Available online at www.ijpe-online.com

**International Journal of Performability Engineering**

vol. 15, no. 3, March 2019, pp. 872-883
DOI: 10.23940/ijpe.19.03.p16.872883

# Time-Oriented Modeling and Analysis for Real-Time System under Variability

Rongfei Xu*

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

**Abstract**

With the advent of MDA, there is an urge to analyze the time performance in real-time systems under various design decisions at the very early stages of design. With the wide application of customized real-time operating system (RTOS) based on a microkernel, we propose a time-oriented modeling and analysis approach for real-time systems based on RTOS at the early stages of design. According to the commonality and variability in the system, a modeling approach for analyzing the time under variable design decisions is presented. These design decisions include various hardware environment, user-level services adopted in RTOS, and the task settings. In the analysis approach, a timing tree with the operating and timing rules is defined and used based on the time annotations of the basic system call of RTOS and worst-case execution time (WCET) of the functional block in a task to analyze the execution time. The modeling and analysis approach proposed is capable of analyzing new decisions without any changes in the model, which is helpful to find the best design decision to improve the real-time in the system. Finally, a real-life aircraft landing control system is taken as an example to evaluate this approach.

*Keywords*: real-time system; time-oriented modeling and analysis; MDA; variability; event-driven rules

(Submitted on October 24, 2018; Revised on November 26, 2018; Accepted on December 28, 2018)



## 1. Introduction

In real-time systems such as embedded systems or cyber-physical systems, time is a key factor to ensure the correctness of these software systems [1]. More and more studies focus on the timing performance in such systems. However, most of this research begins to analyze the time performance after the software has been developed [2-3], which is already too late. With the rise of MDA [4], studying software problems at the design stage has become a hot research topic [5]. The existing research on analyzing the time performance in a real-time system still has the following problems: (1) as the real-time system is closely related to the platform [6], it should combine the application with the run-time environment to analyze the time [7]. However, existing studies only consider some partial aspects of the platform and cannot take full account of the whole platform. For example, only the scheduling aspect [8] or the resource access protocol [9] is considered. (2) The studies are always in the late stages of design when the model has already been established [10]. At this stage, if the analysis result does not meet the requirements, the design model needs to be reconstructed. There is an urge to consider the real-time at an earlier stage to reduce the gap between requirement and design. (3) The existing analysis approaches are always based on the time-driven mode [11], which is sensitive to the clock in the simulation. Even MARTE [12] provides a precise and sufficient method for modeling a clock, but many factors affecting the accuracy of a clock are only determined at runtime and difficult to be quantitatively described at the design stage.

Real-time systems have the following characteristics: (1) with the wide application of real-time systems, most of them are based on RTOS [13]. As real-time systems have a wide range of application scenarios, the RTOS used is becoming more and more specialized and often applied with the way of customization [9,14] or configuration [15]. With the rise of the microkernel [16], more and more customized RTOSs are based on a microkernel. (2) The real-time in such systems depends on the interactions with other tasks, which are managed by RTOS. The time spent at the system calls during the management of RTOS is directly determined by the hardware in a deployment environment [17-18]. (3) As the functions in

* Corresponding author.
*E-mail address*: xurongfeik@163.com

a task have not been refined (only annotated by the worst-case execution time [19]) at the early stages of design, the design decision for a task is only the priority class [20] at this stage.

As for the above characteristics of the real-time system, an approach of time-oriented modeling and analysis is proposed in this paper to analyze the task time under various design decisions [21] at the early stages of design. Based on the RTOS customized from microkernel, this paper aims to analyze the factors affecting the time from a global perspective and verify the real-time under the various design decisions of user-level services and hardware type adopted by the RTOS as well as the priority settings for a task. Figure 1 illustrates the workflow of this paper: (1) firstly, a high-level design model for the real-time system (RTS) is generated by the time-oriented modeling approach according to the requirements. In this modeling approach, the time-oriented modeling elements and rules for task and RTOS are defined from the static and dynamic perspectives. A criterion for modeling the system under variable design decisions is presented according to the commonality and variability in a system. (2) The time-oriented analysis approach is used to analyze the real-time of the RTS model. In this analysis approach, a timing tree is defined, and its nodes consist of the operating and timing rules driven by an event. Every timing tree in a system is traversed and computed to get the time information of each task under a specific variability.

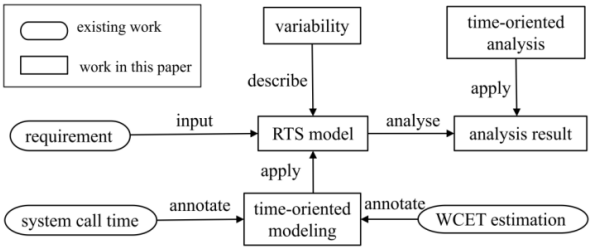


Figure 1. Workflow of this paper

The approach proposed in this paper is applicable to the following two scenarios: (1) for the evolution of an existing system, this approach is used to verify the new design decisions in advance. In this case, the design model is built by reference to the original system, and the system call time and WCET estimation in the model are also obtained from the original system. (2) For a new system, this approach is used to compare various design decisions and choose a better one to guide the design and development in a later stage. In this case, the system call time is obtained by measurement in a real runtime environment [17], and the WCET estimation is evaluated and given by program developers and analysts [19].

The rest of this paper is organized as follows: Section 2 introduces the foundation work. Section 3 presents an approach of time-oriented modeling and analysis for the real-time system. In Section 4, the accuracy and validity of this modeling and analysis approach are evaluated by a real case study. Section 5 presents the related work. Section 6 concludes the paper.

