

Received March 1, 2019, accepted March 12, 2019, date of publication March 18, 2019, date of current version April 8, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2906011

Modeling and Timing Analysis for Microkernel-Based Real-Time Embedded System

RONGFEI XU¹, LI ZHANG¹, AND NING GE²

¹School of Computer Science and Engineering, Beihang University, Beijing 100083, China

²School of Software, Beihang University, Beijing 100083, China

Corresponding author: Ning Ge (gening@buaa.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 61672078 and Grant 61732019.

ABSTRACT Currently, more and more application-specific operating systems (ASOSs) are applied in the domain of real-time embedded systems (RTESs). With the development of a microkernel technique, the ASOS is usually customized based on a microkernel using the configurable policy. Evaluating the timing requirements of an RTES based on the ASOS is helpful to guide the designer toward the choice of the most appropriate configuration. Modeling and analyzing the time requirements for such a system in the early design stage are essential to avoid redesigning or recoding the system at a later stage. However, the existing works are insufficient to support the modeling for both the specific domain of the microkernel-based RTES and the variability of the configurable policy, as well as a general analysis for the various configurations. To solve these problems, this paper presents a modeling and timing analysis framework (MTAF) for the microkernel-based RTES. Our main contributions are twofold: 1) proposing a domain-specific language (DSL) for the timing analysis modeling of the microkernel-based RTES; then, we define and implement this DSL as a UML profile and (2) proposing a static timing analysis approach for the RTES design modeled by the DSL, where a timing analysis tree and uniform execution rules are defined to analyze the variability in a general way. In the case study, we take the scheduling policy as an example to show the use of our framework on a real-life robot controller system.

INDEX TERMS Real-time embedded system, microkernel-based RTOS, application-specific operating system, timing analysis modeling, timing analysis tree.

I. INTRODUCTION

Real-time embedded systems (RTES) have been widely applied in various safety-critical fields, such as automotive, medical, military, aerospace, etc. For RTES, guaranteeing the correctness of timing requirements is a key issue. Especially for the hard real-time systems, the response time of tasks in system must be within the deadline. If a design can not meet the timing requirements in the early design, the error might propagate to the final code, and possibly cause hazards; besides, if the error is detected in the later stages of the development, the cost of re-design is extremely high. Usually, a real-time operating system (RTOS) is used to manage the tasks in an RTES, and directly impacts the timing performance of the system. RTES in various domains may suffer from the general-purpose OS due to their specific characteristics. Accordingly, many works were aimed at the

application-specific operating systems (ASOS) to enhance the performance for a certain application. Nowadays, more and more ASOS are designed based on a microkernel, such as QNX, Integrity, and FreeRTOS. A microkernel [1] is a minimalistic kernel that contains the near-minimum amount of functions and features required to implement an OS, and allows the users to tailor the OS with respect to the functions needed by an OS. It adopts the “separation of mechanism and policy” principle, where the policy runs in the user space and is configurable to build arbitrary OS services. Since many RTES are safety-critical, most of them are based on the uniprocessor with basic OS functions to ensure the safety in practical use [2]. Hence, in this work, we are interested in the modeling and analyzing of the timing requirements for such RTES that is implemented on the configurable ASOS by extending a microkernel with the basic OS functions on a uniprocessor.

In the microkernel-based RTES context, the flexible policies (called variability in this paper) are key configurations

The associate editor coordinating the review of this manuscript and approving it for publication was Rashid Mehmood.

for the ASOS, and crosscut the function of tasks. Hence, modeling the microkernel-based RTES for the timing analysis is concerned with the modeling approaches for the system and for the variability. (1) For modeling the system, AADL [3], UML [4], and MARTE [5] are the commonly applied solutions. As a UML profile, MARTE is a standard for modeling and analyzing the RTES. MARTE covers a large scope of modeling elements; consequently, it is difficult for software engineers to identify where they should start and which elements should be used to achieve the design. In practice, only a subset of MARTE elements is required for the domain-specific purpose. There exist some works aiming at the modeling of the microkernel-based RTES, but they focus on the architectural modeling without considering the timing requirements. (2) For modeling the variability, existing works [6], [7] are limited to configuring partial aspects of the RTOS, e.g., software resources, or scheduling parameters, or supporting the predefined configurations. Therefore, it is insufficient to model the complete variability on the essential functions of the microkernel, which covers the scheduling, the IPC (inter-process communication) and the resource access. To conclude, it lacks a domain-specific modeling method that supports both the system modeling and the variability modeling.

In the model-driven development (MDD) context, timing analysis for the RTES model is mainly based on two approaches: dynamic approach and static approach. The dynamic approaches, which include simulation-based approach and model checking, suffer from the efficiency problem. Specifically, the simulation-based approaches do not suit to assess the variability. This is because that each alternative policy requires to generate a policy-dedicated simulation model, which is not feasible for a general-purpose simulator. The model checking-based approaches have the same problem as they also need to concern the policy-dedicated rules in the task model [8]. For the static analysis approach, it includes three main classes: structure-based, path-based, and techniques using implicit path enumeration (IPET). Both the path-based approach [9] and the IPET [10] are limited to consider the OS in the execution process of tasks. The structure-based approach analyzes the time by traversal of the syntax tree of tasks in consideration of the OS [11]. However, the structure-based approach still does not support the analysis of variability in the tree. Therefore, a method for analyzing the timing requirements under the variability is needed in the microkernel-based RTES.

Based on our previous work [12], in this paper we propose a modeling and timing analysis framework (MTAF) for the microkernel-based RTES, which is used to assess the configured ASOS at the early design stage. The scope of application of our framework is the hard real-time embedded system with aperiodic tasks, which is implemented on the uniprocessor with basic OS functions of scheduling, IPC and resource access (without cache and pipeline).

- We define a domain-specific language (DSL) concerning the designs of the system and the variability. In the DSL, we follow the aspect-oriented modeling (AOM) methodology [13] to handle the variability in the target system, which is flexible to model the variability on a base model. Then, we define and implement this DSL as a UML profile.
- In order to perform the timing analysis of the design model with variability uniformly, we propose a static analysis method relying on a timing analysis tree (called ETAT), where the execution rules for refining the variability are configured in a uniform way.

Finally, we show the experimental results by applying our framework to modeling and analyzing a real-life robot controller system.

The paper is organized as follows: Sect. II reviews related works; Sect. III introduces the background and overview of our approach; Sect. IV describes the domain-specific modeling for the microkernel-based RTES; Sect. V proposes the timing analysis approach; Sect. VI evaluates our approach on a real-life case; and Sect. VII gives some concluding remarks and perspectives.

II. RELATED WORKS

A. MODELING FOR THE RTES

Currently, many works have been proposed for the modeling of RTES. MARTE [5] is a large-scope standard and caters the typical needs for the modeling and the analysis of RTES. There exist some works for customizing MARTE to model and analyze the specific systems. The work [14] proposed a methodology for the environment modeling based on UML/MARTE, and provided an extension to the standard UML class diagram and state machine notations. The work [15] proposed a MARTE-based methodology named Optimum to support the schedulability analysis for UML designs. Besides, AUTOSAR [6] and EAST-ADL [16] are domain-specific modeling languages, which are used for the automotive software. However, these works do not concern the microkernel architecture. CAMKES [17] provided an architecture for modeling the componentized microkernel-based systems for the domain of vehicular applications; however, it did not focus on the timing requirements.

The works on modeling of the configurable RTOS have been proposed. AUTOSAR [6] concerned the software resources in the configuration of a OS. The work [7] proposed a strategy to manage the configuration of scheduling parameters for the real-time component-based applications. The work [18] proposed an approach to modeling the details of the target execution platform, where all platform-specific implementation choices were described explicitly by a set of variation points. To tackle the mismatching problem between the software model and the RTOS model, a re-configurable middleware was proposed to explicitly describe the software assumptions in the design [19]. The above works focus on configuring some one aspect of a RTOS; or focus on the

refinement in the implementation of a RTOS. Therefore, the existing works are not sufficient to handle the flexible configuration on the microkernel. Besides, the variability modeling languages such as Kconfig and the Component Definition Language [20] are proposed to model the variability of the open-source RTOS, they aim to manage the vast configuration space of the variability and do not concern the structure of a RTOS.

B. TIMING ANALYSIS OF THE RTES

1) DYNAMIC APPROACH

The work [21] performed the schedulability analysis for a UML-MARTE model by using the MAST analysis models. The works [22] mapped the MARTE model to the SymTA/S model for timing analysis based on the symbolic simulation. The work [23] proposed a simulation-based timing analysis approach, which depended on a more complicated system model that described the execution control flow at the code level. The work [24] was designed for checking the task temporal constraints of a real-time application. The simulation-based methods need the model transformation or a more detailed system model; therefore, it is not well suited for the timing analysis under the variability in this paper. The reason is that each alternative of the variability needs to realize a transformation or a refinement, which is trivial at the early design stage.

Other works addressed the timing analysis using the model checking based on Timed Automata [25] or Time Petri Nets [26]. UPPAAL [27] is widely used for the automatic verification of the safety and bounded liveness properties of the real-time systems that are modeled as the timed automata. TIMES [28] is designed for the symbolic schedulability analysis for the real-time systems under the predictable behaviors. These tools can result in a state-space explosion issue, which makes such exhaustive analysis infeasible in practice. To deal with the issue, the work [29] proposed an approximate response time analysis approach based on the real-time task graphs; the work [30] limited the formal verification to the level of isolated components; the work [31] proposed an approximate timing analysis framework for CRTES. The improvement works make the model checking method capable of being used in RTES. For example, the work [32] presented a flexible analysis method for the worst-case execution time (WCET) using UML-MARTE Model Checker, which was aimed at detecting the wrong software designs and refining the correct ones with respect to the WCET; The work [33] mapped the UML activity diagram into the priority time petri net (PTPN) to enhance the formal schedulability test; The work [34] mapped the workload model of the real-time systems into a Petri Nets to perform the P-invariant method to generate all transactions. However, the model checking method needs to specify the policy-dedicated rules throughout the task model for the case in this paper, which makes it inflexible for analyzing the variability.

