed automata model does not have timing constraints information such as priority, deadline, and execution time.

7 A case study for cruise control system software

CCS software is safety-critical real-time embedded software used in automotive systems. It is used to manage the speed of a vehicle via an automated control. If the speed is not controlled correctly, it can lead to catastrophic system failures. The goal of this case study is to validate whether the proposed timing consistency checking approach can effectively detect inconsistencies in UML/MARTE models.

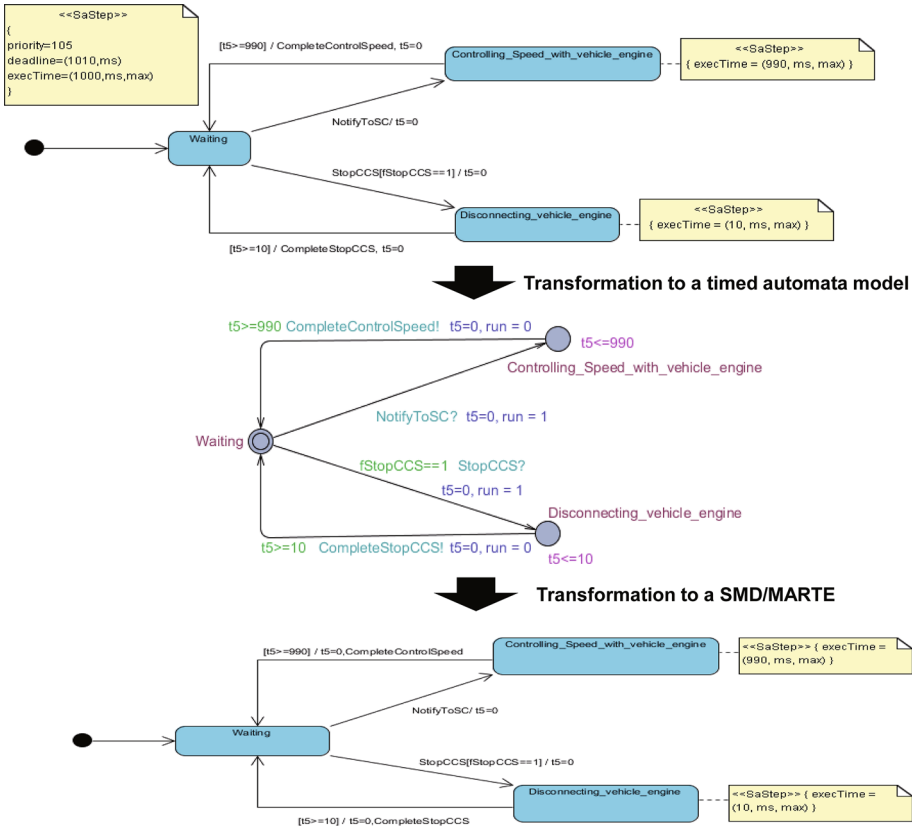


Fig. 16 Correctness test of model transformation using UMCA

7.1 Experimental setting

Based on literature (Wieringa 2003), we elicited six classes as shown in Fig. 17. SettingController sends an input signal (i.e., turn on, turn off, speed up, or speed down) to CruiseController. CruiseController receives the inputs and checks the current vehicle speed from SpeedCounter. CruiseController can also receive an input (i.e., braking or accelerating) directly from a user. SpeedCounter calculates the current speed of a vehicle. If the vehicle speed should be controlled, CruiseController sends a command to SpeedController. SpeedController increases or decreases the current vehicle speed according to the command of CruiseController. Display shows the current speed of a vehicle. User is used to generate input signals for timing scenarios that are specified by SDs/MARTE and TDs/MARTE. In this case study, User generates turn on, turn off, turn on, speed up, speed down, and turn off events every 10 s.

Two graduate students at computer science department in KAIST participated in UML/MARTE behavioral modeling of this case study. We provided them with CCS software requirements and UML/MARTE modeling guidelines. And then, they specified UML/MARTE behavioral models for CCS software. With the specified UML/MARTE models, we conducted the timing consistency checking process. Six SMDs/MARTE, six SDs/

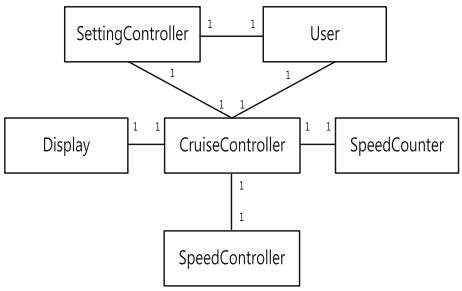


Fig. 17 Class diagram of CCS software

Table 8 Results of the timing consistency checking

Iteration	1st	2nd	3rd	4th	5th
Experimental time (days)	16	1	1	1	1
Rule-based timing consistency checking					
SMD-MARTE ExecTime	3	0	1	0	0
SD-MARTE ExecTime	2	0	0	0	0
SD-SMD MARTE	20	0	0	0	0
SD-SMD Lifeline	8	0	0	0	0
SD-SMD Message	8	0	0	0	0
TD-SMD MARTE	14	0	1	0	0
TD-SMD Lifeline	1	0	0	0	0
TD-SMD ExecTime	4	0	4	0	0
TD-SMD Message	8	0	0	0	0
TD-SMD State	2	0	0	0	0
SD-TD MARTE	59	0	4	0	0
SD-TD Lifeline	2	0	0	0	0
Model checking-based timing consistency checking					
Deadlock-freeness	–	deadlock	–	Pass	Pass
SD-SMD Timing	–	–	–	6	0
TD-SMD Timing	–	–	–	21	0
Total number of inconsistencies	131	0	10	27	0

MARTE, and six TDs/MARTE models were specified in the CCS software case study. The final SMDs/MARTE that passed the checking process are shown in Appendix 2.

7.2 Timing consistency checking results and analysis

Table 8 shows the timing consistency checking results for the case study of CCS software. An iteration consists of UML/MARTE behavioral modeling and timing consistency checking. As mentioned earlier, two graduate students specified SMDs/MARTE, SDs/MARTE, and TDs/MARTE. We checked timing consistencies of the specified UML/MARTE models using the UMCA tool. At next iteration, they fixed the inconsistency points of UML/MARTE models with the consistency checking results. We count one for each pair of related inconsistency points (e.g., two mismatched timing information between an SMD/MARTE and an SD/MARTE) detected by the UMCA.

As shown in Table 8, we spent 20 days for the five iterations to make consistent UML/MARTE behavioral models. Even though an iteration is finished in less than one day, we adopted a policy of using at least one day for an iteration to prevent unexpected modeling errors and to communicate effectively. At the first iteration, the students spent much time to describe UML/MARTE models because they were not accustomed to the modeling tools (i.e., Papyrus and Visual Paradigm) and did not understand fully the CCS software requirements. One hundred and thiry-one inconsistencies were detected at the first iteration. Most inconsistencies were due to mismatched timing information. At the second iteration, inconsistencies were not detected in the rule-based timing consistency checking. However, a deadlock was detected in the model checking-based timing consistency checking. At third iteration, SMDs/MARTE were modified to solve the deadlock but 10 inconsistencies were identified because the timing constraints information in TDs/MARTE were not updated. The rule-based timing consistency checking and deadlock-freeness checking passed at the fourth iteration. However, in the model checking-based timing consistency checking, 27 timing inconsistencies were detected due to incorrect timing specifications of SDs/MARTE and TDs/MARTE. At the fifth iteration, six SMDs/MARTE, six SDs/MARTE, and six TDs/MARTE were specified consistently.

7.2.1 Analysis 1: Inconsistencies in the rule-based timing consistency checking

We detected MARTE, Message, ExecTime, Lifeline, and State inconsistencies in this case study. Six inconsistencies were detected in checking intra-model rules, while 135 inconsistencies were identified in checking inter-model rules. The detected inconsistencies were classified into 112 timing inconsistencies and 29 non-timing inconsistencies.

