Timing Analyzing for Systems with Task Execution Dependencies

Yue Lu[1], Thomas Nolte[1], Iain Bate[2], and Christer Norström[1]
[1]Mälardalen Real-Time Research Centre (MRTC), Västerås, Sweden
[2]Department of Computer Science, University of York, York, YO10 5DD
{yue.lu, thomas.nolte, christer.norstrom}@mdh.se, iain.bate@cs.york.ac.uk

Abstract

This paper presents a novel approach to timing analysis of complex real-time systems containing data-driven tasks with intricate execution dependencies. Using a system model inspired by industrial control systems, we show how the execution time of tasks can be represented as a mathematical expression instead of a single numeric value. Next, based on this more detailed modeling, we introduce a concrete process of formally obtaining the exact value of both Worst-Case Execution-Time (WCET) and Worst-Case Response-Time (WCRT) of tasks by using upper-part binary search and TIMES (a timed model checker). Finally, in order to show the potential of the proposed approach, we apply it to a model created from a real robotic control system for which the traditional way of obtaining a WCET estimate (through static WCET analysis) on tasks for usage in basic RTA is not appropriate. Our results indicate a significant reduction of pessimism when compared to basic RTA using WCET estimates on tasks given by a basic assumption.

1 Introduction

To date, most existing embedded real-time software systems have been developed in a traditional code-oriented manner, i.e., making extensive use of legacy software. Many such systems are maintained over extended periods of time, sometimes spanning decades, during which the systems become larger and increasingly complex. The result is that these systems are difficult and expensive to maintain and verify. Some existing complex industrial embedded software systems consist of millions of lines of C code. These systems are typically based on periodic tasks, executed on a single processor under Fixed- Priority Preemptive Scheduling (FPPS) and stringent real-time constraints. Examples of such systems include the aircraft engine controller in [1] and the robotic control systems developed by ABB [2]. Contrary to the assumption in most real-time theory, i.e., independent tasks in the analysis model, tasks exhibit strong temporal dependencies, e.g., asynchronous message-passing and globally shared state variables used for dispatching to the control branches that vary task execution time radically.

One desirable approach to avoid timing-related errors in such complex systems is to use schedulability analysis methods, such as Response-Time Analysis (RTA) [3]. Nevertheless, RTA (and most other schedulability analysis techniques), although providing a prediction about timing behavior of task response times in worst-case scenarios, rely on the existence of a fixed Worst-Case Execution-Time (WCET) of the tasks. Correspondingly, the quality of the analysis is directly correlated to the quality of the WCET estimates. Unfortunately, without having detailed runtime information about task execution dependencies, using static WCET analysis to obtain a single numeric WCET estimate on tasks in such complex systems is not feasible. Let us consider the following simplified example in Figure 1, taken from an industrial robotic control system. In this example, a task reads all messages buffered in a message queue and processes them accordingly.

```
        msg = recvMessage(MyMessageQueue);
        while (msg != NO_MESSAGE){
            process_msg(msg);
            msg = recvMessage(MyMessageQueue);
        }
```

Figure 1. Iteration-loop wrt. message passing

Using static WCET analysis to obtain the WCET estimate on the task is not feasible in this example, as the variable `msg` can be changed from outside of the task due to potential preemptions caused by tasks with a higher significant priority. Using program annotation[1] concerning the number of messages received by the task might remedy the situation, but it may be error-prone and time-consuming. Consequently, without making any assumptions on the upper bound of the variable `msg`, basic RTA cannot be applied to the systems with this type of task behavior.

An alternative approach is to use simulation-based methods that sample the state space of response times. The first type of simulation technique to use is Monte Carlo simulation, which can be described as keeping the highest result from a set of randomized simulations. Several frameworks already exist in this realm, such as the commercial tool *VirtualTime* [4] and the academic tool *ARTISST* [5]. However, the main drawback of using Monte Carlo simulation is the low state-space test coverage, which subsequently decreases the confidence in the

[1]Program annotation is the information manually provided by a programmer.

results of finding rare worst-case scenarios. The other category is to apply an optimization algorithm (e.g., a (meta)heuristic search algorithm) on top of Monte Carlo simulation, as in [6] and [7], which yields substantially better results, i.e., tighter lower bounds of the WCRT estimation.

Another interesting approach features the use of stochastic task execution times in RTA of hybrid task sets in priority-driven soft real-time systems [8] and schedulability analysis [9]. Nevertheless, this approach currently does not allow for execution dependencies between tasks in the analysis. More recently, [10] presented one promising research concerning the use of statistical-based response time analysis of complex systems containing the task execution dependencies as introduced above, based on Extreme Value Theory [11]. The proposed method has the potential of providing a tighter upper bound of the WCRT estimation when compared to basic RTA, especially regarding the fact that basic RTA can not easily be applied.

In this paper, we present a novel approach to timing analysis of complex real-time systems containing data-driven tasks. Our approach combines a set of analysis methods traditionally inherent in different domains, extending the work presented in [12]. Specifically, the contributions of this paper are three-fold:

- We propose a system model depicting a detailed behavior of data-driven tasks concerning intricate task execution dependencies such as asynchronous message-passing and globally shared state variables between tasks, and we discuss the necessity of having such a new analysis model.

- We highlight a way of representing the WCET of each task as a symbolic formula which is then maximized by using the bounds of the adhering parameters derived from inter-task analysis at system-level, i.e., tasks are not analyzed in isolation.

- We propose a novel idea of formally obtaining WCET and WCRT values of tasks by using upper-part binary search [13] and a model checker, TIMES, after performing a semantic-preserving model transformation between different analysis models as presented in [14]. This also highlights how model-driven techniques can be used to assess relevant issues in the real-time realm.