## 2. Background

### *2.1. Microkernel-based RTOS*

With the wide application of real-time systems, they are demanded to be more powerful. Most real-time systems adopt the operating mode based on RTOS, and the RTOSs adopted are often customized based on microkernel, such as QNX and Integrity. A microkernel [16] is the near-minimum amount of software that provides the mechanisms needed to implement an operating system. As a microkernel allows building arbitrary operating system services on top, it adopts the "separation of mechanism and policy" principle [22] and provides some core functionality. An RTOS based on microkernel is described as a 2-tuple $OS_{\mu k} = (M_{\mu k}, S_{um})$, $M_{\mu k}$ is kernel-level mechanism, and $S_{um}$ is user-level service. $M_{\mu k}$ is consistent with the three basic mechanisms of scheduling, inter-task communication, and resource access in microkernel [16]. $S_{um}$ provides tailorable services needed by specific intention for RTOS.

### *2.2. MARTE Modeling Approach*

With the development of real-time systems, the object management group (OMG) proposes a modeling and analysis of real-time embedded systems UML profile: MARTE [12], which defines the foundations for a model-based description of the real-time and embedded systems. Its core concepts are then refined for both modeling and analyzing concerns. MARTE is not applicable to address the question mentioned here in the following two aspects: (1) MARTE provides a precise and sufficient modeling method for clock. For example, the clock rate is described by a stability parameter to reflect the variability of physical time in *ChronometricClocks* Package. However, many factors (such as the stability parameter) are uncertain at this design stage. (2) As for the scheduling analysis and resource modeling, the description of a hardware

platform is adequately considered in MARTE. Aiming to analyze the real-time at very early stages of design, this paper only focuses on the characteristics of platform relevant with time instead of the details.

### *2.3. Event-Driven Mode based on Time Annotation*

At the present time, the traditional hypothesis that the time sampling in a system is fixed is no longer true [23]. In this situation, the event-driven mode has an advantage over the time-driven mode [24], and it makes up for MARTE in describing the time deviation at runtime. As the timing method based on annotation is widely used in the event-driven mode, many studies begin to analyze the execution time of system calls in RTOS under a certain hardware environment by static analysis [25] or test cases [17]. Moreover, most RTOSs also come with information about the average execution time of system calls, and it can be obtained through benchmarks (e.g. Chorus, RTEMS [26]). As the tasks managed by RTOS are event-driven [20], these works provide basic data to make the analysis result more precise.

## 3. Time-Oriented Modeling and Analysis

### *3.1. Time-Oriented Modeling Approach*

A real-time system is usually designed as an RTOS and a set of tasks [9]. The RTOS based on microkernel is composed of a kernel-level mechanism and user-level service together with some relevant system calls. The time spent at system call is determined by the condition of a hardware platform as mentioned above. The tasks are usually viewed as a sequence of functional blocks and some relevant system calls [19] at the early stages of design. The main objects in the real-time system model are concluded as shown in Figure 2. A time-oriented modeling approach is proposed to describe the objects from both static and dynamic perspectives.

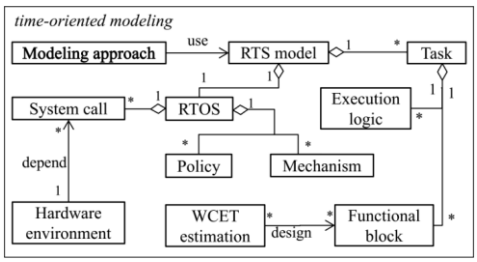


Figure 2. Time-oriented modeling approach

(1) Static View

This modeling approach defines the modeling elements for both RTOS and tasks at the component level from the time aspect to provide a domain-specific modeling for the real-time system.

Time-oriented RTOS is defined as *RTOSto* = (*HR*, *SR*, *KM*, *US*, *SCT*).

- *HR* is the type of hardware provided to the RTOS.
- *SR* is a description of the software resource.
- *KM* is the kernel-level mechanism relevant with time.
- *US* is the user-level service relevant with time.
- *SCT* is the time spent at basic system calls in RTOS.

Given a hardware type, the time of system calls in RTOS is usually fixed [17]. SCT is measured in a real operational environment and used to annotate the basic interfaces in the model, so the analysis result of such a model is closer to the real circumstance.

Before defining the task, we first define the functional block. Every task can be separated into unbreakable functional portions, each of which has its own estimated execution time and is usually described by WCET [19]. As the approach in this paper is applied at the early stages of design, the functions in a task have not been refined yet at that time. To get the maximum unbreakable functional block that only has one single interactive relationship with others, every task is divided into smaller blocks according to Equation (1). This generated functional block is convenient for both time annotation and interaction analysis.

$$(\forall FB_i, FB_j \in RTS \;\&\&\; (i \neq j) \rightarrow NUM(IP(FB_i) \bigcap IP(FB_j)) > 1) \rightarrow DIV(FB_i) \;\&\&\; DIV(FB_j) \quad (1)$$

$FB_i$ and $FB_j$ represent two arbitrary functional blocks in the system, $IP(x)$ represents the interaction point between $x$ and the other functional blocks, $NUM(x)$ is the number of $x$, and $DIV(x)$ denotes the operation of re-dividing the functional block $x$. Functional blocks generated by the above method are the basic interaction units between tasks.

The functional block is defined as follows: *FunctionalBlock* = (*NA*, *TP*, *PDT*, *AT*, *BS*, *CT*).