2) STATIC APPROACH

The static analysis approach analyzes the execution time of a task based on its control flow and its timing information. There are three main classes of static analysis approaches: structure-based, path-based, and techniques using implicit path enumeration (IPET). For the path-based approach [9], the execution time is determined by analyzing the paths in the task. For the IPET [10], the control flow and the basic-block execution time are combined into the constraints to analyze the execution time. The above two approaches are limited to considering the OS functions in the execution process of a task.

For the structure-based approach [35], the execution time is analyzed in a bottom-up traversal of the syntax tree of the task. Based on the syntax tree, many extended approaches appear. For example, the work [36] proposed a scope-tree, where an expression stating the maximum execution frequency of the function and some variable declarations were associated. The work [37] proposed a task tree, which encoded the hierarchical decomposition of a task into sub-tasks, as well as the synchronization constraints between them. The work [38] proposed an execution flow tree to represent the functional blocks and the control flows in a task. Besides, a timing analysis tree consisting of sub-tasks [39] or functions [40] was proposed to analyze the WCET for each node in the tree. The above trees take the functions or the subtasks within a task as the nodes, the executions or the interactions of the nodes are concerned from various aspects, such as WCET, synchronization or instruction cache locking. Besides, the task trees [11] consisting of a global set of tasks as its nodes were proposed to analyze the scheduling of these tasks.

To conclude, the structure-based approach is convenient for analyzing the execution time of a task considering the function of OS. However, the existing works only concern some one function of the OS, such as synchronization or scheduling, it is insufficient to analyze the ASOS proposed in this paper. Besides, there is no general method to analyze the various realizations of the OS function. For example, the work [37] specifically proposed three analysis methods for the three instruction cache locking strategies respectively: static locking, semi-dynamic locking and dynamic locking. Therefore, the structure-based approach needs to be improved to analyze the variability in ASOS.

III. BACKGROUND AND OVERVIEW

A. MICROKERNEL-BASED ARCHITECTURE

In the traditional OS based on a monolithic kernel, all of the OS functions are placed in the kernel mode, which makes the OS invariant and is not suitable for the ASOS. In contrast, the OS based on a microkernel only retains the basic functions in the kernel mode, and removes other functions (such as device drivers, file systems) to the user mode. The functions of user mode in the microkernel-based OS are realized by the policy of the basic functions or additional services.

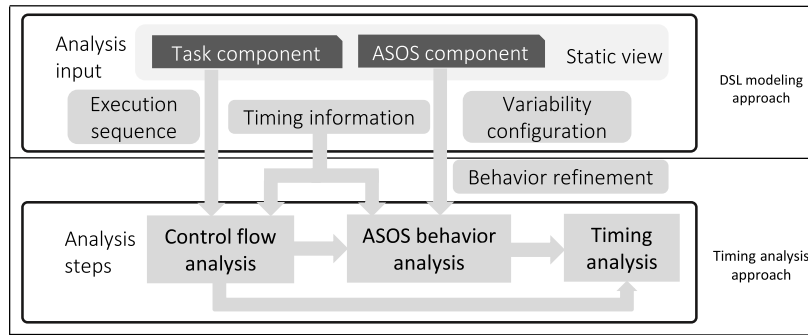


FIGURE 1. Process of timing analysis.

The services on the microkernel-based OS are tailorable and vary with different ASOS, while the basic functions are general. In this paper, we concern the timing analysis of the configurable policies for the basic functions in ASOS, i.e., scheduling, inter-process communication(IPC), and resource access.

B. OVERVIEW OF OUR METHODOLOGY

This paper deals with the timing analysis of the RTES with aperiodic tasks under a given application scenario, to assess the variability in ASOS at the early design stage, i.e., whether the response time of tasks under the variability can meet their deadline. The analysis process is shown in Fig. 1.

As our work aims at evaluating the configuration of the variability, we thus propose to study the component in the RTES from a static view to represent the configuration, which includes the task component and the ASOS component. To obtain the control flow, the execution sequence needs to be specified for the task. The control flow is influenced by the ASOS behavior, which varies with the different configuration of the variability. Here, we propose to refine such behavior at the analysis stage, and only the static components of the ASOS are specified at the early design stage. Besides, the timing information needs to be annotated in the task component and the ASOS component, i.e., the execution time. In the design stage, the worst-case execution time (WCET) of functions in the task is set (conservatively) by a designer. For a given microkernel, WCET of the basic system calls in ASOS is calculated by the existing tool. Our work is to analyze the response time of tasks in the worst case (i.e., worst case response time, WCRT) for the design under a given microkernel. If the response time in the worst case meets the timing requirements of a system, then the design provides a bound to guarantee the time for tasks. Therefore, we first propose a modeling approach for this specific domain.

Given a configuration of the variability, the design model is analyzed by the following steps: first, the control flow for every task is analyzed, then the ASOS behavior is analyzed with the refinement of the behavior. Thus, the execution time of the task is analyzed by combining the control flow and the ASOS behavior. Then, we specifically propose a timing analysis approach for the design model.

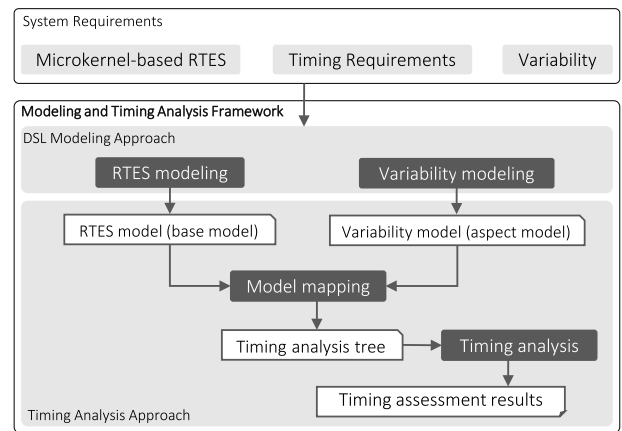


FIGURE 2. Framework of the approach.

The DSL modeling approach and the timing analysis approach compose a modeling and timing analysis framework (MTAF) as shown in Fig. 2. The input of our approach is the system requirements, which include three parts: the microkernel-based RTES, the timing requirements, and the variability. The variability covers three essential functions of the microkernel: the task scheduling, the inter-process communication (IPC), and the resource access. With respect to the above requirements, the RTES is modeled as a base model concerning the system requirements and an aspect model concerning the variability requirements. The base model and the aspect model compose the timing analysis model for the RTES. Then, the timing analysis model is mapped to a timing analysis tree to analyze the response time of a task, which is used to assess the variability.

IV. DOMAIN-SPECIFIC MODELING FOR THE MICROKERNEL-BASED RTES

This section concerns the domain-specific modeling approach for the microkernel-based RTES. First, a DSL is described in Sect. IV-A to identify the basic components used in the timing analysis for the target systems. The semantics of these components and their relationships are explained with the aid of metamodels. After that, we define and implement a UML profile for the DSL in Sect. IV-B. As the DSL is

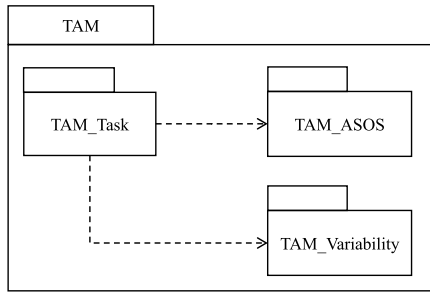


FIGURE 3. Structure of TAM domain model.

dedicated to the timing analysis, we thus name this DSL as Timing Analysis Modeling (TAM in short).

A. TIMING ANALYSIS MODELING (TAM) FOR RTES

The DSL of TAM includes the RTES modeling and the variability modeling, and has many facets that are grouped in individual packages. The overall package structure is shown in Fig. 3. As is known that a real-time embedded application is usually designed as a set of tasks managed by a RTOS [41]. Accordingly, the RTES modeling in TAM includes the task package (TAM_Task) and the ASOS package (TAM_ASOS). The package of TAM_variability is defined to model the variability of the configurable policies in ASOS. The purpose and contents of each package are described in Sect. IV-A.1, Sect. IV-A.2 and Sect. IV-A.3 respectively.

1) TAM_ASOS PACKAGE

The TAM_ASOS package contains the components related to the configured ASOS in RTES, as shown in Fig. 4. The microkernel-based ASOS is composed of kernel mechanisms and configured policies, as well as a set of basic system calls [42]. The TAM_Variability package is used to model the variability of the configurable policies, which will be introduced in Sect. IV-A.3. Besides, the software resources and the hardware are also concerned within the ASOS.

The mechanism and the policy are explicitly modeled in the ASOS. The microkernel-based ASOS includes three

types of mechanisms that cover the essential functions of the microkernel, including the task scheduling, the inter-process communication (IPC), and the resource access. Each type of mechanism can be extended by using a set of alternative configurable policies, i.e., the variability. The configured policy is realized depending on its corresponding mechanism.

Both the software resource and the basic system call in a system are used to implement the mechanism or the policy in ASOS. The software resource includes four classes: synchronization resource, concurrency resource, communication resource and computing resource. Each basic system call realizes an independent function, such as task suspend, task resume. As the ASOS is implemented on a hardware, the WCET of a basic system call depends on the hardware. Specifically, the processor in hardware effects the WCET [43], so the processor type is parameterized in our modeling approach. Given a specific processor, WCET of the basic system call is calculated by the existing tool called Chronos [44]. After the WCET is calculated, the value of the WCET is used to annotate the ASOS model.

2) TAM_TASK PACKAGE

The TAM_Task package contains the components to compose a task in RTES, as shown in Fig. 5. A task consists of a sequence of functional blocks and system calls [45]; therefore, Task, FunctionalBlock and SystemCall are respectively defined in TAM_Task.

All tasks in RTES are managed by the ASOS. Each task is assigned a unique priority, which is used to schedule the tasks. During the scheduling, four basic states (i.e., running, ready, blocked and suspended) are used to describe the execution state of a task. Once released, the task is enqueued to the ready queue and waits to run. Each task is required to conform to its deadline, which indicates the maximal time bound on the completion of this task; therefore, WCRT of a task shall meet its deadline.