In our evaluation work, we show that the proposed approach can find the exact WCET and WCRT of tasks in the model, yielding less pessimistic results compared with the WCET estimates on tasks given by a basic assumption, and basic RTA in [15][2].

[2]Basic RTA [15] is also called classical RTA or traditional RTA in this paper.

The remaining part of the paper is organized as follows: Section 2 firstly presents the necessity of using a new system model to analyze the target system. Secondly, we propose a system model where the WCET of each data-driven task is represented as an expression containing parameters. Then we introduce our prior work on extracting models from real systems, and we show how the modeling language is used in practice. Section 3 firstly gives the problem definition, then highlights how to perform formal timing analysis in TIMES using upper-part binary search, after a semantic-preserving model transformation has been conducted. Section 4 firstly describes the implementation details of our testbed and proposed toolchain, and then it introduces the evaluation performed on a model taken from a real robotic control system. Section 5 and Section 6 present the scalability of the method and related work respectively, and finally, Section 7 concludes the paper and discusses future work.

2 System Model

In this section, we firstly discuss the necessity of using a new system model to perform timing analysis of the target system containing intricate execution dependencies between data-driven tasks. Then we present the proposed system model and its corresponding task execution-time modeling, and finally, we introduce a brief view of our prior work on model extraction including a modeling language used in practice.

Our targeted industrial control systems contain a number of tasks communicating via asynchronous message-passing, sharing Globally Shared State Variables (GSSVs) used for dispatching the control branch, FPPS, on a single processor. The adhering task execution time varies dramatically due to the number of messages consumed or sent by tasks, and the value of GSSVs impacting the selection of control structures in tasks.[3] To perform RTA on such systems, basic RTA uses a task model which assumes that task-level WCET estimates are known, in terms of single numeric values. However, unless program annotation concerning the number of messages consumed or sent by tasks and the value of GSSVs is given, the application of standard static WCET analysis to obtain the WCET estimates of tasks is not feasible. The reason is that the relevant upper bounds of the number of messages and the value of GSSVs are considered to be infinite due to the fact that they can be changed by other tasks. On the other hand, manual annotation may be error-prone (because of lack of detailed runtime information which is the typical case for the systems with above intricate execution dependencies) and far less efficient. Therefore, to obtain a WCET estimate on a single task in the referred system is not appropriate. As shown in Figure 2, `Iteration Loops` (concerning the number of messages) and `GSSVs` are in black with the indication of raising the mentioned

[3]There are no dependencies between the number of messages consumed or sent by tasks and the value of GSSVs.

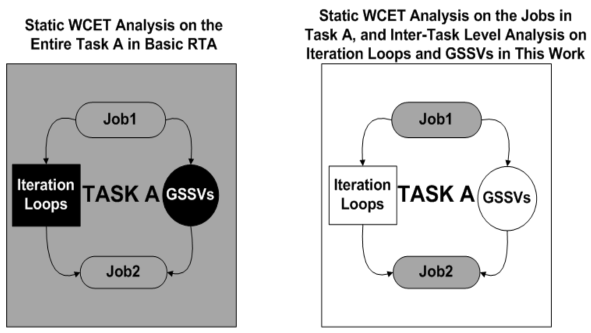
Static WCET Analysis on the Entire Task A in Basic RTA
Job1
Iteration Loops
TASK A
GSSVs
Job2
Static WCET Analysis on the Jobs in Task A, and Inter-Task Level Analysis on Iteration Loops and GSSVs in This Work
Job1
Iteration Loops
TASK A
GSSVs
Job2

Figure 2. Standard static WCET analysis on the entire TASK A and standard static WCET analysis on the jobs in TASK A. The parts in gray denote that they are under static WCET analysis.

problems when applying standard static WCET analysis to obtain the WCET estimate on the entire task. Further, it is necessary to develop a new system model which could describe a more detailed behavior concerning data-dependent task execution time. Such the detailed behavior modeling is not considered in existing RTA and most other schedulability analysis methods.

In this paper, we propose a system model in which the WCET of tasks is represented as a symbolic formula centering around the number of messages in the buffers consumed or sent by a task and the value of the GSSVs used in selecting control branches. A WCET dependent on external context is often referred to as a parametric WCET [16]. Moreover, the system model uses job-level[4] WCET estimates, which make static WCET analysis feasible as there will be no execution dependencies inside jobs. More important, these task WCET expressions can be maximized by using the bounds of the adhering independent parameters that are obtained by using (exhaustive search) inter-task analysis performed at system-level. Consequently, such WCET estimates can also significantly reduce the pessimism in RTA, when compared to basic RTA using WCET estimates with a basic assumption. As shown in Figure 2, `Iteration Loops` and `GSSVs` are in white, indicating that the problems introduced previously can be solved by using the analysis method (to be introduced in Section 3) on the proposed system model.

Hence, the system model S contains a set of non-blocking tasks, each of which consists of n *jobs*, where $n \in \mathbb{N}$. Each deadline-constrained task τ_i is a tuple $\tau_i(T_i, C_i^p, D_i, O_i, J_i, P_i)$, where T_i is task period with maximum jitter J_i, constant offset O_i and a priority P_i, C_i^p is the WCET expression as a function of b buffers (i.e., $U_{i,1}, ..., U_{i,b}$) and g GSSVs (i.e., $V_{i,1}, ..., V_{i,g}$) associated with task τ_i and execution time on jobs, D_i is the relative deadline ($max(C_i^p) \leq D_i \leq T_i$). The execution of task τ_i is divided into two types of sections: firstly, a *non-volatile* (NV) section in which there is no input buffer and GSSV, and secondly, a *volatile* (V) section containing either one buffer, or a GSSV. In both sections, preemption caused by higher priority tasks is allowed.

