Automatic Construction of Timing Diagrams from UML/MARTE Models for Real-Time Embedded Software

Minh Chau Nguyen  
Software Research  
Department  
ETRI, Daejeon  
Republic of Korea  
chau@etri.re.kr

Eunkyoung Jee  
Computer Science  
Department  
KAIST, Daejeon  
Republic of Korea  
ekjee@se.kaist.ac.kr

Jinho Choi  
Precision Guidance  
Technology Center  
ADD, Daejeon  
Republic of Korea  
jhchoi93@add.re.kr

Doo-Hwan Bae  
Computer Science  
Department  
KAIST, Daejeon  
Republic of Korea  
bae@se.kaist.ac.kr

ABSTRACT

Analysis of timing constraints is an essential part in developing real-time embedded software. Performing the timing analysis during the early development phases prevents timing violations and enhances software quality. In the development of real-time embedded software, UML timing diagrams can play a significant role since they can provide not only intuitive specifications for timing constraints, but also valuable information for verifying system requirements. However, as software complexity increases, modeling timing diagrams becomes difficult and costly. We propose an automated construction approach of timing diagrams from UML sequence diagrams and state machine diagrams with MARTE annotations. The proposed approach enables developers of RTES to save time required for modeling timing diagrams and prevents making mistakes in construction of timing diagrams.

Categories and Subject Descriptors

D.2.10 [Software Engineering]: Design—Methodologies

Keywords

UML, MARTE, timing diagram, state machine diagram, sequence diagram, real-time embedded software

1. INTRODUCTION

Real Time Embedded Software (RTES) plays a crucial role in most advanced technical systems such as avionics, automobile and medical equipments. It has become the main driver and facilitator for innovation [1]. The consistent information of real-time constraints such as timing requirements, resource limitations, hardware dependencies, has a substantial impact on the success of RTES development. Any inconsistencies may become a source of serious faults leading to the failure of a whole system. Timing constraints need to be identified and analyzed as soon as possible in the software development process, particularly in the design phase.

With the current trend toward model-driven development, the choice of Unified Modeling Language (UML) [2] models as a source of information at the design level is natural and practical. However, modeling RTES by using UML was a complex task because of UML’s insufficient capability of real-time specification. Fortunately, by combining with the Modeling and Analysis of Real-Time and Embedded Systems (MARTE)[3] profile that addresses specific aspects associated with real-time system modeling, a completed design model for RTES can be achieved more conveniently.

The UML timing diagram is one of the most suitable diagrams to specify RTES among multiple types of UML diagrams. They facilitate the communication between software engineers and hardware engineers due to their intuitiveness of specifications for timing constraints [4]. Through timing diagram, we can have knowledge about the interactions between different objects, as well as the changes of state on each object in certain scenarios [2]. Timing diagrams therefore have close relationship with sequence diagrams and state machine diagrams which are frequently used for specifying software behavior. The information in timing diagram is useful to generate timed test case which is much important for RTES [5].

Nevertheless, modeling RTES with timing diagrams, is not a simple task. One of the reasons is complex structure of timing diagrams in aspect of performing the system modeling. The designers need to consider not only the interactions between different objects, but also the changes of state on each object under a timing viewpoint when they perform modeling timing diagrams. Especially, as software complexity increases, specifying a large number of timing constraints requires the designers spending much time and effort. As a result, mistakes, such as inconsistent information with other diagrams are inevitable. It can be time-consuming to recognize and fix these mistakes. If timing diagrams can be generated automatically from existing information, we not only can save time required for modeling timing diagrams, but also can prevent mistakes caused by designers.

Our goal is to construct timing diagrams with MARTE annotations (TDs/MARTE) from well-formed sequence diagrams with MARTE annotations (SDs/MARTE) and state machine diagrams with MARTE annotations (SMDs/MARTE) for RTES. We propose a set of transformation rules for decomposing complex SDs/MARTE and a systematic way of combining them with information in SMDs/MARTE to syn-
thesize the corresponding TDs/MARTE models. An automated tool based on our proposed approach is provided. To the best of our knowledge, it is the first complete tool allowing an automatic construction of TD/MARTE.