A task can be split into unbreakable functional blocks, each of which is used to realize an independent function and

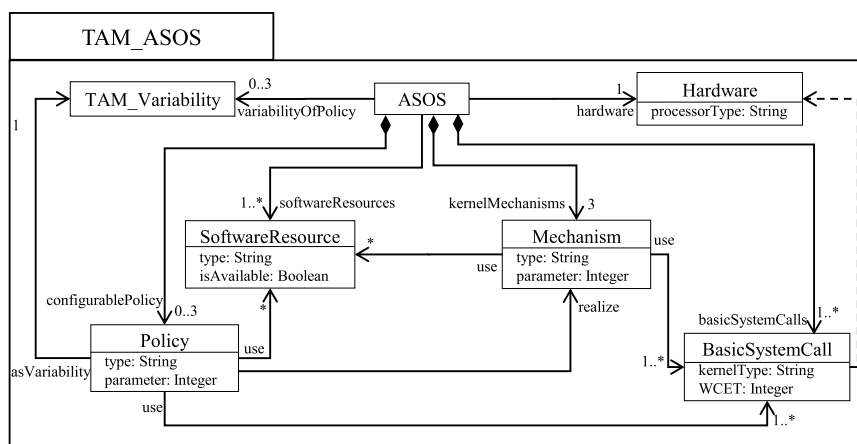


FIGURE 4. ASOS domain model.

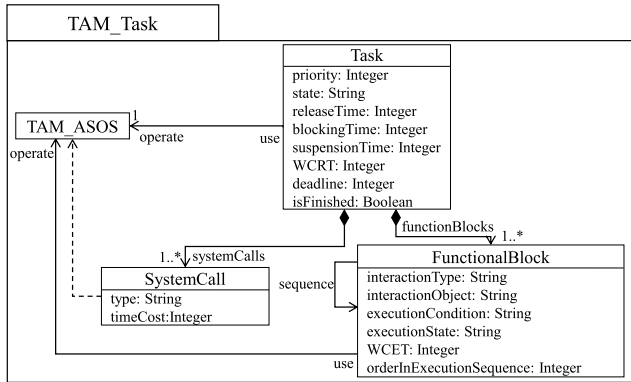


FIGURE 5. Task domain model.

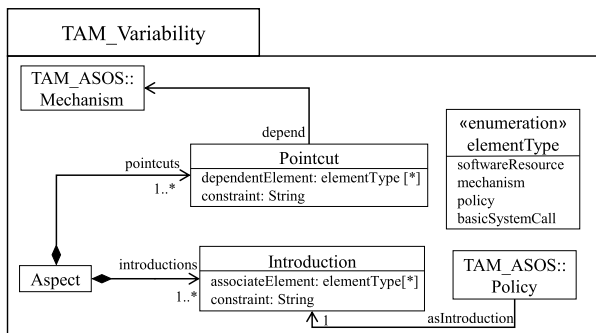


FIGURE 6. Variability domain model.

has its execution time. The functional blocks compose an execution sequence, which specifies the workload behavior of the task. In the modeling, we set an attribute of orderInExecutionSequence to indicate the order where the functional block is in the execution sequence. The WCET attribute is annotated by the WCET of a functional block, whose value is set by the designer according to the system requirements to specify the maximum budget of the execution time. The interaction type indicates whether the functional block interacts with other functional blocks. If the interaction exists, the interaction time between them should be concerned to perform the timing analysis. Both the interaction (denoted by interactionType and interactionObject) and the execution sequence (denoted by orderInExecutionSequence) specify the behavior of the functional block. Besides, the execution state and the execution condition are also concerned in our modeling. The executionState specifies the execution states (“to execute” or “executed”) of the functional block. The executionCondition specifies the condition of activating the execution of the functional block.

When interacting with the ASOS, a task is usually executed through a set of system calls, whose time cost contributes to the WCRT of a task. Such system call is realized based on the basic system calls in ASOS.

3) TAM_VARIABILITY PACKAGE

The TAM_Variability package contains the components related to the configurable policy in an ASOS, as shown in Fig. 6. The policy crosscuts the task’s execution in

a system. The crosscutting structure can be modeled by the aspect-oriented modeling (AOM) [13], which is a technique that allows reworking a reusable software asset (model) for the purpose of customization. Based on the AOM, an aspect-oriented method is provided for modeling the variability.

In AOM, the concept of Aspect is defined to describe the crosscutting structure. An Aspect consists of a set of Introductions and Pointcuts. The Introduction describes the crosscutting structure, which acts as an aspect model introduced to the base model. The Pointcut describes a set of join points between the aspect model and the base model. For the microkernel-based RTES, the kernel and the task are modeled as the base model; the configurable policy (i.e., variability) is modeled as the aspect model. When configuring a policy, the policy is modeled as the Introduction, and the elements in the base model used to realize the policy are modeled as the Pointcut. In our modeling approach, a new policy can be modeled by the aspect model without changing the base model.

When modeling the Introduction, the associate elements and constraints need to be specified. The associate element indicates the elements, which are associated with the policy and need to be introduced together with the Introduction. For example, for the round-robin scheduling policy, a time slice should be introduced together with the policy. Such associate elements include the software resource and the sub-policy. The constraint specifies the conditions to apply the Introduction. For example, when applying a dynamic priority scheduling policy, the priority of task should be changeable.

When modeling the Pointcut, the dependent elements and constraints need to be specified. The dependent element indicates the elements in the base model that contain the Pointcut. For example, the scheduling mechanism is a dependent element for the Pointcut between scheduling policy and base model. Such dependent elements include the mechanism and the basic system call. The constraint indicates the conditions to apply the Pointcut.

B. UML PROFILE FOR THE DSL

UML [4] is the most widely used modeling language, it provides an extension mechanism for the particular domains. Based on the extension mechanism, a UML profile for the TAM domain model (named TAM_UML) is proposed in terms of the class diagram. TAM_UML defines how the elements of domain model extend the metaclasses of UML metamodel to obtain the corresponding stereotypes.

In the stereotypes of TAM_ASOS, both the Mechanism and Policy extend the UML::Class to describe the essential functions in the ASOS. For the two stereotypes, the type attribute indicates the type of the OS function; the parameter attribute describes the parameters needed to realize the OS function. SoftwareResource, Hardware, and BasicSystemCall extend the UML::Class and the UML::Property. The type of a kernel determines the WCET of its basic system calls. The detailed semantic descriptions of these stereotypes are provided in Table.1.

TABLE 1. Stereotypes for TAM_ASOS.

Stereotype	Attributes	Description
Mechanism	type parameter	Type of mechanism used in ASOS. Parameter needed in the mechanism.
Policy	type parameter	Type of policy configured in ASOS. Parameter needed in the policy.
SoftwareResource	type isAvailable	Indicates the type of resource. Indicates whether the resource is available.
Hardware	processorType	Type of processor used in the hardware platform.
BasicSystemCall	kernelType WCET	Type of the kernel to realize the basic system call. WCET of basic system call on a given hardware.

TABLE 2. Stereotypes for TAM_Task.

Stereotype	Attributes	Description
Task	priority	Priority setting for task to indicate the execution order.
	state	State of task during scheduling.
	releaseTime	Indicates the time when the task is released.
	blockingTime	Blocking time caused by other tasks during scheduling.
	suspensionTime	Suspension time of task suspended by the processor.
	WCRT	Worst case response time of task.
	deadline	Maximal time bound on the completion of this task that must be met.
isFinished	Indicates the task is completed.	
FunctionalBlock	interactionType	Indicates whether the functional block interacts with others.
	interactionObject	Interaction object of the functional block (if exists).
	executionState	Indicates the state of functional block during execution.
	executionCondition	Specifies the condition for activating the execution of functional block.
	WCET	WCET budget preset for the functional block.
SystemCall	orderInExeSeq	Indicates the order of functional block in the execution sequence of task.
	type timeCost	Type of system call used in task. Time spent at the system call.

TAM_Task defines three kinds of stereotypes for the task: Task, FunctionalBlock, and SystemCall, whose detailed semantic descriptions are provided in Table.2. Task extends the UML::Class and is described in terms of priority setting, scheduling state, scheduling time, and timing requirement. Specifically, the attributes of *state*, *blockingTime*, *suspensionTime*, *WCRT* are defined to specify the scheduling. The default state of a task is set as “ready”. The FunctionalBlock extends the UML::Class and the UML::Property, which is specified from two aspects. The first aspect represents the relationship between the functional block with other functional blocks, which is described in terms of the interaction type, the interaction object and the order in execution sequence. The second aspect represents the execution of the functional block, which is described in terms of execution state, execution condition, and WCET. The attribute of *executionState* is defined to specify the execution of a functional block. The default value of the *executionState* is set as “to execute”, which indicates that the functional block is ready to execute. Once the execution is finished, the attribute of

TABLE 3. Stereotypes for TAM_Variability.

Stereotype	Attributes	Description
Pointcut	depElement constraint	Indicates the elements that the pointcut depends on. Constraint of implementing the pointcut.
Introduction	assElement constraint	Indicates the elements that the introduction associates with. Constraint of implementing the introduction.

executionState is set as “executed”. The *executionCondition* of a functional block is that: its execution state is “to execute”, the execution state of its predecessor in the execution sequence is “executed”, and the state of the task is *running*. The SystemCall extends the UML::Class and the UML::Property. The time cost of the system call, which depends on the type of the system call, also contributes to the time cost of a task. Then, the attributes of type and time cost are added to the SystemCall.

There are two stereotypes in TAM_Variability: Pointcut and Introduction. Both the stereotypes extend the UML::Class and the UML::Association. The Introduction is used to model the configurable policy and its associate elements. The Pointcut is used to model the joint point and its dependent elements in the kernel. The attribute of constraint is defined for the two stereotypes to indicate their implementation constraints. The reason by which the two stereotypes extend the UML::Association is that the relationship between aspect model and base model is established by the associations between them. The detailed semantic descriptions for TAM_Variability are provided in Table.3.

V. TIMING ANALYSIS FOR MICROKERNEL-BASED RTES DESIGNS

This section concerns the timing analysis approach for the microkernel-based RTES designs based on our modeling. In what follows, we give an overview of our timing analysis approach in Sect. V-A; define the extensible timing analysis tree (ETAT) in Sect. V-B; explain the model mapping approach in Sect. V-C; and detail the timing analysis method in Sect. V-D.