- *NA* is the name of a functional block.
- *TP* is the type of functional block, and *TP* = (*CB*, *IB*) represents the two cases of whether the functional block involves any interactions with others. *CB* represents the pure compute blocks that do not involve any interaction. *IB* represents the blocks that have at least one interaction with others. There are a wide variety of interaction types according to different operations. As for the kernel mechanism, there are six basic kinds of *IB* that include *ARCon*, *ARCom*, *AS*, *AR*, *SS*, and *SR*, and they represent concurrent resource access, non-concurrent resource access, asynchronous send/receive, and synchronous send/receive respectively.
- *PDT is* the preset WCET of a functional block, and it can be regarded as a constraint solution for implementation of every functional block.
- *AT* describes the activities in a functional block.
- *BS* is the state of a functional block and includes three cases of executed (*ED*), to execute (*TE*), and executing (*EG*).
- *CT* is the constraint for performability of every functional block. As the event-driven mode is adopted, the *CT* for functional block *i* is defined as that the predecessor of *i* in executing sequence (see the following definition *Task*) is completed and the state of the task that includes *i* is running.

According to the definition of the functional block, a task is defined as *Task* = (*TD*, *FBS*, *ES*, *TS*, *TG*).

- *TD* is the description of a task, including ID and priority.
- *FBS* includes the whole functional blocks in a task.
- *ES* represents the executing sequence of these functional blocks.
- *TS* is the task state.
- *TG* is a timer for the task itself and is used to record the current moment when an event triggers.

(2) Dynamic View

The time-oriented modeling rules are proposed to describe the behaviours in a real-time system from a dynamic perspective. For the RTOS, the particularity of modeling its behaviours mainly lies in the actions during its management process. That will be introduced together with the analysis process in the next section. The modeling rules for the behaviours in a task are mainly discussed here.

As the internal details of every functional block have not been determined yet at the early stages of design, only its time can be described by pre-set WCET. Moreover, the scheduling and interaction between tasks are both handled by RTOS. Therefore, the behaviours in a task at this stage mainly involve the executing logic of functional blocks here. The modeling rules for the executing sequence are defined in the form of Backus Normal Form (BNF) in Figure 3. These elements in BNF are defined by reference to the concepts in definition *FunctionalBlock*. The rules are considered from two aspects of resource access and compute. The resource access includes concurrent access and non-concurrent access. The compute includes inter-task operation and pure compute within a functional block. When accessing a concurrent resource, it is defined to access the corresponding synchronous resource first. These rules describe the structure of functional blocks in a task.

① *functional_block*::= {[*access_resource*]|[*compute*]}
② *access_resource*::= {[*access_nonconcurrentResource*]|[*access_concurrentResource*]}
③ *access_concurrentResource*::= <*acess_synchronousResource*> <*doAccess_concurrentResource*>
④ *compute*::= {[*pure_compute*]|[*inter-task_operation*]}

Figure 3. Rules for modeling functional blocks in a task

On the basis of modeling elements and rules, a commonality and variability modeling criterion is proposed in Figure 4. The base model, which consists of functional blocks, execution logic in tasks, and the kernel mechanism, is annotated by WCET estimation of functional block and the time spent at system calls under a given hardware condition, and it is connected

with the variability of upper user-level services and the design scheme for tasks to create a whole model. Such a whole model is capable of modeling the variability without changing the base model, which is helpful to analyze the variability.

① *Build a base model based on the functional blocks and execution logic in tasks as well as the RTOS mechanisms*;
② *Extend the base model by adding user-level services*;
③ *Annotate the extended base model by the system call time and preset WCET, and set the task priority in models to generate a whole model.*

Figure 4. Modeling criterion

### *3.2. Time-Oriented Analysis Approach*

It can be shown that the real-time system model is composed of the base model and the variable part. The variable part can be concluded as three kinds of variability including hardware type, user-level service (described as policy), and task priority, as shown in Figure 5. These three kinds of variability correspond to the aspects of hardware, RTOS, and application respectively. The time spent at system call determined by hardware type and the task priority are both considered in the form of parameters during the analysis, and the variability of user-level policy relevant with time is mainly discussed here.

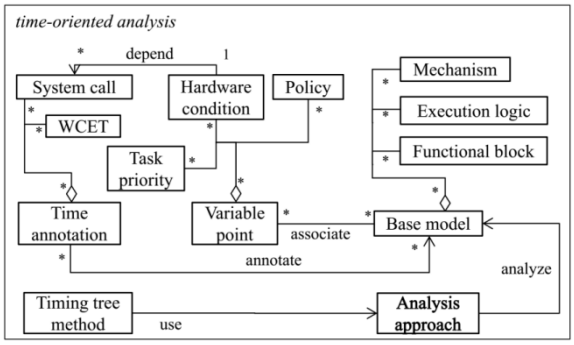


Figure 5. Time-oriented analysis

As mentioned above, the task time is composed of scheduling time among tasks, interaction time between functional blocks, and execution time within the functional block, as shown in Equation (2). This time can be computed by traversing the sequence of functional blocks in every task under the management of RTOS. A tree structure is defined to describe the factors relevant with time for every task in a model, and it is referred to as the timing tree.

$$Task\ Time = Scheduling\ Time + Interaction\ Time + Execution\ Time \quad (2)$$

**Definition 1** (Timing Tree) A timing tree is an unordered tree, and its nodes are provided with properties to describe the objects on nodes. The nodes are defined as *TTNode* = (*Obj*, *Op*, *Arg*), which represents the three kinds of object node, operation node, and parameter node respectively. For every object node, its child nodes include the operation node and the related object node. For every operation node, its child nodes include the operation node required (if any exist), object node, which is an action object of this operation (if any exist), and related parameter node.