The rest of our paper is structured as follows. Section 2 explains how SMDs/MARTE, SDs/MARTE and TDs/MARTE can be used in modeling RTES. Section 3 describes the TDs/MARTE construction approach. Section 4 presents the related work. Section 5 concludes the paper.

2. BEHAVIORAL MODELING FOR REAL-TIME EMBEDDED SOFTWARE

Given well-formed SDs/MARTE and SMDs/MARTE, our approach generates well-formed TDs/MARTE. This section clarifies what is well-formed UML/MARTE diagrams to specify RTES. We demonstrate our approach using a pacemaker software example throughout the paper. We assume that timing behaviors are performed under synchrony hypothesis. It means that message exchange time is not specified in UML/MARTE behavioral models. If a message exchange time needs to be considered, the message exchange time can be added to an execution time of an object.

2.1 Sequence Diagrams with MARTE Annotations

An SD/MARTE describes interactions among objects. We distinguish two types of SD/MARTEs: basic SD/MARTE (bSD/MARTE) and complex SD/MARTE (cSD/MARTE). A bSD/MARTE is used for specifying one specific scenario while a cSD/MARTE can be used for specifying multiple scenarios with extra information based on combined fragments. Figure 1 shows an example bSD/MARTE which represents a normal operation without ventricular signal scenario in the pacemaker system. The SensingController sends a message NoVSensingSignal to the PacemakerController at 1600 ms. After receiving this message, PacemakerController executes during 10 ms and sends a message VPacingSignal to PacemakerController at 1610 ms. PacemakerController works for 5 ms for generating VPulse and sends it to the heart. As shown in Figure 1, time observations, execution specifications and MARTE annotations are used to specify timing constraints of RTES. An example cSD/MARTE in Figure 2 represents the normal operation scenario of the pacemaker system. In this example, using a combined fragment with an Alt operator enables the compact and convenient specification of two alternative scenarios: normal operation without ventricular signal and normal operation with ventricular signal.

According to [7], 13 types of operators in UML 2.4.1 can be grouped into three categories: compacting operators including Alternative(Alt), Option(Opt), Break and Loop; ordering operators including Parallel(Par), Strict/Weak Sequencing and Critical; categorizing operators including Negative(Neg), Assertion(Assert), Ignore and Consider. We focus on four representative operators: Alt, Loop, Par and Neg. The others can be expressed by one of the selected operators (e.g., Opt can be considered as Alt with only one operand), can be skipped by our modeling guideline (i.e. time specification of every message removes the ambiguity of message’s order) or are not often used in the RTES.

2.2 State Machine Diagrams with MARTE Annotations

An SMD/MARTE specifies the overall behavior of an object focusing on state changes of the object. SMDs/MARTE specify the overall behaviors of RTES. Figure 3 shows an example SMD/MARTE which specifies the PacemakerController with the variable t representing a local clock. After sending a message EnableSensing from the Waiting state, the PacemakerController moves to the Waiting_VSense_VVI state to wait for the ventricular signal. Once the PacemakerController receives VSensingSignal after 800ms, it checks whether the working state of the pacemaker system is Magnet Test or not, in the Checking_Magnet_Test state, to decide whether it should send a message VPacingSignal and return to the Waiting state. If at the Waiting_VSense_VVI state, there is a message NoVSensingSignal, depending on the current state of system (PaceNow or Permanent), the corresponding VPacingSignal is generated. MARTE annotations linked to states are used to specify duration (e.g., worst case execution times) of each state.