Figure 18 shows an example of an SMD/MARTE with an intra-model inconsistency that was detected in the rule-based timing consistency checking. Three SMD-MARTE

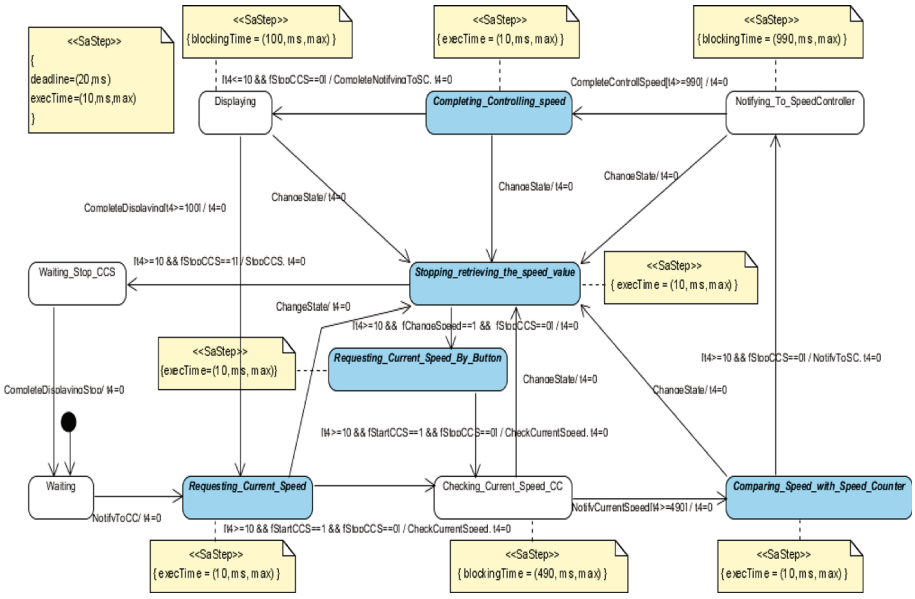


Fig. 18 CruiseController SMD/MARTE at the first iteration

ExecTime inconsistencies were detected in the CruiseController SMD/MARTE of Fig. 18. An execution time of CruiseController is 10 ms in Fig. 18. However, execution times (i.e., 30 ms) of three execution paths are greater than the execution time of CruiseController. The first execution path is composed of Requesting_Current_Speed, Stopping_retrieving_the_speed_value, and Requesting_Current_Speed_By_Button states. The second path consists of Comparing_Speed_with_Speed_Counter, Stopping_retrieving_the_speed_value, and Requesting_Current_Speed_By_Button states. The last execution path is composed of Completing_Controlling_speed, Stopping_retrieving_the_speed_value, and Requesting_Current_Speed_By_Button states.

7.2.2 *Analysis 2: Inconsistencies in the model checking-based timing consistency checking*

Table 9 shows the total number of timing properties that were extracted from SDs/MARTE and TDs/MARTE. At the fourth iteration, 34 timing properties were extracted from SDs/MARTE and six timing properties were not satisfied by the behaviors of SMDs/MARTE. In the case of TDs/MARTE, 99 timing properties were generated and 21 timing properties were not satisfied by the behaviors of SMDs/MARTE. At the fifth iteration, all timing properties were satisfied.

Figure 19 shows an example of a TD/MARTE with timing inconsistency at the fourth iteration. This TD/MARTE has a scenario of disengaging a CCS from 0 ms to 140 ms. Ten timing properties were extracted from the TD/MARTE. However, some timing properties were not satisfied by the behaviors of SMDs/MARTE as shown in Fig. 20. For example, the Disconnecting_vehicle_engine state cannot be reached from 10 ms to 20 ms. It means

Table 9 Total number of extracted timing properties

Iteration	4th	5th
Total number of timing properties extracted from SDs/MARTE	34	34
Total number of timing properties extracted from TDs/MARTE	99	101

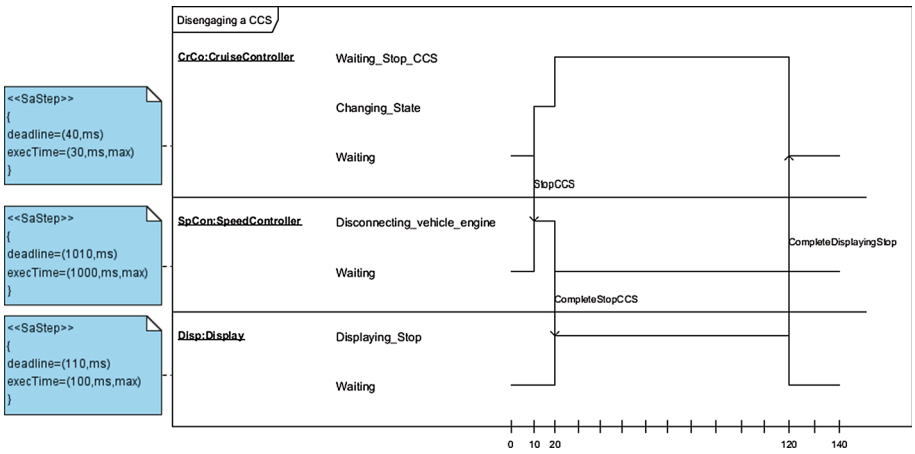


Fig. 19 TD/MARTE for disengaging a CCS at the fourth iteration

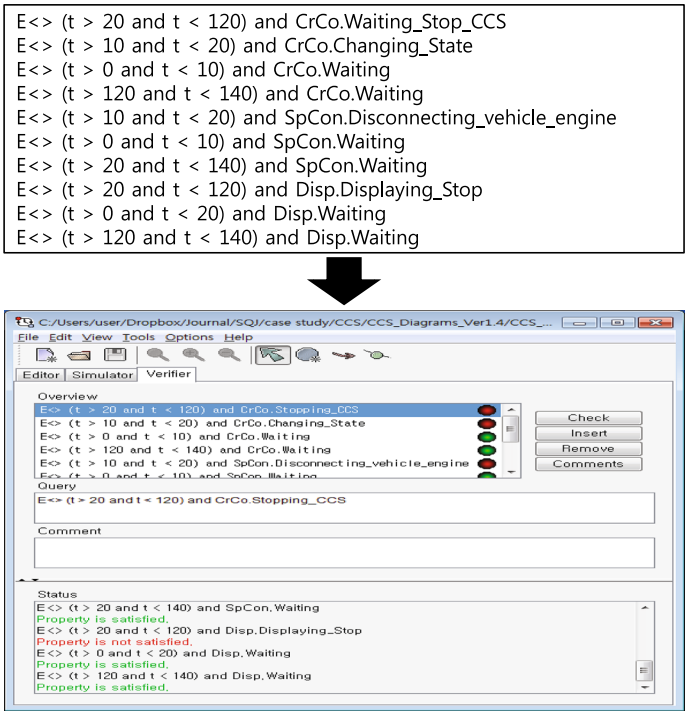


Fig. 20 Extracted timing properties and checking results

that the scenario of the TD/MARTE was not feasible against the behaviors of SMDs/MARTE for CCS software. Because CruiseController does not send the StopCCS message to SpeedController in the time range of the TD/MARTE. In this time range, a turn on event is generated as described in Sect. 7.1. Thus, the time range of Fig. 19 should be modified to reach the Disconnecting_vehicle_engine state.

8 A case study for guidance and control unit software

A GCU is a safety-critical real-time embedded system used in avionics systems. Figure 21 shows the control structure of GCU in an avionics system. GCU software controls GCU hardware resources and communicates with other subsystems (i.e., sensors and actuators). Also, it executes flight-related algorithms and functions. Diverse experts in the fields of aerospace, electronics, mechanics, and computer science participate in a project to develop a GCU. Thus, a common understanding among stakeholders is very important. The goal of a case study for GCU software is to validate the practicality of the proposed approach.