A. OVERVIEW OF TIMING ANALYSIS APPROACH

We propose a static approach to analyze the WCRT of a task based on the timing behavior of the functional blocks and that of the system calls in the task. The execution sequence and the WCET of the functional blocks have been modeled for each task, as well as the ASOS operations used in the task and the WCET of the corresponding basic system calls. We define a timing analysis tree to represent the execution sequence, WCET and the ASOS operations. As the timing analysis tree is extensible to express the variability, it is also called the extensible timing analysis tree (ETAT for short). As shown in Fig. 7, the task model, including the base model and the aspect model, is mapped to a ETAT. In the ETAT, the policy in the aspect model is refined by the execution rules defined in the child node of the policy node. If a new policy (variability)

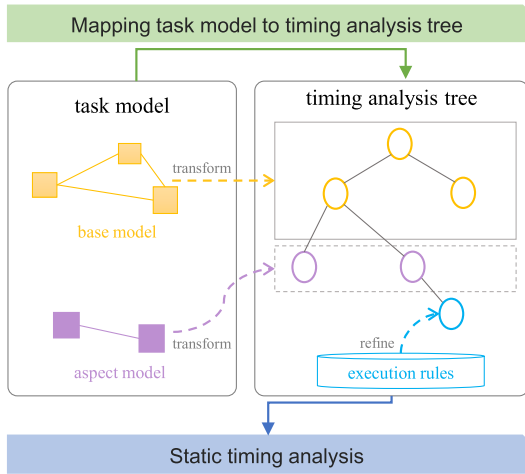


FIGURE 7. Timing analysis for the RTES model.

is configured, the only part to be modified in the ETAT is the policy node together with its child node. The benefits of our method based on the ETAT are manifold: first, it allows the evaluation of the timing requirements under the configurable policies (i.e., variability); second, it simplifies the analysis by fixing the part of base model and only replacing the part of aspect model; third, it unifies the analysis method for the alternative policies, either existing ones or user-defined ones.

B. TIMING ANALYSIS TREE

In this section, we first introduce the definition of the timing analysis tree (ETAT); then, we propose a canonical form to define the execution semantics for the ETAT by refining the ASOS operations, based on which the timing analysis is performed. It should be noted that the focus has been placed on the operations in the ASOS, while the operations in the tasks are specified as a set of functional blocks. Based on the proposed canonical form, we introduce the execution rules for the three basic OS mechanisms (i.e. configurable policies): scheduling, IPC and resource access with respect to the ASOS.

1) DEFINITION OF ETAT

An ETAT is a tree structure, which consists of a set of nodes and edges. Formally, the ETAT is defined as: $ETAT = (TreeNode, TreeEdge)$.

There are three types of nodes in the ETAT as follows:

- *object*: The object node specifies the task and the functional blocks in the task model;
- *operation*: The operation node specifies the ASOS operations in the task model, including the kernel mechanism and the configurable policy;
- *parameter*: The parameter node specifies the basic system calls used in the task model, as well as the defined execution rules, which will be introduced in the next section.

The relationship between nodes is specified as the edge in ETAT, which includes five types as follows:

- *use*: A task or a functional block uses the mechanism (or policy) in the ASOS; The execution rules and the WCET of basic system calls are used by the mechanism (or policy).
- *realize*: The policy is realized based on the mechanism.
- *consist*: The functional blocks compose a task.
- *sequence*: The successor of a functional block is its sub-sequence in the execution sequence.
- *operate*: The mechanism (or policy) operates on the task or the functional block.

2) EXECUTION SEMANTICS FOR ETAT

The operations in an ASOS specify how the OS manages the tasks. The operation is refined by the execution rules in ETAT, which define the operating action and the timing action (i.e. time cost). The time cost for each operation is calculated based on the WCET of basic system calls. When a task in the system interacts with the OS platform, the basic system calls adopted by the task are affected by the system-parameters (such as the number of tasks in the system, the maximum size of the blocking queue, the maximum number of suspended tasks at any given moment, and so on). For example, whether a task calls the basic system call of *suspending a task* is affected by the maximum number of suspended tasks in the system at any given moment. Therefore, such relation between the system calls and the system-parameters is also specified in the execution rules.

A canonical form for the execution rules is defined as

$$State \xrightarrow{[Condition]/Action} State' \tag{1}$$

where *State* represents the current state of a task, *Condition* means the condition affecting the execution of the task, and *Action* is the timing or the operating action of the task triggered by the satisfaction of conditions.

The execution rules for *scheduling mechanism* are defined as shown in Fig. 8.

- Four basic states of a task (running, ready, blocked, suspended) are represented by *St_Run*, *St_Ready*, *St_Block* and *St_Suspend*, respectively.
- The set of conditions consists of *Cond_Preempted*, *Cond_First_Run*, *Cond_Wait_Event*, *Cond_Event_Arrive*, and *Cond_Time_Out*.
 - *Cond_Preempted* represents the condition of a task being preempted, either active or passive. The active preemption enables a task to give up the processor actively. For instance, the task is finished. The passive preemption enables a task to give up the processor passively due to the behavior of other tasks. For instance, other task has a higher priority.
 - *Cond_First_Run* represents the task is selected to run first among the tasks in ready queue.
 - *Cond_Wait_Event* represents that the task is waiting for an event.
 - *Cond_Event_Arrive* represent that the event (a task is waiting for) arrives.
 - *Cond_Time_Out* represents the waiting is timeout.

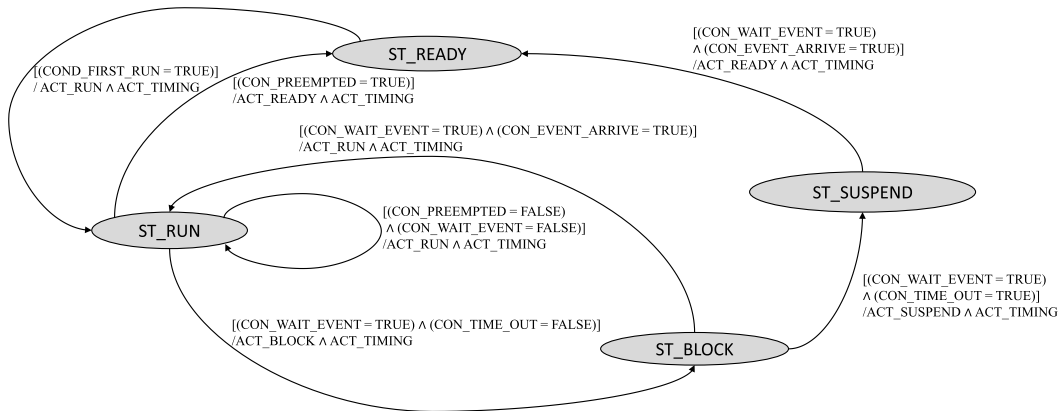


FIGURE 8. Execution rules for scheduling.

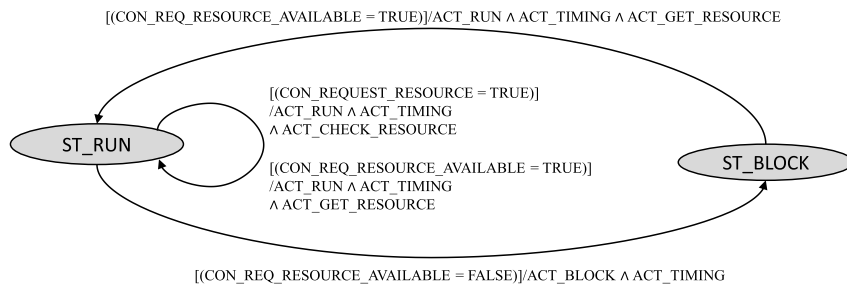


FIGURE 9. Execution rules for resource access.

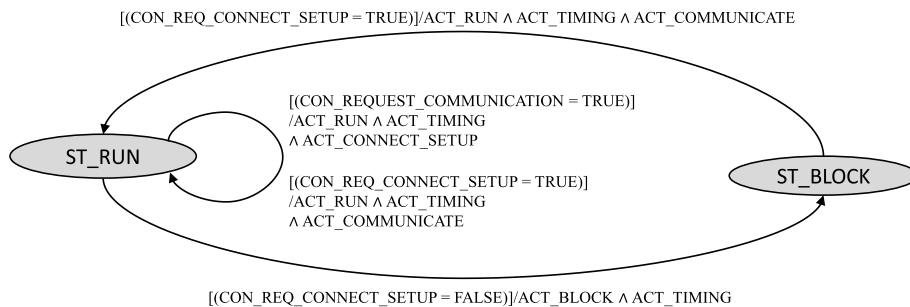


FIGURE 10. Execution rules for IPC.

- The operating actions (i.e. running, readying, blocking and suspending) are represented by *Act_Run*, *Act_Ready*, *Act_Block* and *Act_Suspend* respectively. The timing action is defined as *Act_Timing*, which is used to record the time cost of the operating actions.

The execution rules for *resource access mechanism* are defined as shown in Fig. 9.

- Two basic states of *St_Run* and *St_Block* are involved.
- The conditions of *Cond_Request_Resource* and *Cond_Req_Resource_Available* are used.
 - *Cond_Request_Resource* represents the task requests a resource during the execution.
 - *Cond_Req_Resource_Available* represents the resource requested is available right now.
- The operating actions include *Act_Run*, *Act_Block*, *Act_Timing*, *Act_Check_Resource* and *Act_Get_Resource*. Among them, *Act_Check_Resource* is defined to check

whether the resource is available, *Act_Get_Resource* is defined to obtain the available resource.

The execution rules for *IPC mechanism* are defined as shown in Fig. 10.

- Two basic states of *St_Run* and *St_Block* are involved.
- The conditions of *Cond_Request_Communication* and *Cond_Req_Connect_Setup* are used.
 - *Cond_Request_Communication* represents the task requests a communication with other task during the execution.
 - *Cond_Req_Connect_Setup* represents the connection for the requested communication is set up.
- The operating actions include *Act_Run*, *Act_Block*, *Act_Timing*, *Act_Connect_Setup* and *Act_Communicate*. Among them, *Act_Connect_Setup* is defined to set up the connection, *Act_Communicate* is defined to communicate with other task.