- The object node is defined as *Obj* = (*TK*, *FB*) and includes the task and functional block two kinds. For the task node, it is considered the root node, and its child object node is the first functional block in its execution sequence of functional blocks. For every functional block node, its child object node is its descendant functional block in the execution sequence.
- The operation node is defined as *Op* = (*SD*, *IR*) and includes the two kinds of scheduling and interaction. Both kinds of these operations are in the form of mechanism and policy. The operation includes the timing and operating two parts, of which the first part is used to compute time during the task execution and the second part defines the behaviors driven by event for task execution. Every operation node has a scope according to the type of operation. For example, the scope of scheduling operation includes all the tasks in RTOS, and the scope of interaction operation includes the related functional blocks.
- The parameter node is defined as *Arg* = (*SC*, *WC*) and includes the two kinds of time of system call and preset WCET.

The time-oriented analysis approach uses the parameters provided by *Arg* nodes to compute the time of *Obj* nodes according to the rules of *Op* nodes. As for the *Op* node, a formula is defined as follows to describe its behaviours. The

formula is used as a paradigm to define the event-driven rules and represents that "given a certain condition or trigger event, task *T* in some a state will execute a series of actions".

$$State\ (T)\ \&\&\ (Condition\ (T)\ ||\ Event\ (T)\ ||\ (Condition\ (T)\ \&\&\ Event\ (T)))\ \rightarrow\ Actions\ (T) \quad (3)$$

RTOS is a complex system and has a wide range of functions depending on the implementation of policy. As the behaviors of these policies are triggered by an event, all of them can be easily described as the form of Equation (3). As the basic mechanisms in microkernel provide fundamental functions of RTOS and will be used in all cases, the user-level policies based on these mechanisms are taken as a representative here to illustrate this analysis approach. The other policies can be easily analyzed by the same way as well.

(1) Construct the timing tree

Firstly, a timing tree based on these mechanisms is built to describe the basic structure of a real-time system. Figure 6 is a sketch map only describing the main objects in this timing tree without considering the hierarchical relation. Its root node represents a task with a timer to store the time (see *Ti* node in Figure 6). The non-leaf nodes can be regarded as the two kinds of scheduling between tasks (*TS*) and execution within the task (*TE*). The task execution includes two cases of between and within functional blocks. For the scheduling node (*SD*), it has a property of state to represent the scheduling state of this task. The task execution is composed of a set of operations and functional blocks. The operations include two kinds of resource access (*RA*) and inter-task communication (*ITC*) according to the basic mechanisms in microkernel. The functional block without any interaction with others is described as a pure compute node (*CB*). Every non-leaf node has the operating and timing rules corresponding to its type except the pure compute node, whose execution is within a functional block and does not involve any rules. The leaf nodes include the system call time (*sct* node) and pre-set WCET (*pdt* node).

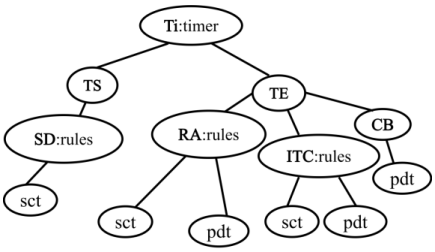


Figure 6. Sketch map of the objects in timing tree

Constructing a complete timing tree from the time-oriented model follows the rules in Figure 7. It can be found that the scheduling operation is at the granularity of task and the interaction operation is at the granularity of a functional block. Based on the operating and timing rules defined below, the visit rules for timing tree are as follows: (1) for every object node, it is visited by in-order traversal, and its time is computed by the rules of its child operation node. After computing the time, its child object node is turned to visit. (2) For every operation node, it is visited by post-order traversal. The parameters of its child nodes are used to operate the timing behaviours. (3) After visiting a node, the timer on the root node should be updated. Finally, the time generated on the root node is the total execution time for a task.

① *Generate a root node with a timer according to the task.*
② *Generate a child operation node of scheduling mechanism for the root node. If the mechanism includes any user-level policy, continue to generate a corresponding child operation node for the mechanism node. Define operating and timing rules for every operation node.*
③ *For the root node, generate a child object node of the first functional block in execution sequence. For the functional block node, generate a child object node of its descendant functional block in execution sequence.*
④ *If the functional block node has an interaction with others, generate a child operation node of interactive mechanism for it. If the mechanism includes any user-level policy, continue to generate a corresponding child operation node for the mechanism node. Define operating and timing rules for every operation node.*
⑤ *If the operation node includes other action objects except for its parent object node, continue to generate corresponding child object nodes for the operation node.*
⑥ *Generate a leaf node of pre-set WCET for the functional block nodes, and generate a leaf node of relevant system call time for the operation.*

Figure 7. Rules to construct a timing tree

(2) Define the operating and timing rules

According to the basic mechanisms in microkernel, the user-level policies are all based on kernel-level events during the scheduling process. In this section, the operating and timing rules are defined according to the state of scheduling and used as the basic rules to visit the timing tree.

Task scheduling in the kernel mechanism only considers the most basic task states, as shown in the above Figure 8. For the three basic states (*Running*, *Ready*, and *Blocking*), formula (3) is used to define the rules for scheduling operations under each state.

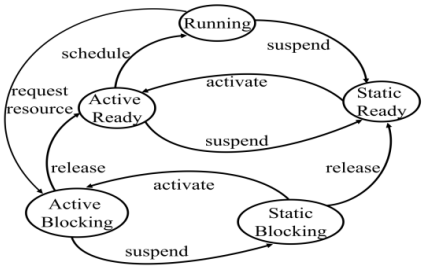

Figure 8. Task state transition diagram

Principle (1): rules for *Blocking* state

- $B(T)\ \&\&\ (TS(T) = FALSE)\ \&\&\ G(T) \rightarrow RD(T)\ \&\&\ C(T)\ \&\&\ M(T)$
- $B(T)\ \&\&\ (TS(T) = TRUE) \rightarrow SB(T)\ \&\&\ C(T)\ \&\&\ M(T)$
- $SB(T)\ \&\&\ BA(T) \rightarrow RM(T)\ \&\&\ C(T)\ \&\&\ M(T)$
- $SB(T)\ \&\&\ G(T) \rightarrow SRD(T)\ \&\&\ C(T)\ \&\&\ M(T)$