8.1 Experimental setting

Objectives of UML/MARTE behavioral modeling in the GCU software development are to understand and analyze timing constraints of GCU software among stakeholders. Figure 22 shows object (i.e., task in implementation phase) execution structure in flight mode of

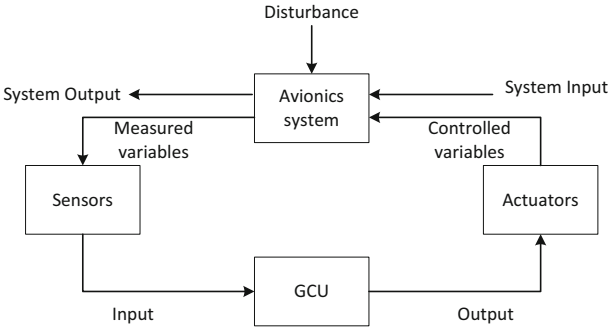


Fig. 21 Control structure of GCU

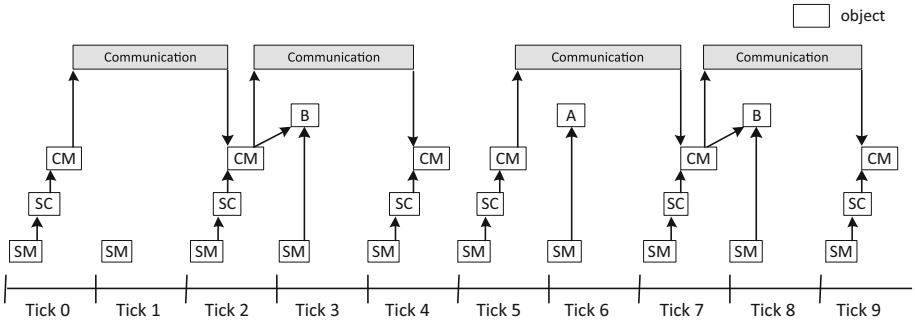


Fig. 22 Object execution structure in flight mode of GCU software

GCU software. The objects are executed at regular tick interval as shown in Fig. 22. Ticks are generated every 20 tu starting from 0 tu. We use tu to describe time unit of GCU software. In flight mode, GCU software consists of five objects (i.e., SM, SC, CM, A, and B) as shown in Fig. 22. SM (System Manager) executes high priority jobs per 20 tu. System Controller (SC) handles analog-to-digital conversion (ADC) and digital-to-analog conversion (DAC) to control GCU hardware. Communication manager (CM) manages MIL-STD-1553B communication. A and B execute flight-related functions. In addition, interrupt service routine (ISR) and communication controller (CC) are specified to represent tick generations and communications as shown in Fig. 22, respectively. Table 10 shows timing constraints information of SM, SC, CM, A, B, and CC in GCU software. For example, the worst-case execution time (WCET) of SM is 3 tu and the deadline of SM is 5 tu. The WCET of ISR is not described because ISR is executed instantaneously.

With the above GCU software requirements and help of domain experts at ADD in the Republic of Korea, we specified seven SMDs/MARTE, five SDs/MARTE, and 10 TDs/MARTE models.

8.2 Timing consistency checking results and analysis

Table 11 shows the results of the timing consistency checking for UML/MARTE behavioral models of GCU software. We spent 14 days and conducted five iterations to specify consistent UML/MARTE behavioral models in this case study. Also, we spent at least one day for an iteration like with the CCS software case study. Even though we understood

Table 10 Timing constraints information

Object	WCET	Deadline	Priority
SM	3	5	High
SC	4	5	Middle
CM	8	10	Middle
B	10	20	Low
A	15	20	Low
CC	30	–	–

Table 11 Results of the timing consistency checking

Iteration Experimental time (days)	1st 10	2nd 1	3rd 1	4th 1	5th 1
Rule-based timing consistency checking					
SMD-MARTE ExecTime	4	0	0	0	0
SD-MARTE Deadline	1	0	0	0	0
SD-MARTE TimeObservation	1	1	0	0	0
TD-MARTE ExecTime	4	0	0	0	0
SD-SMD MARTE	7	0	0	0	0
SD-SMD Lifeline	4	0	0	0	0
SD-SMD Message	1	0	0	0	0
SD-SMD ExecTime	3	1	0	0	0
TD-SMD MARTE	16	0	0	0	0
TD-SMD Lifeline	2	0	0	0	0
TD-SMD Message	12	0	0	0	0
TD-SMD ExecTime	6	0	0	0	0
TD-SMD State	30	0	0	0	0
SD-TD MARTE	11	0	0	0	0
SD-TD Lifeline	11	0	0	0	0
SD-TD ExecTime	2	1	0	0	0
Model checking-based timing consistency checking					
Deadlock-freeness	–	–	Pass	Pass	Pass
SD-SMD Timing	–	–	3	0	0
TD-SMD Timing	–	–	20	7	0
Total number of inconsistencies	115	3	23	7	0

GCU software behavior through previous studies (Choi et al. 2011a, b, 2012a, b), the UMCA tool detected 115 inconsistencies in the specified UML/MARTE models at the first iteration. In particular, 30 State inconsistencies were detected between SDs/MARTE and TDs/MARTE because of a case-sensitivity problem. At the second iteration, three inconsistencies were detected. The inconsistencies existed in an SD/MARTE and a TD/MARTE used at the first iteration. However, they were not detected at the first iteration because of lifeline inconsistencies. Moreover, TimeObservation inconsistency was detected in the updated SD/MARTE. From the third iteration, UML/MARTE models passed the rule-based timing consistency checking. The transformed timed automata models also passed deadlock freeness checking. Twenty-three timing inconsistencies were detected in the model checking-based timing consistency checking at the third iteration. At the fourth

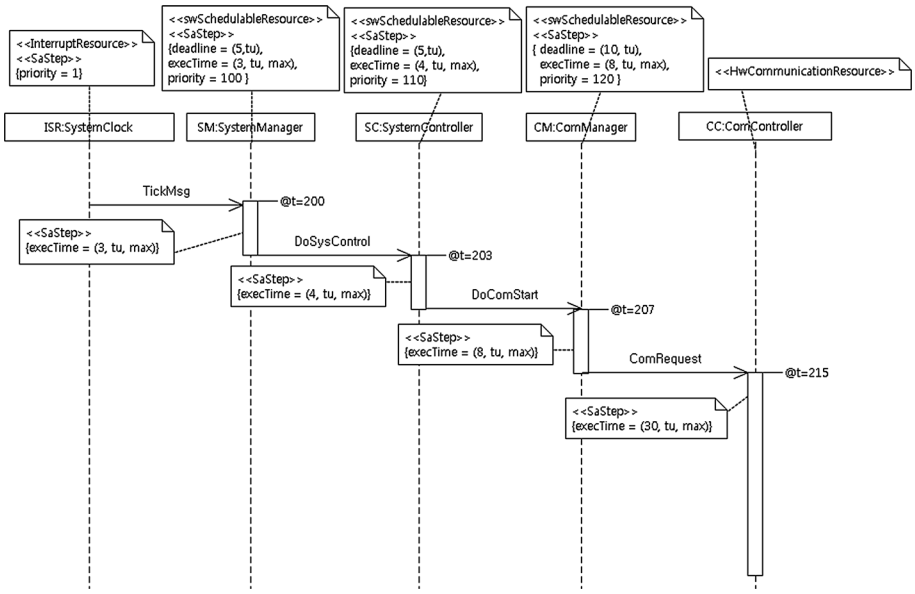


Fig. 23 SD/MARTE at the second iteration

iteration, seven Timing inconsistencies were detected in TDs/MARTE because of wrong modification for TDs/MARTE. All UML/MARTE models were specified consistently at the fifth iteration.