2.3 Timing Diagrams with MARTE Annotations

A TD/MARTE specifies the interaction between objects over time and the changes of states in each object for a specific scenario. Figure 4 shows an example TD/MARTE which represents a timing scenario for normal operation without ventricular signal of the pacemaker system. Similar to the SD/MARTE in Figure 1, this diagram describes how SensingController, PacemakerController and PacingController interact with each other through messages with timing specifications. The TD/MARTE also shows partial information from SMD/MARTE in Figure 3 when it describes changes of state from Waiting_VSense_VVI to Generating_VPace_VVI.
with corresponding durations. The main advantage of TD/MARTE is that it can provide a viewpoint on timing constraints of the whole system for a specific scenario. For example, in the normal operation without ventricular signal scenario, from TD/MARTE we can know that during the time from 1600 ms to 1605 ms, the SensingController must be in the Waiting_Enable_Sensing state, the PacemakerController must be in the Checking_PaceNow_State state, and the PacingController must be in the Waiting state.

3. AUTOMATIC CONSTRUCTION OF TDS/MARTE

Given a cSD/MARTE and a set of SMDs/MARTE as inputs, a set of bSDs/MARTE is generated in the automatic construction process. Inputs are well-formed models which do not contain inconsistencies [8]. The process consists of two steps: in the first step, the cSD/MARTE is decomposed into a set of bSDs/MARTE by the SD/MARTE Decomposer; in the second step, each bSD/MARTE and a set of SMDs/MARTE are combined to generate a set of TDS/MARTE by the TD/MARTE Generator.

3.1 SD/MARTE Decomposer

Before explaining the details of the decomposition process, we explain how to classify cSD/MARTE first. Based on the relative position of combined fragments in diagram, cSD/MARTE can be categorized into three types. cSD/MARTE is called multiple combined fragment SD/MARTE if it contains more than one separate combined fragments. Otherwise, it is called single combined fragment SD/MARTE. In Figure 7 sd a is a multiple combined fragment SD/MARTE and sd a_2, sd a_4 are single combined fragment SDs/MARTE. Single combined fragment SDs/MARTE are categorized into nested and non-nested ones based on the number of combined fragments inside. Nested single combined fragment SD/MARTE contains a combined fragment including combined fragments inside as well as messages, while non-nested single combined fragment SD/MARTE just contains a combined fragment with only messages inside.
fragment indicates that the behavior of system will be repeated some number of times with a period value. For obtaining a complete set of behaviors, each loop needs specifically generate messages with the time observations updated by adding the appropriate period time specified in the MARTE annotation of the corresponding objects. Hence, applying Loop-Pattern results in an SD/MARTE containing all messages of each loop. The identifier of this SD/MARTE is named by the original name of input SD/MARTE with Loop and number of loops as a suffix.

Par represents a parallel merge between the behaviors represented in the operands. The event occurrences of the different operands therefore can be interleaved to each other. Because in RTES each message is specified at a specific time observation, an SD/MARTE containing all messages inside all operands can be generated according to the order of time observations. The identifier of this SD/MARTE is named by the original name of input SD/MARTE with Par as a suffix.

Neg defines a forbidden behavior of the system. For Neg-Pattern, an SD/MARTE containing all messages inside the combined fragment will be generated with the identifier named by the original name of the input SD/MARTE and Neg as a suffix. The Neg suffix is useful to notify designers about the forbidden attribute of the diagram.

**Pattern 5, 6, 7, 8:** Alt-Pattern, Loop-Pattern, Par-Pattern and Neg-Pattern for Nested Single Combined Fragment SD/MARTE

The principle of applying these patterns is similar to the corresponding patterns for the non-nested case. The difference is that in the identifiers of generated SDs/MARTE, the current level of the combined fragment is added as a suffix.

**Pattern 9:** Pattern for Multiple Combined Fragment SD/MARTE

This pattern will be applied for SD/MARTE with multiple combined fragment. In the first step, for each combined fragment inside, an SD/MARTE is generated. Similarly, for each consecutive message segment, an SD/MARTE containing that message segment is also generated. The identifiers of these SDs/MARTE are named by the original name of the input SD/MARTE with the order of the combined fragment or message segment as a suffix. For example, in Figure 7, four SDs/MARTE a1, a2, a3, and a4 are generated from the corresponding combined fragment and message segment. In the second step, after all output SDs/MARTE of the first step are decomposed to partial bSDs/MARTE, the merging procedure is performed based on the combination principle. For example, if there are four SDs/MARTE a1, a2, a3, a4 and each one is decomposed into two SDs/MARTE, the merging procedure will result in $2 \times 2 \times 2 \times 2 = 16$ SDs/MARTE.