$B(T)$, $SB(T)$, $RD(T)$, and $SRD(T)$ represent that task $T$ is in the state of blocking (active blocking by default), static blocking, ready (active ready by default), and static ready respectively. $TS(T)$, $BA(T)$, and $RM(T)$ represent that $T$ is suspended, activated, and resumed respectively. $G(T)$, $C(T)$, and $M(T)$ represent the actions of acquiring a resource, changing the task state, and modifying the task time respectively. As the time is represented by annotation, there is a need to compute the current moment for a task at each step. For example, the second rule represents that task $T$ is suspended and its state is changed from blocking to static blocking, so $M(T)$ modifies the current time of $T$ by adding the time spent at system call of suspending action. This principle describes the operation rules for four events of acquiring a resource and suspending the task after $T$ is blocked and activating a task and acquiring the resource when $T$ is in the state of static blocking.

Principle (2): rules for *Ready* state

- $RD(T)\ \&\&\ (TS(T) = FALSE)\ \&\&\ (T(T) = MIN)\ \&\&\ (P(T) \geq CP) \rightarrow RN(T)\ \&\&\ C(T)\ \&\&\ M(T)$
- $RD(T)\ \&\&\ (TS(T) = FALSE)\ \&\&\ (T(T) = MIN)\ \&\&\ (P(T) < CP) \rightarrow RD(T)\ \&\&\ C(T)\ \&\&\ M(T)$
- $RD(T)\ \&\&\ (TS(T) = TRUE) \rightarrow SRD(T)\ \&\&\ C(T)\ \&\&\ M(T)$
- $SRD(T)\ \&\&\ RA(T) \rightarrow RM(T)\ \&\&\ C(T)\ \&\&\ M(T)$

$T(T)$, $P(T)$, and $CP$ represent the current time of task $T$, the priority of $T$, and the CPU priority respectively. The CPU priority equals that of the task that is obtaining the CPU at this moment. $RN(T)$ and $RA(T)$ represent that $T$ is running and activated respectively. The meaning of other symbols is the same as above. This principle describes the operation rules for four events of preempting CPU, preempting unsuccessfully, suspending, and activating the task in the ready state. When more than one task is ready, it should judge whether $T$ is the earliest ($T(T) = MIN$), as a different task has its own "clock" (the timer).

Principle (3): rules for *Running* state

- $RN(T)\ \&\&\ (TS(T) = FALSE)\ \&\&\ (RQ(T) = FAIL) \rightarrow B(T)\ \&\&\ C(T)\ \&\&\ M(T)$
- $(RN(T)\ \&\&\ (TS(T) = FALSE)\ \&\&\ (\forall t \in CRT(T)) \rightarrow (\exists\, t \in CRT(T) \rightarrow PRP(T) = FALSE)\ \&\&\ (FH(T) = FALSE)) \rightarrow RN(T)\ \&\&\ C(T)\ \&\&\ M(T)$
- $(RN(T)\ \&\&\ (TS(T) = FALSE)\ \&\&\ (\forall t \in CRT(T)) \rightarrow (\exists\, t \in CRT(T) \rightarrow PRP(T) = FALSE)\ \&\&\ (FH(T) = TRUE))$

$\rightarrow C(T)$ && $M(T)$

- $(RN(T)$ && $(TS(T) = FALSE)$ && $(\forall t \in CRT(T)) \rightarrow (\exists t \in CRT(T) \rightarrow PRP(T) = TRUE)) \rightarrow RD(T)$ && $C(T)$ && $M(T)$
- $RN(T)$ && $(TS(T) = TRUE) \rightarrow SRD(T)$ && $C(T)$ && $M(T)$

$CRT(T)$ represents the remaining time of task $T$ that still needs to execute. $RQ(T)$, $PRP(T)$, and $FH(T)$ represent the actions of acquiring a resource, being preempted by others, and executing completely respectively. *PRP* is a variation point that carries out the preemptive action based on specific user-level scheduling policy. This principle describes operation rules for five events of acquiring a resource, running normally, executing completely, being pre-empted, and suspending when the task is in the running state. As there is no unified clock in the system, it needs to compute the remaining time that still needs to execute for every task before running. The tasks should be judged whether to be preempted for every tick during the running process.

## 4. Case Study

### *4.1. Experimental Objects*

In this section, a real-life aircraft landing control system (ALCS) [27] was used as a case study to evaluate the modeling and analysis approach. Firstly, we constructed the ALCS model with our modeling approach. Secondly, we focused on evaluating the accuracy and validity of our analysis approach, which was based on the modeling approach. The following two experiments were conducted: (1) compare the analysis results of tasks in the ALCS model with the run time of each task in the original system under the real environment; (2) compare the analysis results of this model under two different scheduling algorithms to verify the capacity of analyzing a new policy.

### *4.2. Experimental Setup*

The ALCS used here was a real-life case of an engineering institute. This system is a real-time critical system with a high demand for security and time. As seL4 is the formally-verified operating system kernel [28], it can be used as the ideal platform for this system. However, seL4 does not consider the real-time in the system. The designer was to verify the real-time of tasks when adopting a new policy on the seL4.