8.2.1 Analysis 1: Inconsistencies in the rule-based timing consistency checking

The most frequent inconsistencies were detected by TD-SMD State rule because we did not consider case-sensitiveness in specifying state names of TDs/MARTE. We found 11 inconsistencies in checking intra-model consistency rules, while we detected 107 inconsistencies in checking inter-model consistency rules. The inconsistencies were also classified into 53 timing inconsistencies and 62 non-timing inconsistencies. Figures 23 and 24 show an example of inter-model inconsistency that was detected by the SD-SMD ExecTime rule. In Fig. 23, an execution time of an execution specification of ComManager in the SD/MARTE is 8 tu. In the ComManager SMD/MARTE of Fig. 24, the corresponding execution path is found by using DoComStart message and ComRequest message. There exists an execution path that is composed of TxRxStarting state. However, an execution time of the path is 5 tu. It is not equal to the execution time of ComManager in the SD/MARTE.

Thus, SD-SMD ExecTime inconsistency exists between the SD/MARTE and the SMD/MARTE in Figs. 23 and 24.

8.2.2 Analysis 2: Inconsistencies in the model checking-based timing consistency checking

The model checking-based timing consistency checking was conducted at the third, fourth, and fifth iterations. Table 12 shows the total number of timing properties generated from SDs/MARTE and TDs/MARTE.

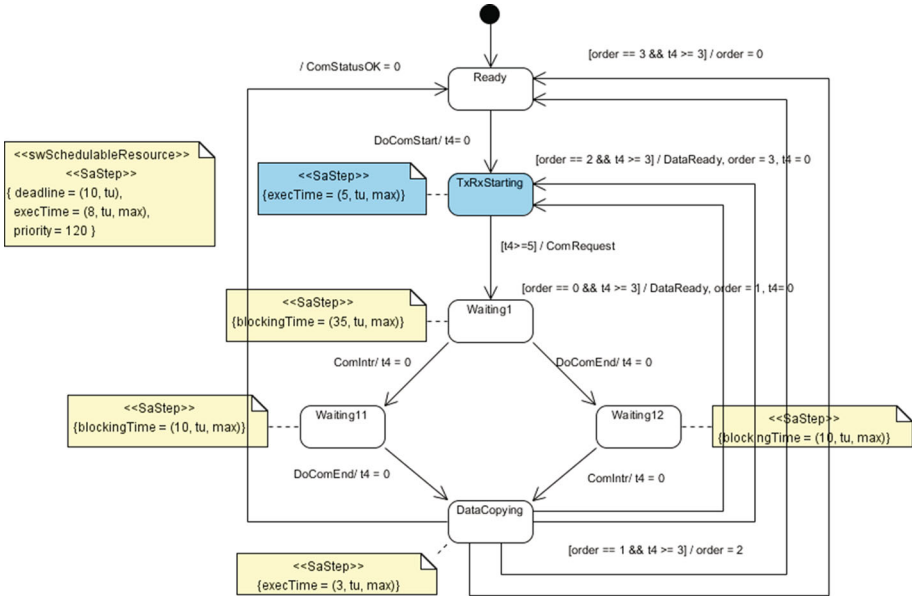


Fig. 24 ComManager SMD/MARTE at the second iteration

Table 12 Total number of extracted timing properties

Iteration	3rd	4th	5th
Total number of timing properties extracted from SDs/MARTE	14	14	14
Total number of timing properties extracted from TDs/MARTE	105	105	101

At the third iteration, 14 timing properties were generated from SDs/MARTE and three properties did not satisfy the behaviors of SMDs/MARTE. One hundred and five timing properties were generated from TDs/MARTE, and 20 timing properties were not satisfied by the behaviors of SMDs/MARTE. At the fourth iteration, all timing properties of SDs/MARTE satisfy the behaviors of SMDs/MARTE. However, seven timing properties of TDs/MARTE failed to pass the model checking-based timing consistency checking because of wrong modifications of TDs/MARTE. At the fifth iteration, all timing properties of SDs/MARTE and TDs/MARTE were satisfied by the behaviors of SMDs/MARTE.

Figures 25, 26, and 27 show a TD/MARTE for tick 1 interval at the third, fourth, and fifth iteration, respectively. In Figure 25, "E$<>$ (t $>$ 10 and t $<$ 13) and SM.Tick1_Working" property detected a timing inconsistency of the TD/MARTE because the time range (i.e., from 10 to 13) of Fig. 25 is not included in the time range of tick 1. We fixed the range of the timing ruler and checked the TD/MARTE of Fig. 26 at the fourth iteration. However, an inconsistency was also detected by the timing property "E$<>$ (t $>$ 23 and t $<$ 29) and SM.Tick1_Ready" because the next state of Tick1_Working is not Tick1_Ready but Tick2_Ready. Figure 27 shows the TD/MARTE that is fixed at the fifth iteration.

Figure 28 shows an SD/MARTE that was used at the first iteration. Two timing properties are extracted from the SD/MARTE as shown in Fig. 29. The second property

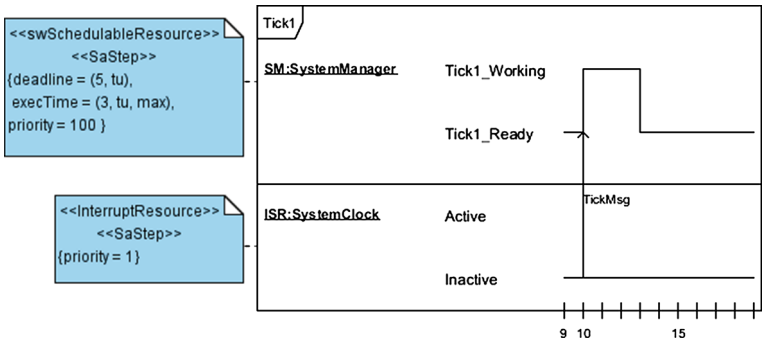


Fig. 25 TD/MARTE for Tick1 interval at the third iteration

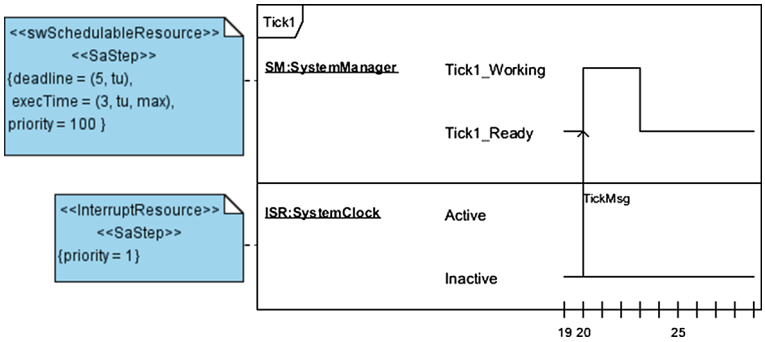


Fig. 26 TD/MARTE for Tick1 interval at the fourth iteration

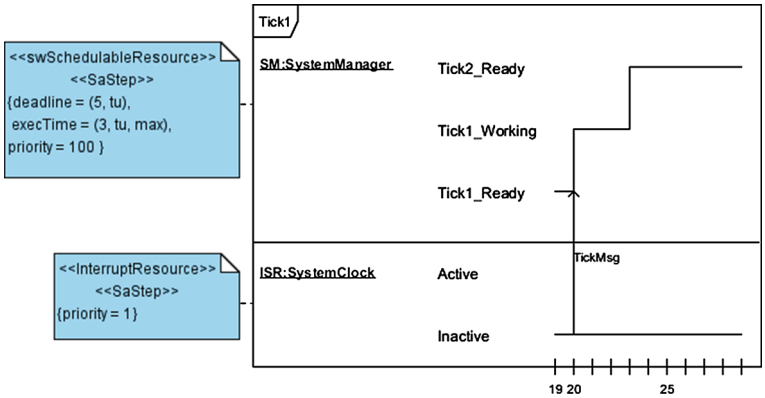


Fig. 27 TD/MARTE for Tick1 interval at the fifth iteration

was not satisfying the timing behaviors of SMDs/MARTE for GCU software because the timing behaviors of A violated the object execution structure in Fig. 22. To specify the timing scenario of the SD/MARTE consistently, the time observation values of SM and A should be changed to 320 tu and 323 tu, respectively.