[4]A task consists of a sequence of jobs.

2.1 WCET Expression of Data-driven Tasks with Execution Dependencies

Each NV section consists of h jobs $j_{i,x}$ in task τ_i, where x is in the range of $[1, h]$. The WCET of the job $j_{i,x}$ is represented as $C(j_{i,x})$, and practically, such a value can be obtained by using either static WCET analysis (which is safe but more pessimistic compared to the exact WCET) that is used in applications with hard real-time constraints, or through dynamic WCET estimates based on measurements with probability distribution when dealing with applications with soft real-time constraints. In this work, such WCETs of jobs are obtained by using static WCET analysis, in terms of a single numeric value for each task. Further, the WCET of a NV section containing h jobs is expressed as follows:

$$C_{i,nv} = \sum_{x=1}^{h} C(j_{i,x}) \qquad (1)$$

There are two types of V section: *execution on GSSV* and *execution on message passing*, of which the WCET is heavily dependent on data stored in the GSSV and the number of processed messages in the input buffer associated to task τ_i respectively.

$Val(V_{i,j}, t, \Gamma)$ is the value of the GSSV $V_{i,j}$ at model time t in the context of Γ, determining the control branch in the model (where j is in the range of $[1, g]$ and g is the number of GSSVs associated to task τ_i). The WCET estimate on the task with respect to $V_{i,j}$ is:

$$C_i^p(V_{i,j}) = Sel(Val(V_{i,j}, t, \Gamma), C_{V_{i,j}}) \qquad (2)$$

where $C_{V_{i,j}} = C_{V_{i,j},1}, ..., C_{V_{i,j},k}$, $C_{V_{i,j},k}$ is a specified execution-time in the *kth* branch of control structure in the system, such as `if-else`, `switch-case`, and Sel is a function returning the argument specified by the first argument, expressed as $Sel(x, y_0, ..., y_{n-1}) \mapsto y_x$. Since there are g GSSVs associated with task τ_i, the corresponding WCET of g V sections regarding GSSVs is expressed as follows:

$$\sum_{j=1}^{g} C_i^p(V_{i,j}) = \sum_{j=1}^{g} Sel(Val(V_{i,j}, t, \Gamma), C_{V_{i,j}}) \qquad (3)$$

The message passing between two non-blocking tasks in the system model conforms to asynchronous communication, i.e., the sending and receiving tasks place no constraints on each other in terms of completion, in the communication process. For each task τ_i either as a sending or receiving task, there are b input buffers associated, and the execution time of message passing primitives invoked (i.e., `sendMessage` and `recvMessage`) is noted as $C^{msg\_primitive}$, as a single numeric value WCET representation obtained by using static WCET analysis. The execution time on handling the messages (which may include both message passing primitives and l jobs) in the buffer $U_{i,j}$ in task τ_i, can be expressed as follows:

$$C_i^p(U_{i,j}) = m_{xi,j} \times (C^{msg\_primitive} + \sum_{x=0}^{l} C(j_{i,x})) \quad (4)$$

where $m_{xi,j}$ is either the number of messages sent by the sending task τ_i to buffer $U_{i,j}$, or the number of messages received from buffer $U_{i,j}$ by the receiving task τ_i. Further, the value of $m_{xi,j}$ may not be bounded by the maximum length of the buffer $U_{i,j}$ as the preemption caused by higher priority tasks may preempt the execution in terms of refilling more messages to the buffer $U_{i,j}$ at runtime.

In summary, by combining the two parts of task execution time, i.e., V and NV sections in task τ_i, the WCET expression of task τ_i is a function shown as follows:

$$C_i^p = \sum_{j=1}^{b} C_i^p(U_{i,j}) + \sum_{j=1}^{g} C_i^p(V_{i,j}) + \sum_{j=1}^{c} C_{i,nvj} \quad (5)$$

where c is the number of NV sections in task τ_i. Since there are no dependencies between $U_{i,j}$ and $V_{i,j}$ (i.e., parameters in the WCET expression are independent of each other), the exact WCET of task τ_i in the system model can be obtained by maximizing Equation 5.

2.2 Model Extraction

In general, a major issue when using model-based timing analysis of existing systems is how to obtain the necessary analysis model, which should be a subset of the original program focusing on behavior of significance for task scheduling, communication and allocation of logical resources. For many systems, manual modeling would be far too time-consuming and error-prone. Two methods for automated model extraction are proposed in [17]. A tool for automated model extraction is in development, named MXTC - Model eXtraction Tool for C. The MXTC tool targets large implementations in C, consisting of millions of lines of code, and is based on program slicing [18]. Though the MXTC tool does not support composing the WCET expression of tasks in the real system automatically after the model extraction process, such functionality could be interesting to be developed in a future version.

The models, as the outcome of the tool MXTC, are described by the modeling language used by RTSSim, which describes both architecture and behavior of task-oriented systems developed in C. An RTSSim simulation model consists of a set of tasks, sharing a single processor. Each task in RTSSim is a C program, which executes in a "sandbox" environment with similar services and runtime mechanisms as a normal real-time operating system, e.g., task scheduling, inter-process communication (message queues) and synchronization (semaphores). The default scheduling policy of RTSSim is FPPS and each task has scheduling attributes such as priority, periodicity, offset and jitter. A more thorough description of RTSSim can be found in [19].