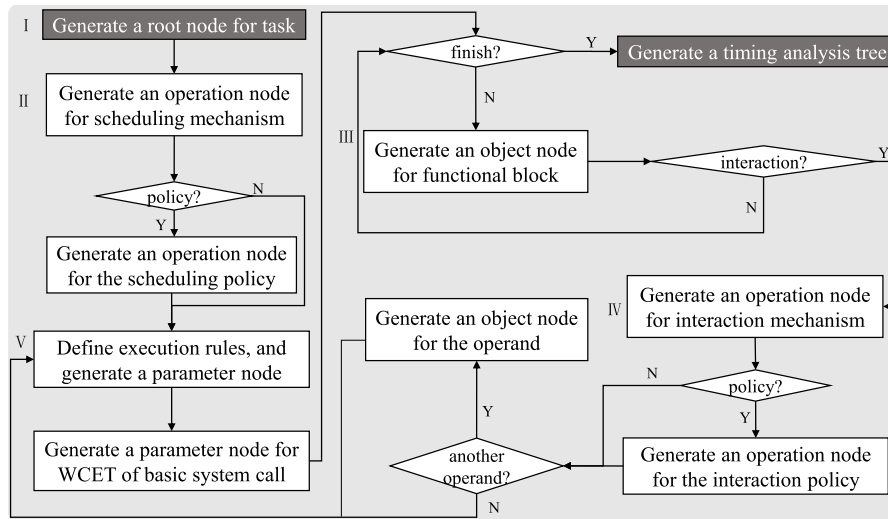


FIGURE 11. Mapping process.

The above three kinds of execution rules are specified for the three mechanisms in the microkernel. For the configurable policy (i.e., variability), we only need to refine the relevant conditions in the execution rules to describe the execution semantics for the policy. Specifically, as the preemption condition varies with different scheduling policy, a given scheduling policy needs to refine the *Cond_Preempted* in the execution rules of Fig. 8. Similarly, a given resource access policy needs to refine the *Cond_Req_Resource_Available* in the execution rules of Fig. 9, and a given IPC policy needs to refine the *Cond_Req_Connect_Setup* in the execution rules of Fig. 10.

C. MAPPING RTES MODEL TO ETAT

This section presents the mapping of the RTES model to the ETAT. In the RTES model, each task model and its associated operations are mapped to a ETAT. The task model consists of functional blocks and system calls. The realization of a system call depends on the mechanism/policy and the basic system calls in the ASOS model. The attributes of the node in ETAT consist of two parts: one part comes from the attributes of its corresponding component in the system, the other part is the attribute of *timeCost*, which is used to record the consuming time by the component. The execution rules defined in the above section are attached to the ETAT to refine the mechanism/policy node. The mapping process is shown in Fig. 11.

- Step I: Generate an *object* node for the task, which serves as the root node.
- Step II: Generate an *operation* node for the scheduling mechanism, which serves as a child of the root node. A *use* edge is used to link the root node and the *operation* node. If there exists a scheduling policy based on the scheduling mechanism, continue to generate an *operation* node for the scheduling policy, which serves as a child of the mechanism node. A *realize*

edge is used to link the mechanism node and the policy node.

- Step III: Generate an *object* node for each functional block in the task. According to the attribute of *order-InExecutionSequence* in the functional block:
 - (1) if the functional block is the first one in the execution sequence, its corresponding *object* node serves as a child of the root node, and a *consist* edge is used to link the root node and the *object* node;
 - (2) if the functional block is neither the first one nor the last one in the execution sequence, its corresponding *object* node serves as a child of its antecedent functional block node, and a *sequence* edge is used to link them;
 - (3) if the functional block is the last one in the execution sequence, the mapping process is finished after generating its corresponding *object* node, and then an ETAT is output.
- Step IV: If the functional block has an interaction (IPC or resource access) with others, generate an *operation* node for the interaction mechanism, which serves as a child of the functional block. A *use* edge is used to link the functional block node and the mechanism node. If there is a policy based on the interaction mechanism, continue to generate an *operation* node for the policy, which serves as a child of the mechanism node. A *realize* edge is used to link the mechanism node and the policy node. If this interaction includes another operand, continue to generate an *object* node for the operand, which serves as a child of this interaction node. An *operate* edge is used to link the operand node and the interaction node.
- Step V: Generate the *parameter* nodes for each operation node with the *use* edge, which include a *parameter* node for the basic system calls, and a *parameter* node for the execution rules.

```

function timing_analysis_process (T)
1:  visit the root node rn of T
2:  if (rn.isFinished == true) then
3:      return
4:  else
5:      visit the scheduling child node sc of rn
6:      OperationNodeAnalysis_forScheduling
7:      update rn.time_cost
8:      if (t.state != "running") then
9:          return
10:     else
11:         visit the functional block child node fb of rn
12:         set fb as the functional block node to be analyzed fb_ta
13:         while fb_ta is not the last
14:             if (fb_ta.executionState == "to execute") then
15:                 if fb_ta.executionCondition is satisfied then
16:                     ObjectNodeAnalysis_forFunctionalBlock
17:                     update rn.time_cost
18:                 else
19:                     return
20:                 end if
21:             end if
22:             set the functional block child node of fb_ta as fb_ta.
23:             visit the node fb_ta
24:         end while
25:         set rn.isFinished = true
26:         return rn.time_cost as the WCRT
27:     end if
28: end if

```

FIGURE 12. Timing analysis process.

D. TIMING ANALYSIS OF ETAT

WCRT of a task consists of the scheduling time, the interaction time with other functional blocks, and the WCET of the functional blocks in this task. Both the scheduling time and the interaction time rely on the operations in ASOS, and consist of the WCET of related basic system calls. We propose to analyze the WCRT by traversing the ETAT. During the traversal, the consuming time of each node is analyzed based on its child nodes. After the traversal, the time cost of the root node is achieved, i.e. the WCRT of the task.

Given an arbitrary task t and its corresponding ETAT T , the timing analysis process is shown in Fig. 12. First, the root node of T is visited to check whether the task t is completed (L. 1). If not, the scheduling child node of the root node is visited to check whether the task is executable (L. 6). The time cost of the scheduling operation is analyzed, which is used to update the time cost of the root node (L. 7,8). If the task is executable (with the state of "running"), the functional block child node of the root node is visited, then the nodes of the functional block which have a *sequence* edge between them are visited one after another (L. 25). For the functional block node that is ready to execute, if its execution condition is satisfied, its time cost is analyzed to update the time cost of the root node (L. 16-19). When all the functional block nodes

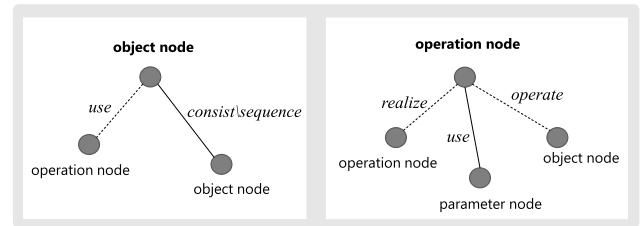


FIGURE 13. Basic structure of object node and operation node.

are visited, the task is set as finished, and the time cost on the root node indicates the WCRT of the task.

Next, we introduce the timing analysis for the scheduling node and the functional block node. The time cost of the scheduling node is the time spent at the scheduling operation. The time cost of the functional block node includes the time spent at the object itself and the interaction operation (if exists). Therefore, we introduce a timing analysis method for such two types of nodes.

According to the definition of ETAT, the basic structure of *object* node and *operation* node is shown in Fig. 13. (1) For the *object* node, it has a child node of object, which has a *consist* (for task) or *sequence* (for functional block) edge with the *object* node. If the *object* node has a scheduling operation or interaction operation, an *operation* node is used as its child node. (2) For the *operation* node, it has a child node of parameter, which is used by the *operation* node. If the *operation* node has an extended operation (for policy), the *realize* edge is used to link them. If the *operation* node has an other operand, there is a child node of object for the *operation* node with the *operate* edge.

The timing analysis for the *object* node of functional block and the *operation* node is presented as follows. For ease of introduction, we call such a child node that has a *use* edge with its father node as the *use* child node of the father node in brief (the same for other edges).

- C1: For the *object* node of the functional block, its time cost includes the time spent at itself and its *use* child node (if exists). The time spent at the functional block itself is specified by its WCET attribute. The *use* child node is actually the operation node, whose time cost can be analyzed by the way in C2.
- C2: For the *operation* node, its time cost includes the time spent at its *use* child node, *realize* child node (if exist) and *operate* child node (if exist). The time spent at the *use* child node is the time cost of the system call, which is analyzed based on the execution rule. Specifically, the execution rule specifies the concrete process of a system call being called by the task. Given a certain system-parameters, the times of the basic system call being used to realize the system call can be obtained. Then, the time cost of a system call is the product of the times and the WCET of the basic system call. If this *operation* node has a *realize* child node, the *realize* child node is actually an operation node, whose time cost is analyzed by the same way as above. If this *operation*

TABLE 4. WCET settings for functional blocks (in one thousand CPU cycles).

FBs in <i>balance</i>	WCET	FBs in <i>navigation</i>	WCET	FBs in <i>remote</i>	WCET
<i>Initialization</i>	5	<i>Initialization</i>	1	<i>Initialization</i>	10
<i>GetInfoFromGyro</i>	10	<i>SendDetector</i>	3	<i>GetInfoFromInfrared</i>	800
<i>GetInfoFromInclino</i>	10	<i>FindObstacle</i>	8	<i>ExecuteCommand</i>	3000
<i>Calculation</i>	30	<i>AvoidObstacle</i>	5		
<i>KeepBalance</i>	50				

node has other operands except its father *object* node (as the *operation* node is used by its father node, the father node is one operand of this operation), the *operate* child node is actually an *object* node, whose time cost is analyzed by the same way as C1.

E. CORRECTNESS OF TIMING ANALYSIS

The response time of a task (RTT) in RTESs consists of the following three parts:

$$RTT = ETf + ITt + ITp \quad (2)$$

where ETf , ITt and ITp denote the execution time of functions in the task, the interaction time with other task(s), and the interaction time with the OS platform respectively.