**3.2 TD/MARTE Generator**

Figure 9 shows how a set of TDs/MARTE can be generated from SD/MARTE and SMDs/MARTE. At first, timing ruler is defined based on values of time observations in SD/MARTE (line 4). After that, lifelines in SD/MARTE with MARTE annotations are mapped into lifelines of TD/MARTE (line 5-6) and messages with time occurrence specifications are transformed into TD/MARTE (line 7). Finally, the states of each lifeline with corresponding durations in TD/MARTE will be specified (from the line 9). In this step, if there is no corresponding SMD/MARTE for an object, it is assumed that a default state machine diagram including only active and inactive states exists for the objects. We initialize the interval for this state and add it to the list of states in TD/MARTE (line 15-19). Otherwise, for each execution specification in SD/MARTE, we need to decide what the corresponding states in SMDs/MARTE are.

Figure 6: An example Alt for Non-Nested case.

Figure 7: An example pattern 9 for Multiple case.

Figure 8: An example of merging step.
For example, in Figure 1, after receiving a message from the SensingController and before sending a message to the PacemakerController, the PacemakerController executes 10 ms to perform several tasks, such as checking working state, preparing for a signal. This results in the several changes of states in the PacemakerController. In our algorithm, the receiving message (line 22) and sending message (line 23) on each execution specification in SD/MARTE are used to find the executing paths in SMD/MARTE, such that the total executing time of states is equal to the executing time in MARTE of the execution specification (line 24). In case there are more than one executing paths found, each path may lead to a different TD/MARTE, therefore more than one TDs/MARTE can be generated (line 25-34). For each state in the executing path, the starting time and ending time are calculated based on its previous states and its executing time or blocking time (line 29-30).

In order to generate TD/MARTE for the input bSD/MARTE in Figure 1, firstly based on the time observations set in SD/MARTE, time units including 1600, 1610 and 1615 are generated. After that, lifelines h: Heart, Scrl: SensingController, PMCtrl: PacemakerController and PCtrl: PacingController with corresponding MARTE annotations will be created in TD/MARTE. Thirdly, messages NoVSensingSignal, VPacingSignal and VPulse with timing occurrence specifications are mapped into TD/MARTE. At the final step, a set of SMDs/MARTE with executing specifications of each lifeline in SD/MARTE are used for deciding specific states and durations. For example, in the PacemakerController shown in Figure 3, based on the sending message VPacingSignal and receiving message NoVSensingSignal, we can find an executing path (Waiting_VSense_VVI, Checking_Permanent_State_VVI, Ready_VPace_VVI, Generating_VPace_VVI). Since the total executing time of this path (0 + 5 + 5 + 0 = 10 ms) is equal to the executing time of the execution specification in SD/MARTE, the states with executing time in MARTE in this path are added to TD/MARTE. However, because the executing path (Waiting_VSense_VVI, Checking_Pace_Now_State, Ready_VPace_Now, Generating_VPace_VVI) also satisfies the time condition, two TDs/MARTE are generated. Figure 4 shows one of the generated TDs/MARTE.

3.3 The TMC tool

TMC (TD/MARTE Constructor) is an automatic tool for constructing TDs/MARTE from SDs/MARTE and SMDs/MARTE. We have developed this tool as an Eclipse plugin, so that it can be combined easily with other plugins to make the process of modeling and analyzing become more convenient. Papyrus [9] and Visual Paradigm [10] plugins in Eclipse are recommended to be used with TMC. Here, we use Papyrus for modeling SD/MARTE and Visual Paradigm for modeling SMD/MARTE and editing TD/MARTE.