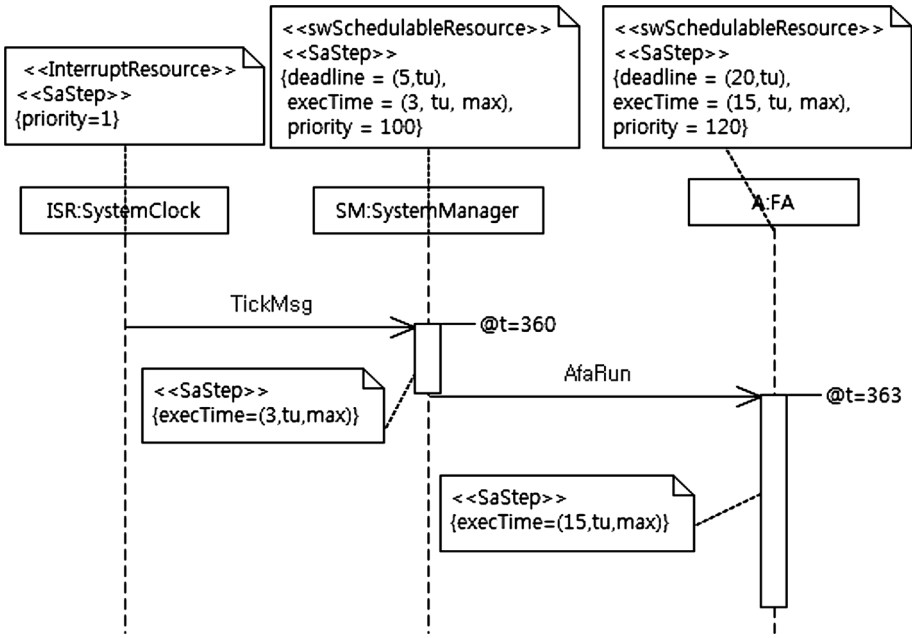


Fig. 28 SD/MARTE used at the first iteration

9 Discussion

We demonstrated the advantages of the timing consistency checking approach with two case studies. We observed that many inconsistencies exist in UML/MARTE models at the first iteration. However, all participants of the case studies gained a full understanding for the timing behaviors of RTES through the timing consistency checking.

In our approach, TDs/MARTE models are input to timing consistency checking, even though SDs/MARTE models describe timing scenarios of RTES. From our experimental observation, we confirm that TDs/MARTE provide more intuitive insights to stakeholders in timing aspects. However, it is a burden to software designers to specify TDs/MARTE. In another study, we studied about automatic construction of TDs/MARTE from SMDs/MARTE and SDs/MARTE to provide different viewpoints of SDs/MARTE models and save time to specify TDs/MARTE (Choi and Bae 2012; Choi et al. 2012b; Nguyen et al. 2014).

We implemented the automated timing consistency checking tool, UMCA, to validate the proposed approach. With the UMCA tool, we showed that our approach can be used in UML/MARTE model-driven development approach.

We consulted with the domain experts at ADD in the Republic of Korea about the results of the two case studies. They agreed that our approach is useful for an effective understanding and communication in the development of RTES.

Usually defense software has diverse temporal events during a long execution time and non-complicated structure to assure reliability and safety. By specifying SDs/MARTE or TDs/MARTE with diverse timing scenarios, our approach can effectively check timing behaviors of defense software in the design phase.

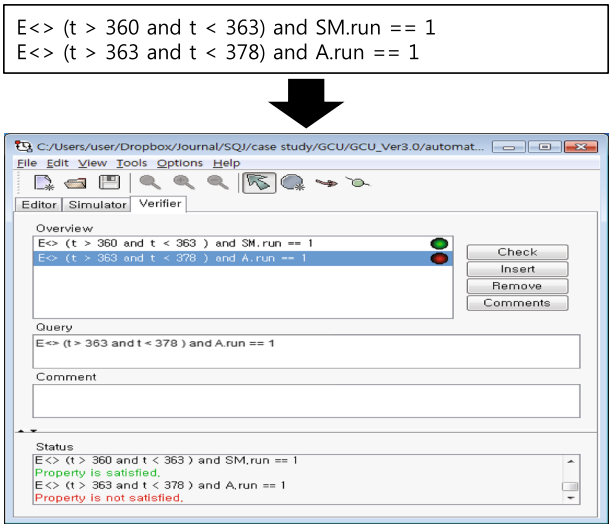


Fig. 29 Extracted timing properties and checking results

In addition, we can specify executable SMDs/MARTE through the timing consistency checking. It means that our approach can be extended to generate code from the consistent SMDs/MARTE by utilizing TIMES tool (Amnell et al. 2002; Amnell et al. 2004; Choi et al. 2013).

In our approach, timing information is important to specify UML/MARTE models for RTES. In the development of defense software, many experiments regarding timing are done with an evaluation board or a prototype platform in the exploratory development process. Also, if RTES is for an evolutionary product, we can refer the timing information of RTES in the previous project. Thus, timing information can be made in RTES requirements analysis of a system development process.

Although the two case studies showed the effectiveness and practicality of our approach, we have several limitations. The proposed checking rules focus mainly on detecting timing inconsistencies such as mismatched timing information and incorrect timing behaviors among UML/MARTE behavioral models. Thus, some inconsistencies cannot be detected by the rules. The validations of the rules and model transformation are not proved in a formal way. Instead, we showed the correctness of rules through an inconsistency injection test. We tested the correctness of model transformation with model retransformation using UMCA. At present, we need two UML/MARTE modeling tools to follow the proposed approach. We believe that the Papyrus tool will support our approach in the near future. Then, UMCA will be modified to support it.

10 Conclusion

This paper describes an approach to check timing consistency among SMDs/MARTE, SDs/MARTE, and TDs/MARTE for RTES. We first showed how UML/MARTE models specify the behaviors of RTES formally to provide a common understanding among stakeholders. In addition, we classified UML/MARTE consistency types into intra-model and inter-model consistency, and timing and non-timing consistency. The specified UML/

MARTE models are validated through a rule-based and a model checking-based consistency checking process. In the rule-based consistency checking, we check intra-model and inter-model consistency for UML/MARTE models. To this end, we provided eight intra-model and 12 inter-model consistency checking rules. Intra-model consistency checking rules check timing consistency in a UML/MARTE model, while inter-model consistency checking rules check non-timing consistency as well as timing consistency among different types of UML/MARTE models. In the model checking-based consistency checking, we check whether the timing scenarios of SDs/MARTE and TDs/MARTE are consistent with the behaviors of SMDs/MARTE. For this purpose, SMDs/MARTE are translated into timed automata models. Timing properties are extracted from SDs/MARTE and TDs/MARTE. The translated timed automata models and timing properties are used as inputs to a model checker. We developed an automatic timing consistency checking tool UMCA for a seamless approach. UMCA supports a rule-based timing consistency checking, model transformation, and timing property extraction. To show the effectiveness and practicality of the timing consistency checking approach, we demonstrated the proposed approach with two case studies; CCS software in automotive systems and GCU software in avionics systems. We showed that the proposed approach helps software designers specify consistent UML/MARTE models. Software designers can understand timing behaviors of RTES fully through the UML/MARTE behavioral modeling and the timing consistency checking. Moreover, the proposed approach can detect infeasible timing scenarios in the design phase of RTES.

There are several works to mature the proposed approach. First, we will conduct additional studies to solve the limitations. Rule completeness should be proved formally. UMCA also will be updated to support user-friendly checking. Finally, we will study how to detect timing inconsistencies in UML/MARTE models for multiprocessor system software.