3 Formal Analysis in TIMES

In this section, we firstly give the problem definition concerning timing analysis of the referred system model containing data-driven tasks. Then we briefly introduce the prior work on semantic-preserving model transformation, and finally, we present our formal timing analysis using TIMES and upper-part binary search in details.

3.1 Problem Formulation

The problem can be defined as follows. We are given a model S containing n data-driven tasks and m adhering independent parameters which would impact the execution time and response time of tasks radically. Let $ET(S)$ and $RT(S)$ denote the highest execution time and response time measured for the task under analysis in the model S. Our goal is then to find the values of all m parameters which produce $ET(S)$ and $RT(S)$.

3.2 Semantic-preserving Model Transformation

In order to exhaustively search the possible value of both parameters (encoded in the WCET expression that may generate the exact WCET of tasks) and the WCRT of tasks in the entire system state space, the RTSSim models are transformed into a network of task automata [20] in TIMES [21] at the meta-model level via a concrete process *semantic anchoring*. This ensures model validity after the transformation, i.e., the semantics of the source RTSSim model is preserved in its counterpart TIMES model, in terms of describing the same system behavior and valued data. The reason why TIMES is chosen in this work, rather than other widely used model checkers such as UPPAAL [22], is that the detailed execution order of jobs in different priority tasks in the RTSSim model can easily and simply be controlled by giving the *task* attributes and arrival pattern to the scheduler automaton encoded in TIMES (the jobs in the RTSSim tasks are modeled as *tasks* in TIMES). For the sake of space, the interested readers are referred to [14] for more details concerning the transformation rules used in the context which are applied at meta-model level of both the source and target model. Further, a more thorough description of task automata such as the syntax, the semantics, and TIMES tool tutorial can be found in [20] and [21].

3.3 Timing Analysis by using TIMES

Briefly stated, the exact value of the parameters used in the tasks' WCET calculation and the WCRT of tasks are obtained by checking if the candidate solution concerning the value of the counterpart of the parameters in the RTSSim model is the maximum in the entire system search space in the TIMES model, after analyzing the corresponding `reachability` properties in TIMES. Moreover, each candidate solution is derived by using Algorithm 1 and Algorithm 2 together with an upper-part binary search algorithm. In the following sections, we will go through the procedure by referring to some detailed examples.

3.3.1 Obtaining the WCET of Tasks

Algorithm 1 shows the procedure of maximizing the task WCET expression, i.e., Equation 5 introduced in Section 2.1, by using upper-part binary search and reachability property check in TIMES. As introduced in Section 2, each τ_i is associated with b input buffers and g GSSVs. The number of messages $m_{xi,j}$ (see Equation 4), which is either the number of messages sent by task τ_i when τ_i is a sending task, or the number of messages received by task τ_i when τ_i is a receiving task, can be obtained by using an upper-part binary search algorithm in the search space (see lines 4 to 25 in Algorithm 1) consisting of a lower bound 0 (i.e., when the queue is empty) and an upper bound $queuesize$ (i.e., the maximum queue length which is given either by the assumption under the worst-case scenario when preemption caused by higher priority tasks occurs, or by the experienced engineers who have the knowledge of such system behavior). For the WCET estimate with respect to a GSSV $V_{i,j}$ (where $1 \leq j \leq g$), it can be obtained by checking the reachability property concerning each condition specified by $Sel(Val(V_{i,j}, t, \Gamma), C_{V_{i,j}})$ (see Equation 3 in Section 2.1). The corresponding part in Algorithm 1 covers lines 26-28. Finally, the WCET calculation can be done by using Equation 5, once the value of parameters M_i, V_i and NV_i[5] are given.

For a better understanding, we refer to an example where there are two parameters, i.e., parameter `i` and `j` (see lines 36-53 in Figure 5). The counterpart of the parameter `i` in the CTRL task is a bounded integer `recvIOQ_i_ctrl` in the TIMES model preserving the same semantics, i.e., it counts for the number of times that the primitive `recvMessage` is executed. Step 4 in Table 1 shows that the maximum of the parameter `i` is 10 (in bold in Table 1), obtained by checking the *reachability* properties with regard to `recvIOQ_i_ctrl`, such as $E <> recvIOQ\_i\_ctrl == 11$ (with verification result `Not satisfied`) and $E <> recvIOQ\_i\_ctrl == 10$ (with verification result `Satisfied`) in TIMES using upper-part binary search. The maximum of the parameter `j` is 0 due to the fact that the corresponding reachability check in the TIMES model regarding the condition `gstate1_ctrl > 6` can not be true. The tightness of the results obtained by our proposed method is 16.7% and 100% less pessimistic when compared to the

[5]The value of NV_i could either be automatically obtained by using the relevant tool developed, or manually derived from code inspection.

Algorithm 1 $wcetcal_{\tau_i}(M_i)$

```
1: $M_i \leftarrow m_{i,1}, ..., m_{i,b}$
2: $V_i \leftarrow v_{i,1}, ..., v_{i,g}$
3: $NV_i \leftarrow nv_{i,1}, ..., nv_{i,c}$
4: for all $m_{i,j}$ such that $1 \leq j \leq b$ do
5:     $lwb \leftarrow 0, upb \leftarrow queuesize$
6:     $success \leftarrow false, ret_{veri} \leftarrow false$
7:     while $success = false$ do
8:         $p \leftarrow \left\lfloor \frac{lwb + upb}{2} \right\rfloor$
9:         $ret_{veri} \leftarrow reachabilitycheck(p)$
10:        if $ret_{veri} = false$ then
11:            if $p + 1 = upb$ then
12:                $m_{i,j} \leftarrow p - 1$
13:                $success \leftarrow true$
14:            else
15:                $upb \leftarrow p$
16:            end if
17:        else
18:            if $lwb = upb - 1$ then
19:                $lwb \leftarrow upb$
20:            else
21:                $lwb \leftarrow p$
22:            end if
23:        end if
24:    end while
25: end for
26: for all $v_{i,j}$ such that $1 \leq j \leq g$ do
27:    $tmp \leftarrow reachabilitycheck(Sel(Val(v_{i,j}, t, \Gamma), C_{v_{i,j}}))$
28:    $C_i^p(v_{i,j}) \leftarrow tmp$
29: end for
30: $wcetcal_{\tau_i} \leftarrow C_i^p$
31: return $wcetcal_{\tau_i}$
```