ALCS is composed of eight tasks including data reception (DRT) and processing (DPT), go-around control (GAT), failure monitoring (FMT), time synchronization (TST), operation record (ORT), window GUI (WGT), and business GUI (BGT) handling (as shown in Table 1). The types of functional blocks described different interactive relationships between blocks (see definition *FunctionalBlock*) and were concluded as follows: a semaphore for the time variable in the system was used by the *TimeSynchronization* functional block in task *TST* and *TimeSignalSend* in *WGT*. *SendData* in *DRT* and *ReceiveData* in *DPT* composed a pair of synchronous communication endpoints. *TimeSignalSend* in *WGT* and *TimeSignalReceive* in *BGT* composed another pair of synchronous communication endpoints. *RecordInfoSend* in *FMT and ReceiveRecordData* in *ORT* composed a pair of asynchronous communication endpoints.

For the hardware type, Cortex-A8 with ARM architecture was used. The time for typical seL4 system calls was analyzed in this operational environment by NICTA [3]. The pre-set WCET of every functional block in the model was measured by testing the run time of the corresponding code portion in the original system. Due to space constraints, the *DataProcessing* task and its *NetworkDataReception* functional block are only put here (in Figure 9) to typically represent the modeling of the task and the functional block.

Table 1. Description of tasks

| Task | Priority | Type of functional blocks (see definition of *functional block* in sect. 3.1) |
|---|---|---|
| DRT | 159 | Type of *SendData* is *SS* and that of others is *CB*. |
| DPT | 154 | Type of *ReceiveData* is *SR* and that of others is *CB*. |
| TST | 172 | Type of *TimeSynchronization* is *ARCon* and that of others is *CB*. |
| GAT | 161 | Type of all functional blocks is *CB*. |
| FMT | 166 | Type of *RecordInfoSend* is *AS* and that of others is *CB*. |
| ORT | 163 | Type of *ReceiveRecordData* is *AR* and that of others is *CB*. |
| WGT | 153 | As the *TimeSignalSend* has two types of interaction, which are *SS* and *ARCon* respectively. Type of others is *CB*. |
| BGT | 150 | Type of *TimeSignalReceive* is *SR* and that of others is *CB*. |

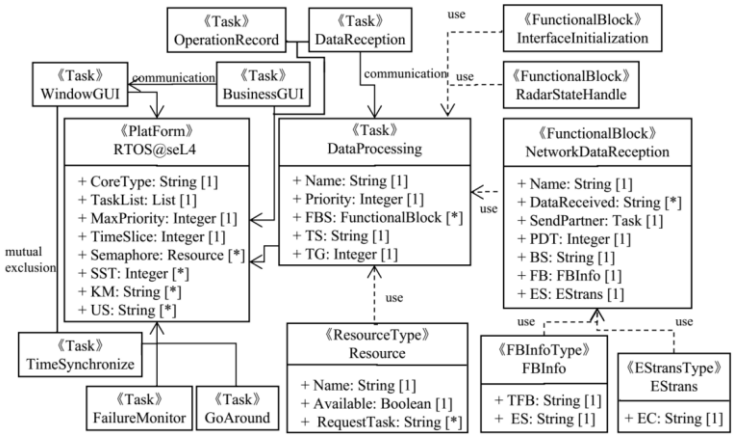


Figure 9. Partial model of ALCS

For experiment (1), the parameters in the ALCS model all came from the original system executed in the real environment. The analysis results of this model were compared with the run-time of the original system to verify whether the rules in this analysis approach could correctly reflect the real circumstances. For experiment (2), the earliest deadline first (EDF) [29] scheduling algorithm was taken as an alternative scheduling algorithm, and its time performance was compared with that of the original scheduling algorithms in seL4.

### 4.3. *Result and Discussion*

The experimental results are shown in Figures 10 and 11.

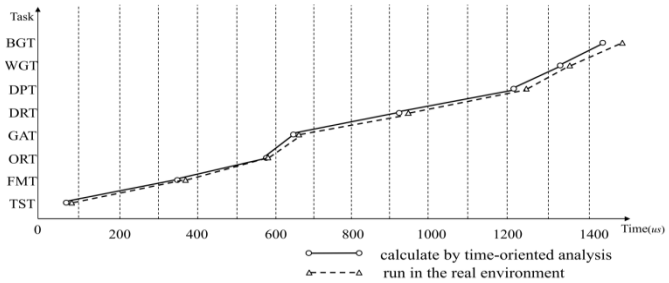


Figure 10. Results of experiment (1)

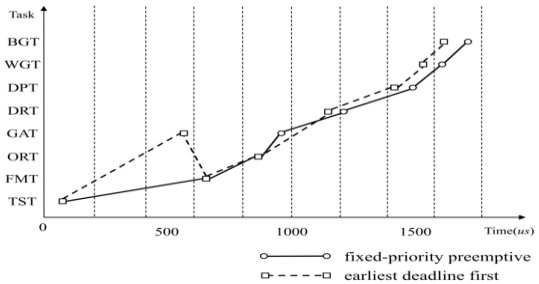


Figure 11. Results of experiment (2)

The experimental results in Figure 10 show that the time analysis of this proposed approach was almost the same with the run time in a real environment. This was because the execution time of every functional block and system call time annotated in the model came from the real environment, so they could reflect the true circumstances. A slight deviation could be found between the analysis results with the real case in Figure 10. As the time used to annotate the model was measured by statistics, the experimental data used here only reflected a few parts of the statistics; it was normal for a slight deviation to emerge. In a word, it could be considered that the analysis results had an acceptable accuracy.

From the results in Figure 11, it could be found that the EDF scheduling policy had a better characteristic of real-time and a higher CPU utilization for this case. As the *GAT* task had an earlier deadline in this scenario, the on-the-fly priority

assignment of EDF scheduling was helpful to ensure the real-time of urgent tasks. This analysis approach could evaluate a new policy by quantitatively comparing its time performance with that of the existing policy. By defining the variation point in rules and adding an operation node in timing tree for the new policy, this analysis approach was convenient to analyze new decisions at the early stages of design.