Appendix 1: Rules for rule-based timing consistency checking

Intra-model consistency checking rules

SMD-MARTE ExecTime

SMD-MARTE ExecTime rule defines that an execution time of an SMD/MARTE should be greater than or equal to an execution time of the following two conditions:

1. An execution time of each state.
2. Execution times of execution paths from a state with a receiving message in a incoming transition to a state with a sending message in a outgoing transition only if all states in execution paths have an execution time. ⌟

Rule. SD-MARTE ExecTime

SD-MARTE ExecTime rule defines that an execution time of a lifeline in an SD/MARTE should be greater than or equal to an execution time of each execution specification in the lifeline. ⌟

Rule. TD-MARTE ExecTime

TD-MARTE ExecTime rule defines that if duration of a state represents an execution time, an execution time of a lifeline in a TD/MARTE is greater than or equal to the following two conditions:

1. An execution time of each state.

2. Execution times of execution paths from a state with a receiving message to a state with a sending message. ⌟

Rule. SMD-MARTE Deadline
Rule. SD-MARTE Deadline
Rule. TD-MARTE Deadline

SMD-MARTE Deadline, SD-MARTE Deadline, and TD-MARTE Deadline rules define that deadline should be greater than or equal to an execution time of a UML/MARTE model. ⌟

Rule. SD-MARTE TimeObservation

SD-MARTE TimeObservation rule defines that a time observation value in an SD/MARTE is the sum of a time observation value and an execution time of the previous execution specification in the SD/MARTE. ⌟

Rule. TD-MARTE TimingRuler

TD-MARTE TimingRuler rule defines that the timing ruler values in a TD/MARTE should be increased at regular intervals. ⌟

Inter-model consistency checking rules

Rule. SD-SMD MARTE

SD-SMD MARTE rule defines that MARTE annotation linked to a lifeline in an SD/MARTE should be the same as a MARTE annotation linked to an SMD/MARTE of the lifeline. ⌟

Rule. SD-SMD Lifeline

SD-SMD Lifeline rule defines that a lifeline in an SD/MARTE should have a corresponding SMD/MARTE. ⌟

Rule. SD-SMD Message

SD-SMD Message rule defines that a receiving message and a sending message of lifelines in an SD/MARTE should be defined in events and actions in an SMD/MARTE of the lifeline, respectively. ⌟

Rule. SD-SMD ExecTime

SD-SMD ExecTime rule defines that an execution time of an execution specification in an SD/MARTE is greater than or equal to an execution time of a state in an SMD/MARTE for the following three conditions:

1. An execution time of an execution specification with only a receiving message in an SD/MARTE should be greater than or equal to an execution time of a state with the same receiving message in an SMD/MARTE of the lifeline.
2. An execution time of an execution specification with only a sending message in an SD/MARTE should be greater than or equal to an execution time of a state with the same sending message in an SMD/MARTE of the lifeline.
3. An execution time of an execution specification with a receiving message and a sending message in an SD/MARTE should be equal to an execution time of a path from a state with the same receiving message to a state with the same sending message in an SMD/MARTE of the lifeline. ⌟

Rule. TD-SMD MARTE

TD-SMD MARTE rule defines that a MARTE annotation linked to a lifeline in a TD/MARTE should be equal to a MARTE annotation linked to an SMD/MARTE of the lifeline. ⌟

Rule. TD-SMD Lifeline

TD-SMD Lifeline rule defines that a lifeline in a TD/MARTE should have a corresponding SMD/MARTE. ⌟

Rule. TD-SMD Message

TD-SMD Message rule defines that a receiving message and a sending message of lifelines in a TD/MARTE should be defined in events or actions of an SMD/MARTE of the lifeline, respectively. ⌟

Rule. TD-SMD State

TD-SMD State rule defines that states of a lifeline in a TD/MARTE should be defined in an SMD/MARTE of the lifeline. ⌟

Rule. TD-SMD ExecTime

TD-SMD ExecTime rule defines that an execution time of a state of a lifeline in a TD/MARTE is the same as an execution time of a state in an SMD/MARTE of the lifeline. ⌟

Rule. SD-TD MARTE

SD-TD MARTE rule defines that MARTE annotation of a lifeline in an SD/MARTE should be the same as a MARTE annotation of the lifeline in a TD/MARTE. ⌟

Rule. SD-TD Lifeline

SD-TD Lifeline rule defines that a lifeline of an SD/MARTE and a lifeline of TD/MARTE should be the same only if MARTE annotations of an SD/MARTE and a TD/MARTE are same and a message of the lifeline in an SD/MARTE is included in messages of the lifeline in a TD/MARTE. ⌟

Rule. SD-TD ExecTime

SD-TD ExecTime rule defines that an execution time of an execution specification of a lifeline in an SD/MARTE is the same as an execution time of a state in a TD/MARTE only if the execution specification of a lifeline has a receiving and sending message, and the lifeline in a TD/MARTE also have the same receiving and sending message. ⌟

Appendix 2: SMDs/MARTE for CCS software

See Figs. 30, 31, 32, 33, 34 and 35.