results obtained through our basic assumption (introduced in Section 4.4). Further, our proposed analysis methods can not only contribute to deriving less pessimistic WCET and WCRT estimates of tasks, but it can also save memory space used in the system by providing shortened but assured queue size of messages covering worst-case scenarios.

Next, by using Equation 8 which is populated by relevant timing information about modeling primitives (refer to Section 4.3), the WCET of the CTRL task is 22 (i.e., $10 \times 2 + 2 + 0 \times 10$). As the result is obtained by exhaustively searching the entire system state space, we believe that this is the exact WCET of the CTRL task in the model. In addition the result is in line with the result derived from simulation-based methods, and thus we have confidence in the result being a safe upper bound.

3.3.2 Obtaining the WCRT of Tasks

Before obtaining the WCRT of the task being considered, the sum of all WCETs of higher priority tasks and the task itself has to be calculated by iteratively repeating

Table 1. The procedure of using upper-part binary search concerning the value of the parameter i.

Step	*lwb*	*upb*	*Checkpoint*	*Results*
1	0	12	6	Satisfied
2	6	12	9	Satisfied
3	9	12	**10**	Satisfied
4	10	12	11	Not satisfied

upper-part binary search as introduced in Section 3.3.1. Next, the concept of deriving the exact WCRT by checking the reachability properties with respect to one clock `task_clk_rt` and a bounded integer `task_finish` (in the range of 0 and 1) in the TIMES model is explained using Figure 3.

- When the task is released, `task_clk_rt` is reset to 0, and `task_finish` is assigned to 0.
- Once the task finishes its execution, `task_finish` is assigned to 1.

The corresponding reachability property involved in upper-part binary search is as follows:

- $E <> task\_clk\_rt == Checkpoint \ and \ task\_finish == 0$

Algorithm 2 $wcrt_{\tau_i}()$

```
1: $lwb \leftarrow 0, upb \leftarrow upb_{init}$
2: $success \leftarrow false, ret_{veri} \leftarrow false$
3: while $success = false$ do
4:   $p \leftarrow \left\lfloor \frac{lwb + upb}{2} \right\rfloor$
5:   $ret_{veri} \leftarrow reachabilitycheck(p)$
6:   if $ret_{veri} = false$ then
7:     if $p + 1 = upb$ then
8:       $wcrtcal_{\tau_i} \leftarrow p - 1$
9:       $success \leftarrow true$
10:    else
11:      $upb \leftarrow p$
12:    end if
13:  else
14:    if $lwb = upb - 1$ then
15:      $lwb \leftarrow upb$
16:    else
17:      $lwb \leftarrow p$
18:    end if
19:  end if
20: end while
21: return $wcrtcal_{\tau_i}$
```

Further, for task τ_i, i.e., the task under analysis, the initial lower bound used in upper-part binary search is the WCET of the task, i.e., C_i, and the upper bound upb_{init} is obtained by Equation 6.

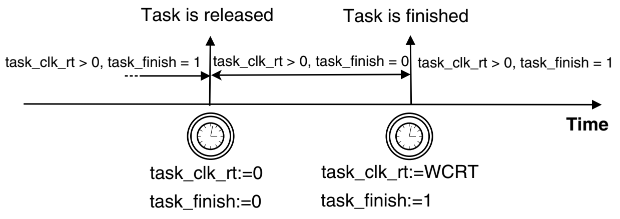
Task is released
Task is finished
task_clk_rt > 0, task_finish = 1
task_clk_rt > 0, task_finish = 0
task_clk_rt > 0, task_finish = 1
Time
task_clk_rt:=0
task_finish:=0
task_clk_rt:=WCRT
task_finish:=1

Figure 3. Using one clock and one bounded integer to calculate WCRT of task on focus in the TIMES model.

$$upb_{init} = \sum_{\forall j \in hp(i)} C_j + C_i \tag{6}$$

where $hp(i)$ is the set of all tasks with priority higher than that of task τ_i.

upb_{init} equates to the sum of the WCET of all higher priority tasks and task τ_i itself. This is a safe upper bound for which the true WCRT of task τ_i cannot be bigger. Next, Algorithm 2 will search the possible maximum of the WCRT of tasks in the model in the range of C_i and upb_{init}. Table 2 shows a concrete example of using upper-part binary search in Algorithm 2 to derive the WCRT of the task under analysis in the system model used in the evaluation as introduced in Section 4. The lower bound of the search space concerning the WCRT estimate on the CTRL task is its WCET estimate, i.e., 22 model-time units. 34 (model-time units) which is in bold in Table 2 is the final result obtained through the search procedure.

Table 2. The procedure of using upper-part binary search to obtain the WCRT of the task under analysis in the example used for the evaluation. The unit is a model-time unit.