Here, we illustrate that our timing analysis based on the timing analysis tree can generate a safe response time for tasks.

In the timing analysis tree (ETAT), five kinds of relationships are defined to express the tasks and the OS platform in system (see Sect.V-B.1). Specifically, the execution of functional blocks in a task is expressed by the relationships of *consist* and *sequence* between the *object* nodes (of the functional blocks); the interaction with other task(s) is expressed by the relationship of *operate* between the *object* node (of other tasks) and the *operation* node (of the interaction); the interaction with the OS platform is expressed by the relationship of *use* between the *object* node (of the task) and the *operation* node (of the interaction). Therefore, the timing analysis tree is capable of specifying all the basic relationships for the tasks and the OS platform.

For the ETAT of a task, it can be seen from above that there are always two basic nodes to be analyzed, i.e., the *object* node and the *operation* node. The timing analysis of these two basic nodes can be calculated by the rules in C1 and C2. Specifically, these rules calculate the execution time and the interaction time for a task based on the *parameter* node, which specifies the timing behavior of the basic units (i.e., functional block and basic system call) in system using the worst-case execution time (WCET). When the nodes in ETAT are traversed to calculate the three parts of the response time (i.e., ETf , ITt and ITp), the result is about the worst-case response time (WCRT) for tasks, which is a safe result for the hard real-time system.

VI. CASE STUDY

A. EXPERIMENTAL SETUP

In this section, we illustrate the application of our approach to a real-life robot controller system. The robot controller system (RCS) [46], which consists of three tasks, is used to keep

the robot operating normally. Among the tasks, the *balance* task is to keep the balance of robot by calculating the input from gyroscope and inclinometer; the *navigation* task is to avoid obstacles during the process of going to the destination; the *remote* task is to receive the remote command via infrared. The services of infrared sensor, gyroscope and inclinometer are realized by the interrupt service routines (ISR), which are corresponding to *infrared_isr*, *gyro_isr* and *inclino_isr* respectively. To implement the RCS, we use an open source kernel $\mu C/OS-II$, which is based on the microkernel and freely available for non-commercial usage, to customize the ASOS. The $\mu C/OS-II$ kernel implements a static priority scheduling policy, and has an optional policy of round robin scheduling. In this case study, we model the RCS on such ASOS, and analyze the timing requirements to assess these two scheduling policies.

The target processor presented in [47] is adopted to implement the ASOS. The corresponding WCET of basic system calls in the $\mu C/OS-II$ kernel on the processor, which is also given in [47], is used in this case study. For the two scheduling policies, the time slice in the round robin (RR) scheduling is set as 10 thousands CPU cycles, the priority (P) for the three tasks in the static priority (SP) scheduling is set as: $P(balance) = 4$, $P(navigation) = 6$, $P(remote) = 5$. For the tasks, their timing requirements are represented by the deadline (D), which are set as (in one thousand CPU cycles): $D(balance) = 200$, $D(navigation) = 40$, $D(remote) = 4000$. Within the tasks, the functional blocks (FBs) together with their WCET are set as shown in Table. 4.

B. EXPERIMENTAL PROCESS

1) MODEL THE RCS

This subsection presents the timing analysis model of the robot controller system, which includes the ASOS model and the task model.

The ASOS model represents the components in the ASOS, each of which is modeled as a class. Among the components, the three mechanisms of scheduling, IPC and resource access are modeled by the stereotype $\ll Mechanism \gg$. For the scheduling mechanism, the corresponding scheduling policy is modeled by the stereotype $\ll Policy \gg$, which includes two alternative realizations: SP scheduling and RR scheduling. These two alternatives are modeled as *SP_Introduction* and *RR_Introduction* respectively by the stereotype $\ll Introduction \gg$. As an introduction, the SP scheduling policy has an associate element of priority; the RR scheduling policy has an associate element of time slice. Both of the associate elements are modeled by the *assElement* attribute. When the scheduling policies are introduced

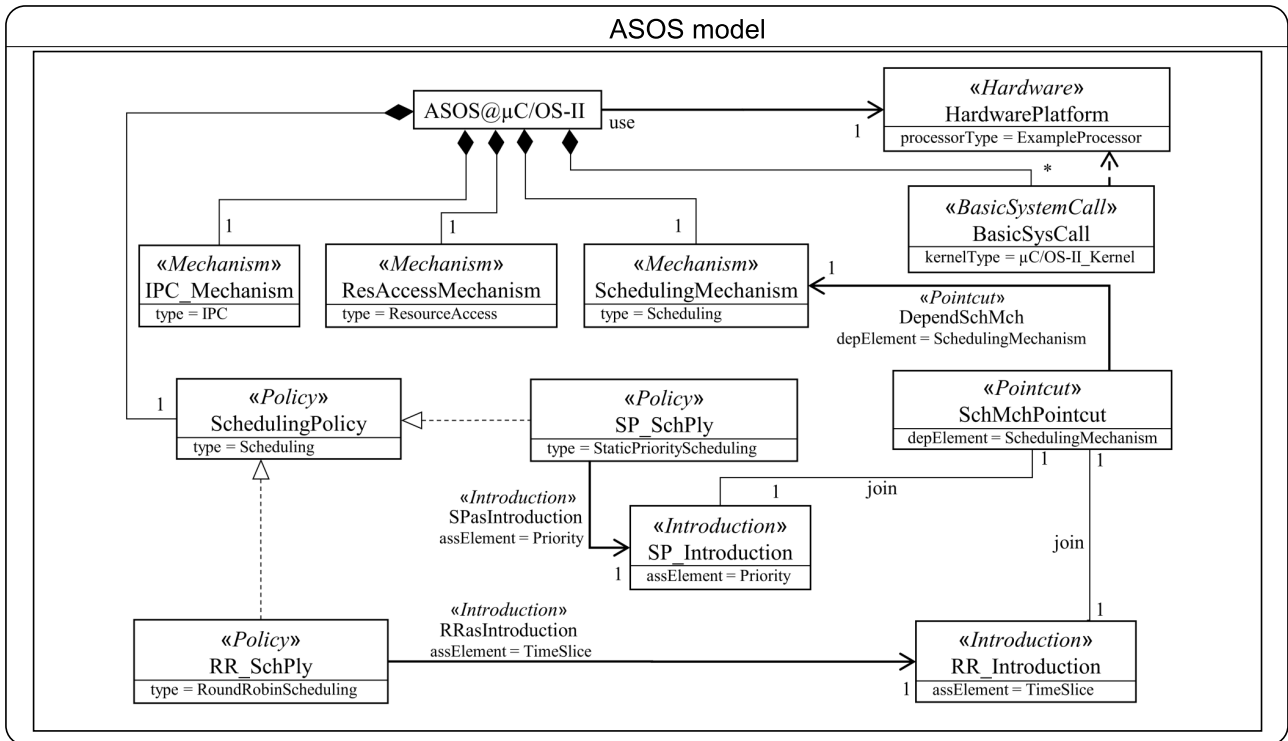


FIGURE 14. ASOS model.

to the ASOS, the corresponding pointcuts are based on the scheduling mechanism, which is modeled by the *depElement* attribute in the stereotype `<<Pointcut>>`. The basic system calls used by the scheduling mechanism are modeled by the stereotype `<<BasicSystemCall>>`, as well as the other two mechanisms. As the $\mu\text{C}/\text{OS-II}$ kernel is used in the ASOS, such type of kernel is specified for the basic system calls as the *kernelType* attribute. The detailed ASOS model is shown in Fig. 14.

As shown in Fig. 15, the tasks of *balance*, *navigation*, and *remote* are modeled by the stereotype of `<<Task>>`, with the priority and deadline settings in Sec. VI-A. We assume the three tasks are released at the same time (*releaseTime* = 0). Once released, the state of the task is set as *ready*. Then, the scheduling mechanism together with its configurable scheduling policy and basic system calls are used to schedule the three tasks. Besides, the IPC mechanism together with its basic system calls are used by the *balance* task and the *remote* task to communicate with the ISRs. When the task is completed, its WCRT specifies the information to return as an analysis result.

For the tasks, each of them has a set of functional blocks and system calls, which are modeled by the stereotypes of `<<FunctionalBlock>>` and `<<SystemCall>>` respectively. Each functional block is modeled in terms of WCET, interaction type, and the order in execution sequence (*orderInExeSeq* attribute). For the interaction type, if it is *interaction*, then the corresponding interaction object should also be specified. For example, the functional blocks of *Get-*

InfoFromGyro and *GetInfoFromInclino* in the *balance* task have the interaction objects of *Gyro_isr* and *Inclino_isr* respectively; the functional block of *GetInfoFromInfrared* in the *remote* task has the interaction object of *Infrared_isr*. For the functional blocks with the interaction type of *interaction*, the system call is involved to realize such interaction. In this case study, there are two types of system call used: scheduling (*SchSysCal*) and IPC (*IPC_SysCal*), both of them are based on the basic system calls in ASOS.