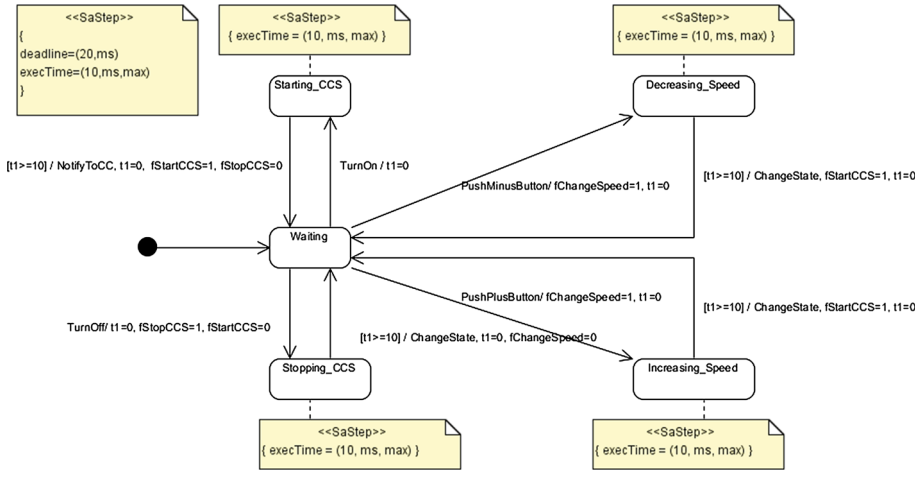


Fig. 30 SMD/MARTE for SettingController

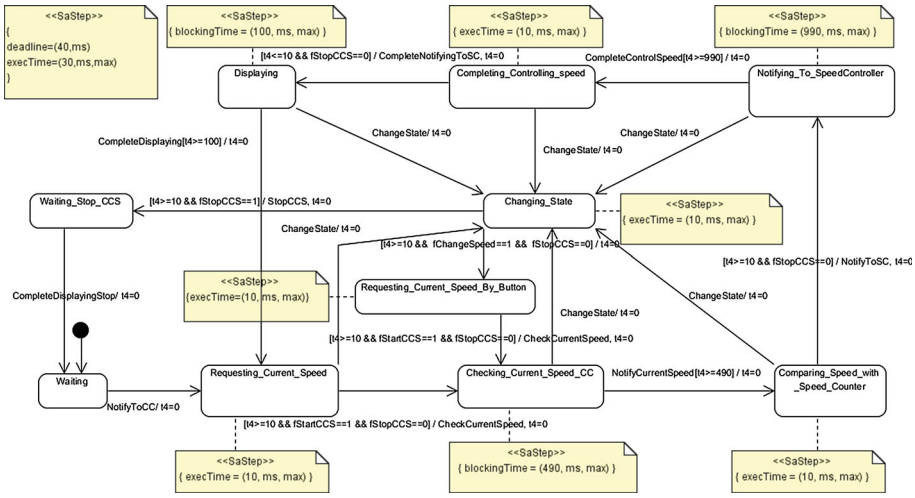


Fig. 31 SMD/MARTE for CruiseController

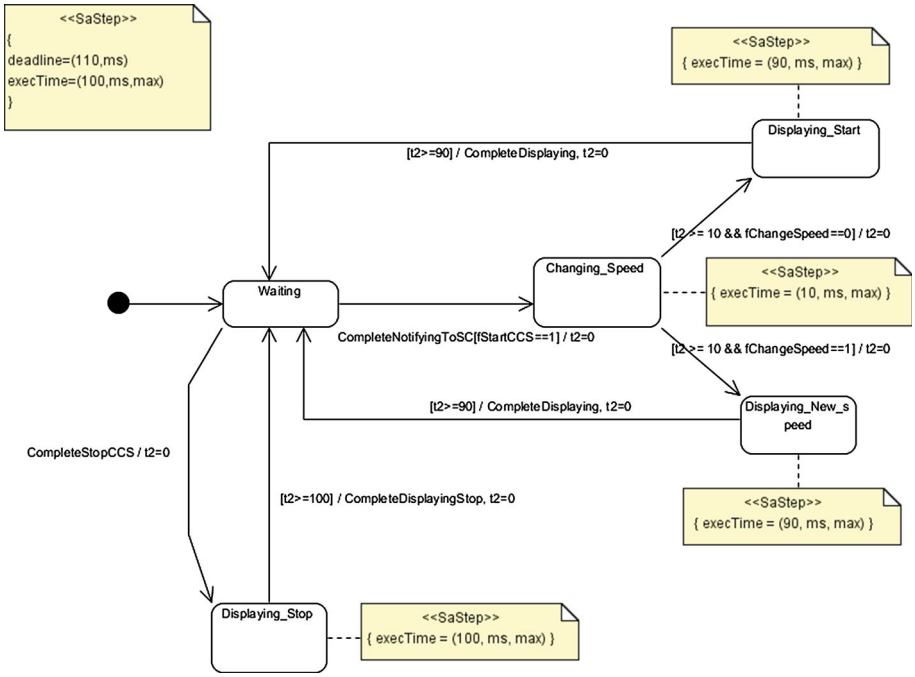


Fig. 32 SMD/MARTE for display

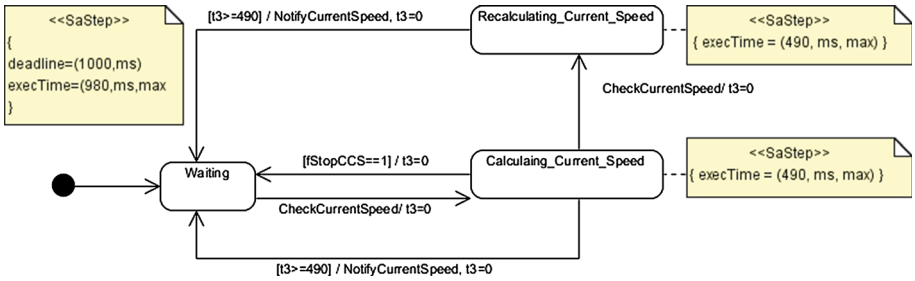


Fig. 33 SMD/MARTE for SpeedCounter

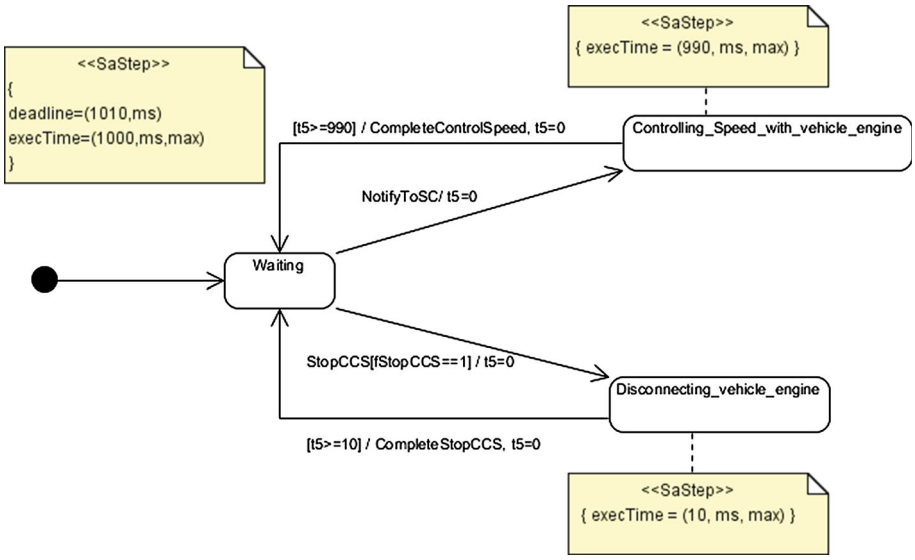


Fig. 34 SMD/MARTE for SpeedController

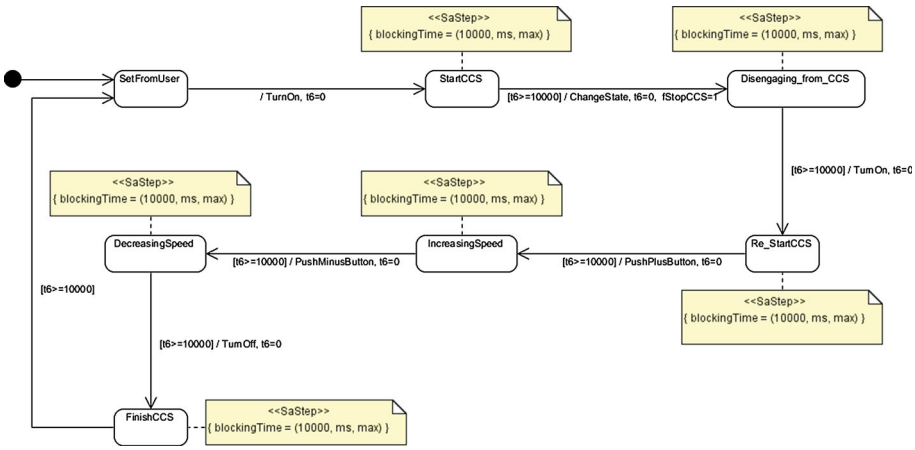


Fig. 35 SMD/MARTE for user

References

Alur, R., & Dill, D. L. (1994). A theory of timed automata. *Theoretical Computer Science*, *126*(2), 183–235.

Amnell, T., Fersman, E., Pettersson, P., Sun, H., & Yi, W. (2002). Code synthesis for timed automata. *Nordic Journal of Computing*, *9*(4), 269–300.

Amnell, T., Fersman, E., Mokrushin, L., Pettersson, P., & Yi, W. (2004). TIMES: A tool for schedulability analysis and code generation of real-time systems. In *Formal modeling and analysis of timed systems* (pp. 60–72). Berlin: Springer.

Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., et al. (2010). *Systems and software verification: Model-checking techniques and tools*. Berlin: Springer.

Chiorean, D., Paşca, M., Cârcu, A., Botiza, C., & Moldovan, S. (2004). Ensuring UML models consistency using the OCL environment. *Electronic Notes in Theoretical Computer Science*, *102*, 99–110.

Choi, J. (2015). UMCA (UML/MARTE timing Consistency Analyzer) tool. http://sites.google.com/site/jhchoi93/.

Choi, J., & Bae, D. H. (2012). An approach to constructing timing diagrams from UML/MARTE behavioral models for guidance and control unit software. In *Computer applications for database, education, and ubiquitous computing* (pp. 107–110). Berlin: Springer.

Choi, J., Shim, J., & Yim, S. (2005). The implementation and performance analysis of soft timer interrupt UML-RT model on a windows platform with real-time extension. In *Proceedings of the korea information science society fall conference* (Vol. 32, pp. 841–843)

Choi, J., Jee, E., Kim, H. J., & Bae, D. H. (2011a). A case study on timing constraints verification for a safety-critical, real-time system. In *Proceedings of the Korea computer congress (KCC)* (Vol. 38, pp. 166–169).

Choi, J., Jee, E., Kim, H. J., & Bae, D. H. (2011b). A case study on timing constraints verification for a safety-critical, time-triggered embedded software. *Journal of KIISE: Software and Applications*, *38*(12), 647–656.

Choi, J., Jee, E., & Bae, D. H. (2012a). Systematic vxworks-based code generation from timed automata model. In *Proceedings of the Korea computer congress (KCC)* (Vol. 39, pp. 138–140).