Step	*lwb*	*upb*	*Checkpoint*	*Results*
1	22	34	28	Satisfied
2	28	34	31	Satisfied
3	31	34	32	Satisfied
4	32	34	33	Satisfied
5	34	34	**34**	Satisfied

4 Experimental evaluation

In this section, we firstly introduce our testbed and the toolchain in details, and then we describe the model used for the evaluation, which is designed to include all the behavioral mechanisms from the system model presented in Section 2. Next, the WCET expression of the tasks in the model is given, and finally, the results comparison between basic RTA using WCET estimates on tasks with a basic assumption and our proposed analysis method is presented.

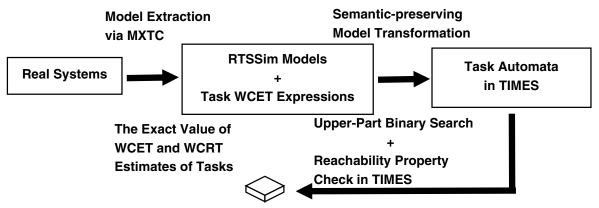
Model Extraction via MXTC
Semantic-preserving Model Transformation
Real Systems
RTSSim Models + Task WCET Expressions
Task Automata in TIMES
The Exact Value of WCET and WCRT Estimates of Tasks
Upper-Part Binary Search + Reachability Property Check in TIMES

Figure 4. The toolchain in this work.

4.1 Testbed and Toolchain

Our testbed is running Microsoft Windows XP Professional, version 2002 with Service Pack 3. The computer is equipped with the Intel Core Duo CPU P9400 processor, 3.45 GB RAM and a 6 MB L2 Cache. The processor has 2 cores and 1 frequency level: 2.40 GHz.

The toolchain proposed in this work is showed in Figure 4. A key point here is that the current work concerning semantic-preserving model transformation is conducted and validated manually to ensure that it is not error-prone. More important, the entire toolchain will be automated when the relevant tools, concerning performing automatic model transformation and the process of obtaining the WCET and WCRT of tasks by using Algorithm 1, Algorithm 2 and TIMES introduced in Section 3.3, are developed, as part of our future work.

4.2 Model Description

The evaluation model contains a First-In-First-Out (FIFO) buffer IOQ with a queue size 12[6] and three periodic, non-blocking tasks executed on a single processor under FPPS, i.e., ENV_IO, IO and CTRL task with the parameters shown in Table 3. The ENV_IO task is an environmental task which generates 2 external events that are stored in the global variable `nofEvents`, as shown in line 8 in Figure 5. The complex temporal dependencies between the IO and CTRL tasks are dependent on the input-dependent data placed in the IOQ queue and the GSSV `gstate1_ctrl` which vary the execution time of tasks radically:

- The IO task sends an uncertain number (from 0 to 6) of messages to the IOQ queue depending on the value of the global variable `nofEvents`: if the value is bigger than 6, then there will be at most 6 messages sent to the IOQ queue; Otherwise, the number of messages sent by the IO task is the same as the value of `nofEvents`. Once a message is sent to the IOQ queue, the value of `nofEvents` is accordingly decreased by 1, in order to note that one external event is successfully stored in the IOQ queue. Moreover, the number of messages sent by the IO task is counted by the local variable `k` (see lines 16-30 in Figure 5). More important, in line 23, the GSSV `gstate1_ctrl` is assigned by the global variable `nofEvents`, impacting the WCET of the CTRL task;
- When the CTRL task starts running, it will firstly consume all the messages stored in the IOQ queue. However, in the event that its priority is lower than the priority of the IO task, the CTRL task may be preempted in the loop `do while` by the IO task which refills an uncertain number of messages as described previously. This will increase the number of messages consumed by the CTRL task, which is counted by the local variable `i` (see lines 39-46 in Figure 5). After consuming all the messages in the IOQ queue, the CTRL task may continue its execution with 10 model-time units more and increase the value of the local variable `j` depending on whether the value of the GSSV `gstate1_ctrl` is bigger than 6 or not (see lines 48-52 in Figure 5).
- The WCET estimate on jobs and modeling primitives in the adhering tasks in the evaluation model is designed as a single numeric value, which is supposed to be obtained by using static WCET analysis in practice.

[6]Due to tasks periodicity and priority (refer to Table 3), in the worst-case scenario, the CTRL task would be preempted by the IO task twice in one of its invocation. Further, per preemption, the IO task will refill the maximum number of messages to the buffer IOQ, i.e., 6. The corresponding safe upper bound of the IOQ size is i.e., $\lceil \frac{T_{ctrl}}{T_{io}} \rceil \times 6 = \lceil \frac{1000}{500} \rceil \times 6 = 12$.

Table 3. Tasks and task parameters for the evaluation model. The lower numbered priority is more significant, i.e., 0 stands for the highest priority.

Task	*Priority*	*Period*	*V section*	*Parameter*	*Job*
ENV_IO	0	200	0	No	0
IO	1	500	1	`k`	1
CTRL	2	1000	2	`i, j`	2

4.3 The Task WCET Expression

For the IO task, there is only one V section inherent in the asynchronous message-passing with the CTRL task via the IOQ queue. For the CTRL task, there are two V sections inherent in the asynchronous message-passing with the IO task via the IOQ queue and the GSSV `gstate1_ctrl` shared with the IO task. Further, all the V sections in both the IO and CTRL tasks are marked with `comments` in Figure 5 for a better illustration. The WCET expression of the two tasks are as follows:

$$C^p_{IO} = C^p_{io,v} = k \times C^{sendMsg} \tag{7}$$

$$C^p_{CTRL} = i \times C^{recvMsg} + 2 + j \times C(10) \tag{8}$$

where $C^{sendMsg}$ and $C^{recvMsg}$ are 2 model-time units consumed by the primitives `sendMessage(tcb,` `IOQ, 1, 0)` and `recvMessage(tcb, IOQ, 0)` respectively; $C(10)$ represents 10 model-time units consumed by the primitive `execute(tcb, 100, 10)`.