In the task model, when the SP scheduling is adopted, the *SP_SchPly* in ASOS model is introduced as the *SchPly-Introduction* to realize the scheduling of the three tasks. It is the same for the RR scheduling. As the scheduling policy is alternative, we set both the *assElement* attributes in *SchPly-Introduction* and *asIntroduction* as variable (“\$v1”, “\$v2” expression respectively)

2) CONFIGURE EXECUTION RULES FOR THE POLICIES

This subsection presents the execution rules configured for the two scheduling policies. As shown in Fig. 16, these execution rules refine the preempted condition in the scheduling mechanism (as shown in Fig. 8). Specifically, for the SP scheduling, an arbitrary task *T* is preempted when there exists a ready task with a higher priority than *T*; for the RR scheduling, the task *T* is preempted when the time slice for *T* is used up. It should be noted that the *CON_PREEMPTED* in the execution rules of scheduling mechanism (as shown in Fig. 8) is set by the actions of *ACT_SET_PREEMPTED_TRUE* or

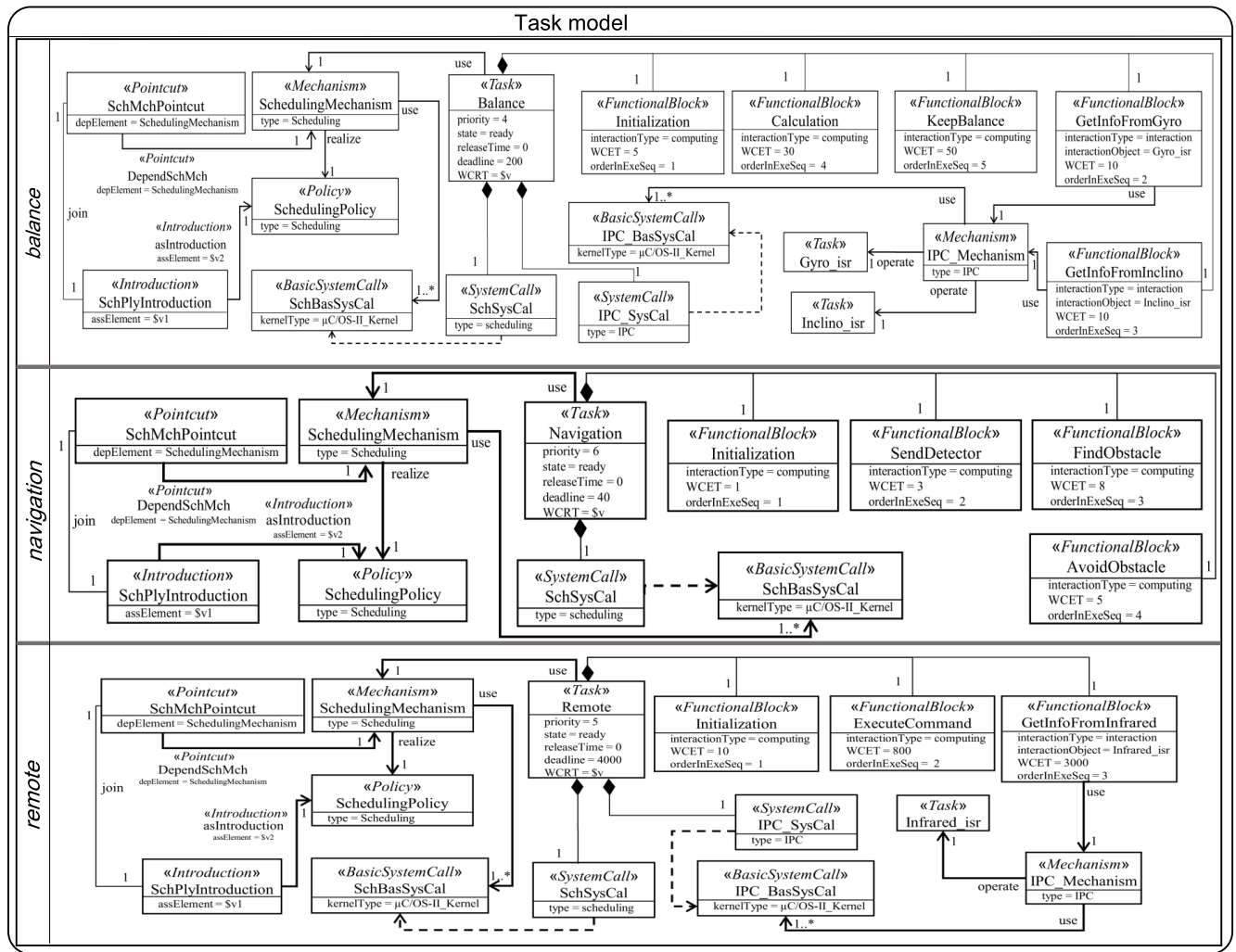


FIGURE 15. Task model.

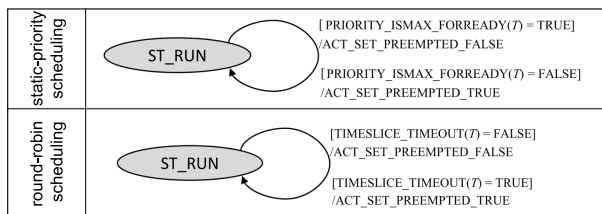


FIGURE 16. Execution rules for two alternative scheduling policies.

ACT_SET_PREEMPTED_FALSE in the execution rules of the two scheduling policies.

3) MAP RCS MODEL TO ETAT

This subsection gives details of the transformation of the RCS model towards the timing analysis tree ETAT.

For the three tasks in RCS model, we map each of them to an ETAT as shown in Fig. 17. In the mapping, we transform the classes in RCS model together with the execution rules to the nodes in ETAT. According to the three kinds of nodes

in the ETAT as defined in Sec. V-B, the mapping includes three parts: 1) Each of the three task classes is mapped to the root node. The functional block classes included in each task are mapped to the successive child nodes of the root node in the respective ETAT based on the execution sequence (represented by *orderInExeSeq* attribute). Task classes and functional block classes are mapped to the nodes with the type of *object* (represented by the green color). 2) The scheduling mechanism class used by the task is mapped to the child node of the root node in each ETAT, and operates on the other two tasks. The scheduling policy is mapped to the child node of the scheduling mechanism. Besides, the IPC mechanism class is mapped to the child node of the functional blocks of *GetInfoFromGyro*, *GetInfoFromInclino*, and *GetInfoFromInfrared*. Mechanism classes and policy classes are mapped to the nodes with the type of *operate* (represented by the orange color). 3) For the mechanisms, the basic system call classes (include *SchBasSysCal* and *IPC_BasSysCal*) and the execution rules (defined in Fig.8 and Fig.10) are mapped to the child nodes of the mechanism. Besides, the configured

i.e., static priority scheduling and round-robin scheduling, as an example to illustrate our approach. Without loss of generality, any other scheduling policies can also be analyzed based on the timing analysis trees in Fig. 17 by configuring their execution rules. The configurable execution rules are mapped to the leaf node (as a parameter node) of the timing analysis tree, which makes it possible to analyze a new policy by only changing this leaf node. This is the same case for the policies of the other two mechanisms: IPC and resource access. Therefore, for a specific application in the domain of RTES, we only need such timing analysis tree (transformed from the RTES model based on our modeling) to assess the timing requirements under the various alternative policies, which cover the basic functions of the microkernel. Any policy (even defined by the designer himself) that can be described by our execution rules is supported by our approach. Thus, our approach can guide a designer towards a better policy in the time aspect for the microkernel-based RTES design.

VII. CONCLUSION AND PERSPECTIVES

With more and more application-specific operating systems (ASOS) based on the microkernel are applied in real-time embedded systems, it is necessary to assess such ASOS from a time point of view in the early design stage. There is a lack of the modeling and analysis approach to support the microkernel-based RTES on the configurable ASOS. In this paper, we are interested in the ASOS configured from the three essential functions of the microkernel: the task scheduling, the inter-process communication (IPC), and the resource access. Such configuration is realized based on a set of alternative policies, which is considered as the variability in this paper. To achieve this objective, we propose a modeling and timing analysis framework (MTAF) for the specific domain. This framework includes two parts: the DSL for the timing analysis modeling of the microkernel-based RTES and the timing analysis approach for the RTES design based on our modeling. Our approach is aimed to model and analyze the variability of the configurable policy in the microkernel-based RTES, which makes it capable of assessing the policy from a set of alternatives in advance at the early design stage.

There are four factors to threaten the internal validity of our approach: the division of the functional blocks in a task, the execution rules defined for a policy, the WCET of basis system calls in the kernel, and the WCET set by a designer for the functional blocks. Specifically, if the size of the functional block is too big, it is imprecise to analyze its time cost, as the interactions and system calls within the functional block may be ignored; on the contrary, if the size is too small, it will increase the burden of modeling and analyzing the functional blocks. Usually, the functional block should have no more than one system call or one interaction (with other functional block). In the analysis process, as the execution rules for the policy are defined by the designer, their correctness has a direct impact on the timing analysis. The WCET of basis system calls in the kernel is analyzed by the existing tools,

whose capacity also affects our results. Besides, the WCET of functional blocks is set by a designer based on his/her experience, which directly affects the precision of the timing analysis, i.e., whether the design provides a tight bound to guarantee the timing requirements for tasks. As the run-time environment in RTES is unpredictable, the WCET of functional blocks is usually set by a random variable to specify its multiple possible values. In this case, our approach is used to analyze the WCRT of tasks under each possible value of the WCET, then the design is assessed in a probabilistic way.

For the external validity of our approach, currently our approach only supports the configurable policies about the three aspects of scheduling, inter-process communication, and resource access. With the RTES is becoming more and more complex, more functions are needed by the ASOS, such as network management, file system, and so on. In the near future, we will improve our approach to support more kinds of OS functions in ASOS, such as synchronization. Besides, our approach currently only supports the real-time embedded system with aperiodic tasks on the uniprocessor. In our ongoing work, we are improving our approach to support the periodic tasks, as well as the multi-core platform.

ACKNOWLEDGMENT

The authors would like to thank the reviewers and editors for their valuable comments on this article.

REFERENCES

- [1] L. F. Friedrich, J. Stankovic, M. Humphrey, and M. Marley, "A survey of configurable, component-based operating systems for embedded applications," *IEEE Micro*, vol. 21, no. 3, pp. 54–68, May/Jun. 2001.
- [2] S. Engle. *It's Time: Avionics Need to Move to Multicore Processors*. Accessed: Mar. 26, 2018. [Online]. Available: <http://www.modern-avionics.com/news/2018/its-time-avionics-need-to-move-to-multicore-processors/>
- [3] P. H. Feiler, D. P. Gluch, and J. J. Hudak, "The architecture analysis & design language (AADL): An introduction," *Softw. Eng. Inst., Carnegie-Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-2006-TN-011*, 2006.
- [4] OMG. (2015). *Unified Modeling Language*. [Online]. Available: <http://www.omg.org/spec/UML/>
- [5] OMG. (2011). *UML Profile for Marte*. [Online]. Available: <http://www.omg.org/spec/MARTE>
- [6] S. Fürst et al., "AUTOSAR—A worldwide standard is on the road," in *Proc. 14th Int. VDI Congr. Electron. Syst. Vehicles, Baden-Baden*, Oct. 2009, pp. 1–16.
- [7] P. L. Martínez, L. Barros, and J. M. Drake, "Scheduling configuration of real-time component-based applications," in *Reliable Software Technology—Ada-Europe*. Berlin, Germany: Springer, 2010, pp. 181–195.
- [8] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, *Model Checking and the State Explosion Problem*. Berlin, Germany: Springer, 2011.
- [9] F. Stappert and P. Altenbernd, "Complete worst-case execution time analysis of straight-line hard real-time programs," *J. Syst. Archit.*, vol. 46, no. 4, pp. 339–355, 2000.
- [10] J. Gustafsson, A. Ermedahl, C. Sandberg, and B. Lisper, "Automatic derivation of loop bounds and infeasible paths for WCET analysis using abstract execution," in *Proc. IEEE 27th Int. Real-Time Syst. Symp.*, Dec. 2006, pp. 57–66.
- [11] G. Aupy, C. Brasseur, and L. Marchal, "Dynamic memory-aware task-tree scheduling," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, May/Jun. 2017, pp. 758–767.
- [12] R. Xu, L. Zhang, N. Ge, and J. Jing, "Timing analysis for microkernel-based real-time embedded system," in *Proc. 30th Int. Conf. Softw. Eng. Knowl. Eng.*, 2018, pp. 512–517.