In TMC's architecture, there are four main components: XML Parser, Consistency Checker, TD/MARTE Transformer and XML Writer. At the beginning, designers model the software system using available modeling tools and then provide this set of model in xml format files as inputs to our TMC. The XML Parser parses the information in these XML input files to let the Consistency Checker perform consistency checking process based on the set of rules in [8], so that the models are ensured as well-formed input conforming to our modeling guidelines. After that, the TD/MARTE Transformer generates the TDs/MARTE which can be shown and edited intuitively, or exported as portable files for different purposes by XML Writer.

Figure 10 depicts the GUI of TMC. It comprises four panels. The left panel is responsible for managing the set of TDs/MARTE...
of TMC projects including input diagrams and output diagrams. The middle panel is for displaying and editing the diagrams in textual or graphical format. The bottom panel shows the consistency checking results. The right panel informs error messages.

4. RELATED WORK

Despite the advantages of UML timing diagram, the number of studies on this diagram is still limited [11][12][13][14][15]. Pulka et al. [11] utilized the usefulness of timing diagrams to show simulation results for timing verification in system on chip. With the same purpose, Binh et al. [12] extracted timing constraints from timing diagrams to verify the conformity between the implementation and its requirement specification. Dynek et al. [13] introduced an approach to estimate system workload with time characteristics by using timing diagrams. They found that the potential bottlenecks of a system could be checked based on timing diagrams. While these works use existing timing diagrams to facilitate related analysis and verification process, our approach can support these process by generating TDs/MARTE automatically.

The intuitiveness of timing diagram is also utilized to improve the weakness of other modeling approaches. Joochim et al. [14] introduced an approach to extend the graphical modeling capability for formal modeling Event-B in real-time system domain by timing diagrams. In their approach, timing diagrams are translated directly into Event-B and UML-B by a set of ATL-based rules. Similarly, Lee et al. [15] took advantages of timing diagrams to reduce the complexity of Simulink-based approach in modeling software specification. Nevertheless, timing diagram in these works had been extended with many extra features which can be a barrier preventing being applied widely. Our approach uses standard SMDs/MARTE and SDs/MARTE to generate standard TDs/MARTE, so the designers can use these generated diagrams without worries about the validity.

Most of the previous works focused on transforming timing diagrams into another forms for particular purposes. To the best of our knowledge, our previous work [16] is the first attempt to generate TD/MARTE from the existing SD/MARTE and SMD/MARTE models. In our work, we has extended and improved that approach in many aspects. Firstly, we consider the combined fragment features in SD/MARTE which are much significant for current complex system. Secondly, our SMD/MARTE is specified-level SMD/MARTE which contains many specific states and transitions, while in [16], SMD/MARTE is task-level SMD/MARTE which is more suitable for specifying small system with a limited number of pre-defined states and transitions. Thirdly, we provide an automated tool which allows to generate TDs/MARTE efficiently.

5. CONCLUSION

We proposed the approach for constructing TDs/MARTE from UML/MARTE behavioral models for RTES. By using the proposed approach, consistent timing diagrams can be generated from SDs/MARTE and SMDs/MARTE models systematically. The proposed approach can handle complex SD/MARTE and SMD/MARTE, thereby reducing significantly time and effort to construct TDs/MARTE models. Moreover, we provided the automated tool TMC to support our approach.

We have a plan to optimize our approach and extend it for supporting reverse transformation from TD/MARTE to SD/MARTE and SMD/MARTE. The software system evolves throughout the software development life cycle, therefore there is often a demand of modifying the generated TDs/MARTE. The reverse transformation will help the designers to synchronize the changes in TDs/MARTE to the corresponding SDs/MARTE and SMDs/MARTE, so that the consistency between models is maintained automatically.

Acknowledgment

This work was supported by the IT R&D program of MSIP/KEIT. [10044457, Development of Autonomous Intelligent Collaboration Framework for Knowledge Bases and Smart Devices]

6. REFERENCES