4.4 Basic RTA

In order to apply basic RTA to analyze the evaluation model, the WCET of tasks has to be computed beforehand. Moreover, such WCETs in basic RTA specify the longest time of executing the code of the task on the CPU uninterruptedly. However, since the variables are dependent on other tasks, such as `ioevent` and `gstate1_ctrl` in the CTRL task, they are unbounded. Unless program annotation is given, the application of static WCET analysis is not feasible. Manual annotation may be error-prone and far less efficient. In addition, with the purpose of performing safe analysis, covering the system model worst-case behavior when runtime information is missing, the annotation to, e.g., `ioevent` and `gstate1_ctrl` is given by the basic assumption, that is, the maximum queue length (i.e., 12) and the maximum of the variable (i.e., 1). Further, our basic assumption is also at system-level, since the way of designing queue size takes the possibility of task preemption into consideration. Once the value of parameters is obtained, the WCET of tasks can be calculated and plugged into the response-time computation formula in basic RTA, in a position to obtain the WCRT of tasks on focus. Note that the cost of executing 12 true evaluations of the control structure `do while` and 1 false evaluation is not considered under the assumption that such costs are far smaller compared with the WCET of jobs and execution of primitives such as message passing.

4.5 Results Comparison

In Table 4, the value of parameter `i` obtained by TIMES is 10, and can be seen to be 16.7% (i.e., $(12 - 10)/12 \times 100\%$) less pessimistic than 12 given by our basic assumption relying on the use of maximum queue length. Moreover, the value of parameter `j` obtained by TIMES, i.e., 0, is 100% (i.e., $(1 - 0)/1 \times 100\%$) less pessimistic when compared with the value assumed by our basic assumption, i.e., the maximum of the variable. This shows that the worst-case behavior can never include executing the statement `execute(tcb, 100, 10)` due to the condition that `gstate1_ctrl > 6` can not be true. Next, concerning the WCET of the CTRL task, the result using the value of parameters obtained by using TIMES is 38.9% (i.e., $(36 - 22)/36 \times 100\%$) less pessimistic when compared with the WCET estimate derived from a basic assumption, as shown in Table 5. Regarding the WCRT of the CTRL task, clearly, the result given by TIMES reduces the pessimism by 29.2% (i.e., $(48 - 34)/48 \times 100\%$) when compared to basic RTA.

5 Scalability of the Method

A common issue when using a model checker is the risk of state space explosion, though the corresponding results of the analysis are ensured to cover the worst-case scenario. We would like to separate the scalability of the method presented in this paper from the efficiency of the model checker handling the problem, since the current analysis tool TIMES may not be the optimal solution.

Table 4. The upper bound of parameters in the evaluation model determined by different analyses.

Parameter	*Basic Assumption*	*TIMES*
i	12	10
j	1	0
k	6	6

Table 5. The results obtained by TIMES and basic RTA using WCET estimates on tasks with basic assumption. The unit is one model time unit.

WCET/WCRT	*Basic RTA using WCET Estimates on Tasks with Basic Assumption*	*TIMES*
WCET(IO)	12	12
WCET(CTRL)	36	22
WCRT(CTRL)	48	34

Further, TIMES provides the functionality of converting the TIMES model which consists of a network of task automata, to the timed automata used in UPPAAL tool, which could be used for instance to measure the growth rate of the system state space and memory assumption when the number of jobs, GSSVs and message queues are changed. However, such a process failed due to many errors caused by the obsolete modeling pattern of timed automata encoded by TIMES in the conversion process, especially regarding the scheduler automaton encoded in TIMES. Therefore, in this work, it is impossible to use, e.g., *verifyta* in UPPAAL to achieve the goal. Nonetheless, we can provide information about the growth rate of the system space concerning the GSSVs in terms of Big O notation [23], i.e., $O(m^n)$, where m is the maximum interval of n GSSVs, and n is the number of GSSVs.

By checking each reachability property using our experimental computer as introduced in Section 4.1, the time in which an answer is returned by a verification engine in TIMES is less than 1 second. Moreover, we succeeded in analyzing the system model containing 4 periodic non-blocking tasks with 2 message queues and 2 GSSVs successfully. However a small example such as this is not sufficient evidence of scalability and therefore more experimentation will be performed as part of future work. Preferably, the method proposed in this work is better off fitting to the case where the size of the system model is relatively small, e.g., there is a small number of tasks, message queues, jobs and GSSVs. However, on the positive side, our approach would return accurate information about the WCET and WCRT of the tasks under analysis in the system model containing intricate execution dependencies between tasks.

6 Related Work

This section introduces related work which has not been discussed previously. The prior work presented in the field of WCET analysis [24] generally does not consider handling asynchronous message-passing and globally shared state variables between tasks. Instead, such work assume that tasks in the analysis are isolated and independent of each other. For RTA, the current work on execution precedence constraints of tasks is presented in [25]. The latest work on temporal dependencies between tasks, in terms of offset, is presented in [26]. The work about using timed automata-based analysis to perform worst-case response-time analysis of an in-car radio navigation system and conformance tests for real-time systems with timed automata specification are presented in [27] and [28] respectively.