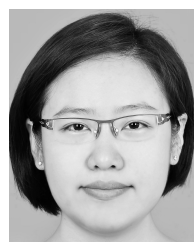
- [13] T. Elrad, O. Aldawud, and A. Bader, "Aspect-oriented modeling: Bridging the gap between implementation and design," in *Generative Programming and Component Engineering*. Berlin, Germany: Springer, 2002, pp. 189–201.
- [14] M. Z. Iqbal, A. Arcuri, and L. Briand, "Environment modeling with UML/MARTE to support black-box system testing for real-time embedded systems: Methodology and industrial case studies," in *Proc. Int. Conf. Model Driven Eng. Lang. Syst.*, 2010, pp. 286–300.
- [15] C. Mraidha, S. Tucci-Piergiorganni, and S. Gerard, "Optimum: A marte-based methodology for schedulability analysis at early design stages," *ACM SIGSOFT Softw. Eng. Notes*, vol. 36, no. 1, pp. 1–8, 2011.
- [16] V. Debruyne, F. Simonot-Lion, and Y. Trinquet, "EAST-ADL—An architecture description language," in *Architecture Description Languages*. Berlin, Germany: Springer, 2005, pp. 181–195.
- [17] I. Kuz, Y. Liu, I. Gorton, and G. Heiser, "CAMKES: A component model for secure microkernel-based embedded systems," *J. Syst. Softw.*, vol. 80, no. 5, pp. 687–699, 2007.
- [18] W. E. H. Chehade, A. Radermacher, S. Gerard, and F. Terrier, "Detailed real-time software platform modeling," in *Proc. 17th Asia Pacific Softw. Eng. Conf.*, Nov./Dec. 2010, pp. 108–117.
- [19] R. Mzid and M. Abid, "UML-based reconfigurable middleware for design-level timing verification in model-based approach," in *Proc. 11th Int. Design Test Symp.*, Dec. 2016, pp. 181–186.
- [20] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki, "A study of variability models and languages in the systems software domain," *IEEE Trans. Softw. Eng.*, vol. 39, no. 12, pp. 1611–1640, Dec. 2013.
- [21] J. L. Medina and Á. G. Cuesta, "From composable design models to schedulability analysis with UML and the UML profile for marte," *ACM SIGBED Rev.*, vol. 8, no. 1, pp. 64–68, 2011.
- [22] M. Hagner and U. Goltz, "Integration of scheduling analysis into UML based development processes through model transformation," in *Proc. Int. Multi-Conf. Comput. Sci. Inf. Technol.*, Oct. 2010, pp. 797–804.
- [23] M. Bohlin, Y. Lu, J. Kraft, P. Kreuger, and T. Nolte, "Simulation-based timing analysis of complex real-time systems," in *Proc. 15th IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2009, pp. 321–328.
- [24] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: A flexible real time scheduling framework," *ACM SIGAda Ada Lett.*, vol. 24, no. 4, pp. 1–8, 2004.
- [25] R. Alur and D. L. Dill, "A theory of timed automata," *Theory Comput. Science*, vol. 126, no. 2, pp. 183–235, 1994.
- [26] L. Popova-Zeugmann, "Time Petri nets," in *Proc. Time Petri Nets*, 2013, pp. 31–137.
- [27] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on UPPAAL," in *Formal Methods for the Design of Real-Time Systems*. Berlin, Germany: Springer, 2004, pp. 33–35.
- [28] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, "TIMES: A tool for schedulability analysis and code generation of real-time systems," in *Proc. Int. Conf. Formal Modeling Anal. Timed Syst.*, 2003, pp. 60–72.
- [29] N. Guan, C. Gu, M. Stigge, Q. Deng, and W. Yi, "Approximate response time analysis of real-time task graphs," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2014, pp. 304–313.
- [30] K. Lampka, S. Perathoner, and L. Thiele, "Analytic real-time analysis and timed automata: A hybrid methodology for the performance analysis of embedded real-time systems," *Des. Automat. Embedded Syst.*, vol. 14, no. 3, pp. 193–227, 2010.
- [31] Y. Lu, T. Nolte, and J. Kraft, "An approximate timing analysis framework for complex real-time embedded systems," in *Proc. IEEE 13th Int. Conf. Comput. Sci. Eng.*, Dec. 2010, pp. 102–111.
- [32] N. Ge, M. Pantel, and B. Berthomieu, "A flexible WCET analysis method for safety-critical real-time system using UML-MARTE model checker," *Rapport LAAS n° 12401*, 2016, pp. 1–12.
- [33] Y. H. Kacem, A. Mahfoudhi, A. Magdich, C. Mraidha, and W. Karamti, "Using MDE and priority time Petri Nets for the schedulability analysis of embedded systems modeled by UML activity diagrams," in *Proc. 19th Int. Conf. Workshops Eng. Comput. Based Syst.*, Apr. 2012, pp. 316–323.
- [34] M. Najja, S. B. Ahmed, and J.-M. Bruel, "New schedulability analysis for real-time systems based on MDE and Petri Nets model at early design stages," in *Proc. 10th Int. Joint Conf. Softw. Technol.*, vol. 1, Jul. 2015, pp. 1–9.
- [35] A. Colin and I. Puaut, "Worst case execution time analysis for a processor with branch prediction," *Real-Time Syst.*, vol. 18, nos. 2–3, pp. 249–274, 2000.
- [36] A. Colin and G. Bernat, "Scope-tree: A program representation for symbolic worst-case execution time analysis," in *Proc. 14th Euromicro Conf. Real-Time Syst.*, Jun. 2002, pp. 50–59.
- [37] R. Simmons and D. Apfelbaum, "A task description language for robot control," in *Proc. IEEE/RSSJ Int. Conf. Intell. Robots Syst.*, vol. 3, Oct. 1998, pp. 1931–1937.
- [38] T. Liu, M. Li, and C. J. Xue, "Instruction cache locking for multi-task real-time embedded systems," *Real-Time Syst.*, vol. 48, no. 2, pp. 166–197, 2012.
- [39] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg, "Fast: Frequency-aware static timing analysis," *ACM Trans. Embedded Comput. Syst.*, vol. 5, no. 1, pp. 200–224, 2006.
- [40] R. Arnold, F. Mueller, D. Whalley, and M. Harmon, "Bounding worst-case instruction cache performance," in *Proc. Real-Time Syst. Symp. (RTSS)*, Dec. 1994, pp. 172–181.
- [41] Y. Harada, K. Abe, M. Yoo, and T. Yokoyama, "Aspect-oriented customization of the scheduling algorithms and the resource access protocols of a real-time operating system family," in *Proc. IEEE Int. Conf. Smart City/SocialCom/SustainCom*, Dec. 2015, pp. 87–94.
- [42] L. K. Chong, C. Ballabriga, V.-T. Pham, S. Chattopadhyay, and A. Roychoudhury, "Integrated timing analysis of application and operating systems code," in *Proc. IEEE 34th Real-Time Syst. Symp.*, Dec. 2013, pp. 128–139.
- [43] D. P. B. Renaux, R. E. De Góes, and R. R. Linhares, "Performance characterization of real-time operating systems for systems-on-silicon," in *Proc. 12th Brazilian Workshop Real-Time Embedded Syst.*, 2010, pp. 1–12.
- [44] X. Li, Y. Liang, T. Mitra, and A. Roychoudhury, "Chronos: A timing analyzer for embedded software," *Sci. Comput. Program.*, vol. 69, nos. 1–3, pp. 56–67, 2007.
- [45] F. Verdier, B. Miramond, M. Maillard, E. Huck, and T. Lefebvre, "Using high-level RTOS models for HW/SW embedded architecture exploration: Case study on mobile robotic vision," *EURASIP J. Embedded Syst.*, vol. 2008, no. 1, 2008, Art. no. 349465.
- [46] T. Braunl, "EyeBot: A family of autonomous mobile robots," in *Proc. 6th Int. Conf. Neural Inf. Process.*, vol. 2, Nov. 1999, pp. 645–649.
- [47] M. Lv et al., "WCET analysis of the μ C/OS-II real-time kernel," in *Proc. Int. Conf. Comput. Sci. Eng.*, vol. 2, 2009, pp. 270–276.



RONGFEI XU received the B.S. and M.S. degrees from Shijiazhuang Tiedao University, Shijiazhuang, China, in 2010 and 2013, respectively. He is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, Beihang University, Beijing, China. His research interest includes modeling and analysis of real-time systems.



LIZHANG is currently a Professor with the School of Computer Science and Engineering, Beihang University, Beijing, China, where she is also the Associate Dean of the School of Software and leading a research group in Software Engineering Institution. She has published over 100 articles in refereed conferences and journals. Her research interests include system modeling, model-driven engineering, software product line, and business process modeling and simulation.



NING GE received the Ph.D. degree in software safety and highly performance computing from the IRT Laboratory of CNRS, University of Toulouse, Toulouse, France, in 2014. She is currently a Lecturer with the School of Software, Beihang University, Beijing, China. Her research interests include formal methods, software safety, model-driven engineering, and real-time embedded systems.

...