*Threats to validity*. (1) The accuracy of this analysis approach is directly based on the accuracy of annotated time. Various factors (such as abnormal problems that may arise during the runtime) should be considered further during the measurement of system call time. (2) As this approach is time-oriented and its range of application is the real-time systems based on RTOS customized from microkernel, it only focuses on the time analysis of such systems without considering the details of these functionalities, which are irrelevant with time as well as other non-functionalities.

## 5. Related Works

The time problem is an important issue for real-time systems. There have been many studies during the entire life cycle of such software. As the real-time system is complex and its time is influenced by many factors, these existing methods focused on different aspects in various stages. In this paper, this time-oriented approach was compared with them from the following perspectives: (1) the stage when this problem was considered; (2) the study object and modeling method; (3) the analysis granularity and analysis method. The comparison of some typical approaches is shown in Table 2.

Table 2. Comparison with previous works

| Literatures | Stage | Study object | Modeling method | Analysis granularity | Analysis method |
|---|---|---|---|---|---|
| [03] | After coding | Platform | — | System call | Static analysis |
| [02] | PIM to PSM | Platform and task | UML | Basic structure/ Task | Model analysis based on timer |
| [30] | After design | Platform and application | SystemC | Basic structure/ Task | Time event modeling and analysis based on annotation |
| [31] | Before design | Task | Graph-based modeling | Task | Approximate analysis based on annotation |
| [08] | Before design | Platform and task | MARTE | Workload and strategy/ Functional block | Model analysis based on timer |
| Our approach | Before design | Platform and task | Time-oriented modeling | System call/ Functional block | Operating and timing rules based on annotation |

As for (1), there were many studies at the stages of coding and design. In the coding stage, static analysis or program testing was often used. For example, literature [3] researched the WCET of source program by using static analysis after the software was developed. These studies were only suitable for analyzing the performance of a platform that already existed. As real-time systems had a characteristic of high demand, most of the research was in the design stage. These included the early and later two cases for the design stage. Literature [2] studied how to conserve the time demand of design models on a target execution platform at the late stages of design. Literature [30] verified the real-time of the system model at the late stages of design. To get rid of model reconstruction, literature [8,31] analyzed the real-time of design solution at the early stages of design.

As for (2), there were two primary study objects of application and platform. Among these studies, literature [3] only considered the platform, and literature [31] only considered the task in an application. As the time performance in real-time systems had a great relationship with the platform environment [6], there were many works that considered the task time together with RTOS [2,8,30]. For different stages of design, SystemC and UML were the two most common modeling languages [2,30].

As for (3), most studies could be concluded as two analysis granularities of the task [2,30,31] and functional block [8] on the basis of a platform. For the platform, there were also two analysis aspects of functional structure [2,8,30] and system call [3]. The analysis based on system call did not need to consider the details of this platform and regarded the platform as a “black box”. The timing mode of these analysis methods included two kinds of logical timer [2,8] and time annotation [30-31]. For the shortcomings of logical timers mentioned above, the annotation method based on the time spent at system call was more accurate.

Though literature [31] was based on time annotation and worked at the early stages of design, it only made an approximate analysis of task time. Our approach provides a time-oriented modeling and analysis methodology for time analysis at the early stages of design. It only needs to annotate the time characteristics of the functional block and the system call without considering the internal implementation details. The time annotation at the early stages of design as well as the

rules driven by an event are the two obvious characteristics of our approach, and they make it more suited to judge the real-time accurately and quickly.

## 6. Conclusions

As real-time systems always have a characteristic of high demand for time, a time-oriented modeling and analysis approach for the real-time system based on RTOS at the early stages of design is presented to evaluate the time performance under various design decisions. This approach is convenient to analyze new design decisions in advance before their implementation in the program, which is helpful to improve the real-time in a system. The modeling approach is time-oriented, the analysis approach is based on time annotation and event-driven mode, and they can be used together to quantitatively evaluate the time performance. This approach is proven to be effective according to a real-life aircraft landing control system. At present, this approach is only suitable for the situation that these real-time systems are based on the RTOS customized from microkernel. In the future, we will improve the capacity of time-oriented modeling and analysis for more customized modes of RTOS to extend its application range.

## Acknowledgements

This paper is supported by the National Natural Science Foundations of China (No. 61672078, 61732019). The authors would also like to thank the reviewers for their valuable comments on this article.