Choi, J., Jee, E., & Bae, D. H. (2012b) Toward systematic construction of timing diagrams from UML/MARTE behavioral models for time-triggered embedded software. In *2012 IEEE sixth international conference on Software Security and Reliability (SERE)* (pp. 118–127).

Choi, J., Jee, E., & Bae, D. H. (2013). Systematic generation of VxWorks-based code from timed automata models. *Journal of KIISE: Computing Practices and Letters*, *19*(2), 90–94.

Clarke, E. M., Grumberg, O., & Peled, D. (1999). *Model checking*. Cambridge: MIT Press.

Egyed, A. (2006). Instant consistency checking for the UML. In *Proceedings of the 28th international conference on Software Engineering* (pp. 381–390). ACM

Egyed, A. (2007). UML/Analyzer: A tool for the instant consistency checking of UML models. In *29th International Conference on Software Engineering, 2007. ICSE 2007* (pp. 793–796).

Egyed, A. (2011). Automatically detecting and tracking inconsistencies in software design models. *IEEE Transactions on Software Engineering*, *37*(2), 188–204.

Engels, G., Küster, J. M., Heckel, R., & Groenewegen, L. (2001). A methodology for specifying and analyzing consistency of object-oriented behavioral models. In *ACM SIGSOFT Software Engineering Notes* (Vol. 26, pp. 186–195)

Fersman, E., Pettersson, P., Yi, W. (2002). Timed automata with asynchronous processes: Schedulability and decidability. In *Tools and Algorithms for the Construction and Analysis of Systems* (pp. 67–82). Berlin: Springer

Fowler, M. (2004). *UML distilled: A brief guide to the standard object modeling language*. Boston: Addison-Wesley Professional.

Gherbi, A., & Khendek, F. (2007) Consistency of UML/SPT models. In *SDL 2007: Design for dependable systems* (pp. 203–224). Berlin: Springer.

Gogolla, M., Büttner, F., & Richters, M. (2007). Use: A UML-based specification environment for validating UML and OCL. *Science of Computer Programming*, *69*(1), 27–34.

Gomaa, H. (2001). Designing concurrent, distributed, and real-time applications with UML. In *Proceedings of the 23rd international conference on software engineering*, (pp. 737–738). IEEE Computer Society.

Gomes, L., Fernandes, J. M., & Global, I. (2010). Behavioral modeling for embedded systems and technologies: Applications for design and implementation. In*Information Science Reference*

Holzmann, G. J. (2003). *The SPIN model checker: Primer and reference manual*. Boston: Addison-Wesley Professional.

Knapp, A., Merz, S., & Rauh, C. (2002). Model checking timed UML state machines and collaborations. In *Formal techniques in real-time and fault-tolerant systems*, (pp. 395–414). Berlin: Springer

Kuster, J., & Stroop J (2001) Consistent design of embedded real-time systems with UML-RT. In *Proceedings. Fourth IEEE international symposium on object-oriented real-time distributed computing, 2001. ISORC-2001*, (pp. 31–40).

Laleau, R., & Polack, F. (2008). Using formal metamodels to check consistency of functional views in information systems specification. *Information and Software Technology*, *50*(7), 797–814.

Lavagno, L., Martin, G., & Selic, B. V. (2003). *UML for real: Design of embedded real-time systems*. Berlin: Springer.

Leveson, N. (2011). *Engineering a safer world: Systems thinking applied to safety*. Cambridge: Mit Press.

Lucas, F. J., Molina, F., & Toval, A. (2009). A systematic review of UML model consistency management. *Information and Software Technology*, *51*(12), 1631–1645.

Millett, L. I., Thomas, M., Jackson, D., et al. (2007). *Software for dependable systems: Sufficient evidence?*. Washington: National Academies Press.

Nentwich, C., Capra, L., Emmerich, W., & Finkelsteiin, A. (2002). xlinkit: A consistency checking and smart link generation service. *ACM Transactions on Internet Technology (TOIT)*, *2*(2), 151–185.

Nguyen, M. C., Jee, E., Choi, J., & Bae, D. H. (2014). Automatic construction of timing diagrams from UML/MARTE models for real-time embedded software. In *Proceedings of the 29th annual ACM symposium on applied computing*, (pp. 1140–1145).

OMG. (2005). UML profile for schedulability, performance, and time specification, version 1.1 (formal/2005-01-02) Edition. http://www.omg.org.

OMG. (2006). Object Constraint Language. version 2.0 (formal/06-05-01) Edition. http://www.omg.org.

OMG. (2011a). UML profile for MARTE: Modeling and analysis of real-time embedded systems. version 1.1 (formal/2011-06-02) Edition. http://www.omg.org

OMG. (2011b). Unified Modeling Language: Superstructure. version 2.4.1 (formal/2011-08-06) Edition. http://www.omg.org.

Papyrus . (2012). Papyrus. http://www.eclipse.org/modeling/mdt/papyrus/

Paradigm, V. (2012). Visual Paradigm for UML Communication Edition. http://www.visual-paradigm.com.

Peraldi-Frati, M. A., Blom, H., Karlsson, D., & Kuntz, S. (2012). Timing modeling with autosar-current state and future directions. In *Design, automation and test in Europe conference and exhibition (DATE), IEEE* (pp. 805–809).

Pont, M. J. (2001). *Patterns for time-triggered embedded systems*. New York: Person Edueation.

Rumbaugh, J., Jacobson, I., & Booch, G. (2004). *The unified modeling language reference manual*. Boston: Addison-Wesley Professional.

Selic, B. (1998). Using UML for modeling complex real-time systems. In *Languages, compilers, and tools for embedded systems* (pp. 250–260), Berlin: Springer.

Sgroi, M., Lavagno, L., & Sangiovanni-Vincentelli, A. (2000). Formal models for embedded system design. *IEEE Design & Test of Computers*, *17*(2), 14–27.

Sommerville, I. (2011). *Software engineering* (9th ed.). Boston: Addison Wesley.

Sourrouille, J. L., & Caplat, G. (2002). Checking UML model consistency. In *Workshop on consistency problems in UML-based software development, Blekinge Institute of Technology* (pp. 1–15).

TIMES. (2007). Uppsala university design and analysis of real-time systems team. TIMES-a tool for modeling and implementation of embedded systems. http://www.timestool.com.

UPPAAL. (2012). Uppsala University and Aalborg University. UPPAAL-a tool for verification of real-time systems. http://www.uppaal.org.

Usman, M., Nadeem, A., Kim, T. H., & Cho, E. S. (2008). A survey of consistency checking techniques for UML models. In *Advanced software engineering and its applications, 2008, ASEA 2008 IEEE*, (pp. 57–62).

Wieringa, R. J. (2003). *Design methods for reactive systems: Yourdon, statemate, and the UML*. Amsterdam: Elsevier.

Zhao, X., Long, Q., & Qiu, Z. (2006). Model checking dynamic UML consistency. In *Formal methods and software engineering*, Berlin: Springer (pp. 440–459).

Jinho Choi is a Senior Researcher in the 1st R&D Institute at Agency for Defense Development (ADD) in Republic of Korea. He received his Ph.D. degree at the Department of Computer Science in Korea Advanced Institute of Science and Technology (KAIST). His research interests include software quality, model-based embedded software design, real-time embedded system, software safety, and formal method.

Eunkyoung Jee is a Research Assistant Professor in School of Computing at KAIST in Republic of Korea. She was a Postdoctoral Researcher in the Computer and Information Science Department at the University of Pennsylvania. She received her B.S., M.S., and Ph.D. degrees in Computer Science from KAIST. Her research interest includes safety-critical software, software testing, formal method, and safety analysis.

Doo-Hwan Bae is a Professor in School of Computing at Korea Advanced Institute of Science and Technology (KAIST). He received his Ph.D. at the Department of Computer Science in the University of Florida. He currently leads many projects funded by Korean government and industry. His research interests include software safety, software testing, quality-driven software development, embedded software design, and mining software repositories.