## References

1. D. Hatley and I. Pirbhai, “Strategies for Real-Time System Specification,” Addison-Wesley, 2013
2. R. Mzid, C. Mraidha, J. P. Babau, and M. Abid, “A MDD Approach for RTOS Integration on Valid Real-Time Design Model,” in *Proceedings of 38th Euromicro Conference on Software Engineering and Advanced Applications*, pp. 9-16, IEEE, 2012
3. B. Blackham, Y. Shi, S. Chattopadhyay, A. Roychoudhury, and G. Heiser, “Timing Analysis of a Protected Operating System Kernel,” in *Proceedings of 32nd Real-Time Systems Symposium* (*RTSS*), pp. 339-348, IEEE, 2011
4. A. G. Kleppe, J. Warmer, and W. Bast, “The Model Driven Architecture: Practice and Promise,” Addison-Wesley Professional, 2003
5. I. W. Soares, L. T. W. Agner, P. C. Stadzisz, and J. M. Simão, “Modeling of Embedded Software on MDA Platform Models,” *Journal of Computer Science & Technology*, Vol. 12, No. 3, pp. 133-139, 2012
6. J. Schneider, “Why You Can't Analyze RTOSs Without Considering Applications and Vice Versa,” in *Proceedings of 2nd WS Worst-Case Execution-Time Analysis*, pp. 1-4, 2002
7. M. Lv, N. Guan, Y. Zhang, Q. Deng, G. Yu, and J. Zhang, “A Survey of WCET Analysis of Real-Time Operating Systems,” in *Proceedings of 2009 International Conference on Embedded Software and Systems* (*ICESS'09*), pp. 65-72, IEEE, 2009
8. C. Mraidha, S. Tucci-Piergiovanni, and S. Gerard, “Optimum: A MARTE-based Methodology for Schedulability Analysis at Early Design Stages,” *ACM SIGSOFT Software Engineering Notes*, Vol. 36, No. 1, pp. 1-8, 2011
9. 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 *Proceedings of 2015 IEEE International Conference on Smart City/SocialCom/Sus -tainCom* (*SmartCity*), pp. 87-94, IEEE, 2015
10. J. Zimmermann, S. Stattelmann, A. Viehl, O. Bringmann, and W. Rosenstiel, “Model-Driven Virtual Prototyping for Real-Time Simulation of Distributed Embedded Systems,” *7th IEEE International Symposium on Industrial Embedded Systems* (*SIES'12*), pp. 201-210, IEEE, 2012
11. E. D. Jensen, C. D. Locke, and H. Tokuda, “A Time-Driven Scheduling Model for Real-Time Operating Systems,” *RTSS*, Vol. 85, pp. 112-122, 1985
12. “UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems,” (http://www.omg.org/spec/MARTE/ Current, accessed September 10, 2018)
13. “Real-Time Operating System,” (https://en.wikipedia.org/wiki/Real -time_operating_system, accessed September 10, 2018)
14. K. Abe, M. Yoo, and T. Yokoyama, “Customization of a Real-Time Operating System Scheduler with Aspect-Oriented Programming,” in *Proceedings of World Academy of Science, Engineering and Technology*, *World Academy of Science, Engineering and Technology* (*WASET*), Vol. 71, pp. 1390, 2012
15. G. Macher, M. Atas, E. Armengaud, and C. Kreiner. “A Model-based Configuration Approach for Automotive Real-Time Operating Systems,” *SAE International Journal of Passenger Cars-Electronic and Electrical Systems*, Vol. 8, No. 2015-01-0183, pp. 270-277, 2015
16. “Microkernel,” (https://en.wikipedia.org/wiki/Microkernel, accessed September 10 2018)
17. D. P. B. Renaux, R. E. Goes, and R. R. Linhares, “Performance Characterization of Real-Time Operating Systems for Systems-on-Silicon,” in *Proceedings of 12th Brazilian Workshop on Real-Time and Embedded Systems*, pp. 63-74, 2010
18. C. Brink, E. Kamsties, M. Peters, and S. Sachweh, “On Hardware Variability and The Relation to Software Variability,” in *Proceedings of 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pp. 352-355, IEEE, 2014
19. 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 Journal on Embedded Systems*, No. 1, pp. 1-17, 2008

20. Y. Wang, C. F. Ngolah, G. Zeng, P. C. Y. Sheu, C. P. Choy, and Y. Tian, "The Formal Design Model of a Real-Time Operating System (RTOS+): Conceptual and Architectural Frameworks," *International Journal of Software Science and Computational Intelligence* (*IJSSCI*), Vol. 2, No. 2, pp. 105-122, 2010
21. T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki. "Variability Modeling in the Systems Software Domain," Generative Software Development Laboratory, University of Waterloo, Technical Report, 2012
22. "Separation of Mechanism and Policy," (https://en.wikipedia.org/wiki/Separation_of_mechanism_and_policy, accessed September 10, 2018)
23. K. J. Åström and B. Bernhardsson, "Comparison of Riemann and Lebesque Sampling for First Order Stochastic Systems," in *Proceedings of 41st IEEE Conference on Decision and Control*, Vol. 2, pp. 2011-2016, IEEE, 2002
24. P. Tabuada, "Event-Triggered Real-Time Scheduling of Stabilizing Control Tasks," *IEEE Transactions on Automatic Control*, Vol. 52, No. 9, pp. 1680-1685, 2007
25. T. Sewell, F. Kam, and G. Heiser, "Complete, High-Assurance Determination of Loop Bounds and Infeasible Paths for WCET Analysis," in *Proceedings of 2016 IEEE Real-Time and Embedded Technology and Applications Symposium* (*RTAS*), pp. 1-11, IEEE, 2016
26. "RTEMS C Users Guide," On-Line Applications Research Corporation (OAR), 2011
27. A. A. Lambregts and R. Hansen, "Aircraft Landing Control System," U.S. Patent No. 4357661, Washington, DC: U. S. Patent and Trademark Office, 1982
28. G. Klein, K. Elphinstone, G. Heiser, and J. Andronick, "SeL4: Formal Verification of an OS Kernel," in *Proceedings of ACM SIGOPS 22nd Symposium on Operating Systems Principles*, pp. 207-220, ACM, 2009
29. J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo, "Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms," Springer Science & Business Media, 2012
30. R. Lee, K. Abdel-Khalek, S. Abdi, and F. Risacher, "Early System Level Modeling of Real-Time Applications on Embedded Platforms," in *Proceedings of 14th International Symposium on Quality Electronic Design* (*ISQED*), pp. 558-565, IEEE, 2013
31. N. Guan, C. Gu, M. Stigge, Q. Deng, and W. Yi, "Approximate Response Time Analysis of Real-Time Task Graphs," in *Proceedings of 2014 IEEE Real-Time Systems Symposium* (*RTSS*), pp. 304-313, IEEE, 2014

**Rongfei Xu** is currently pursuing a Ph.D. in the School of Computer Science and Engineering at Beihang University. His research interests include modeling and analysis of real-time systems.