Another interesting work is to use model-based schedulability analysis of safety critical hard real-time Java programs by using the model checker UPPAAL as presented in [29]. However, the tasks (or threads as introduced in [29]) do not have the execution dependencies that we are aiming at, such as asynchronous message passing impacting the number of the adhering loop iterations and globally shared state variables used for dispatching the control branch.

7 Conclusions and future work

In this paper we have presented a novel approach to timing analysis of software systems with task execution dependencies. We proposed a new system model to allow for a more complex task WCET expression which could be maximized by using bounds of parameters obtained by inter-task analysis at system-level. Given this model, we show how to derive the exact WCRT of tasks using TIMES. By applying the approach presented in the paper to a model derived from a real robotic control system shows the benefit, in terms of reduced pessimism, over basic RTA and WCET estimates using our basic assumption, e.g., maximum queue size and maxima of the variables. As part of our future work, the proposed method will be extended to handle cases in which the priority and period of the task under analysis can be changed by other tasks during runtime. More important, the possibility of using other model checkers will be explored with the focus on better dealing with the state space explosion issue, as well as modeling context switch and/or scheduler cost.

Acknowledgment

This work was supported by the Swedish Foundation for Strategic Research via the strategic research centre PROGRESS. We are grateful to Stefan Bygde, Diane Pecorari and Linh Thi Xuan Phan for the constructive comments and improvement suggestions.

References

[1] I. Bate, "Scheduling and timing analysis for safety-critical systems," Ph.D. dissertation, Department of Computer Science, University of York, November 1998.

[2] "Website of ABB Group," www.abb.com.

[3] N. Audsley, A. Burns, R. Davis, K. Tindell, and A. Wellings, "Fixed priority pre-emptive scheduling: an historical perspective," *Real-Time Systems*, vol. 8, no. 2/3, pp. 129–154, 1995.

[4] "Rapita systems, www.rapitasystems.com, 2008."

[5] D. Decotigny and I. Puaut, "ARTISST: an extensible and modular simulation tool for real-time systems," in *Proc. of the 5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '02)*, 2002, pp. 365–372.

[6] J. Kraft, Y. Lu, C. Norström, and A. Wall, "A metaheuristic approach for best effort timing analysis targeting complex legacy real-time systems," in *RTAS 08*, April 2008, pp. 258–269.

[7] M. Bohlin, Y. Lu, J. Kraft, P. Kreuger, and T. Nolte, "Simulation-based timing analysis of complex real-time systems," in *RTCSA 09*, August 2009, pp. 321–328.

[8] G. A. Kaczynski, L. L. Bello, and T. Nolte, "Deriving exact stochastic response times of periodic tasks in hybrid priority-driven soft real-time systems," in *Proceedings of 12th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'07)*. IEEE Industrial Electronics Society, September 2007, pp. 101–110.

[9] S. Manolache, P. Eles, and Z. Peng, "Schedulability analysis of applications with stochastic task execution times," *ACM Trans. Embed. Comput. Syst.*, vol. 3, no. 4, pp. 706–735, 2004.

[10] Y. Lu, T. Nolte, J. Kraft, and C. Norström, "Statistical-based response-time analysis of systems with execution dependencies between tasks," in *ICECCS 2010*, March 2010.

[11] J. S. J. Beirlant, Y. Goegebeur and J. Teugels, *Statistics of Extremes: Theory and Applications*. Wiley Press, 2004.

[12] Y. Lu, T. Nolte, I. Bate, and C. Norström, "Timing analyzing for systems with execution dependencies between tasks," in *Track on Real-Time Systems, The 25th ACM Symposium on Applied Computing (SAC2010)*. ACM, March 2010.

[13] R. R. T. Cormen, C. Leiserson and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, September 2001.

[14] Y. Lu, A. Cicchetti, S. Bygde, J. Kraft, T. Nolte, and C. Norström, "Transformational specification of complex legacy real-time systems via semantic anchoring," in *2nd IEEE International Workshop on Component-Based Design of Resource-Constrained Systems (CORCS 2009) @ COMPSAC*. IEEE Computer Society Press, July 2009.

[15] M. Joseph and P. Pandya, “Finding response times in a real-time system,” *The Computer Journal (British Computer Society)*, vol. 29, no. 5, pp. 390–395, October 1986.

[16] S. Bygde, A. Ermedahl, and B. Lisper, “An efficient algorithm for parametric wcet calculation,” in *The 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2009*, August 2009.

[17] J. Kraft, J. Huselius, A. Wall, and C. Norström, “Extracting simulation models from complex embedded real-time systems,” in *Real-Time in Sweden 2007*, August 2007.

[18] M. Weiser, “Program Slicing,” in *Proc. of the Int. Conf. ICSE’81*. IEEE Press, 1981, pp. 439–449.

[19] J. Kraft, “RTSSim - A Simulation Framework for Complex Embedded Systems,” Mälardalen University, Technical Report, March 2009.

[20] E. Fersman, P. Krcal, P. Pettersson, and W. Yi, “Task automata: Schedulability, decidability and undecidability,” *Inf. Comput.*, vol. 205, no. 8, pp. 1149–1172, 2007.

[21] 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 *Procs. of FORMATS 03*, ser. LNCS, no. 2791. Springer-Verlag, 2003, pp. 60–72.

[22] G. Behrmann, A. David, and K. G. Larsen, “A tutorial on UPPAAL,” in *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, ser. LNCS, M. Bernardo and F. Corradini, Eds., no. 3185. Springer–Verlag, September 2004, pp. 200–236.

[23] “Big-O Notation and Algorithm Analysis,” /www.eecs.harvard.edu/ ellard/Q-97/HTML/root/node8.html, Jan. 2010.

[24] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, “The worst-case execution-time problem—overview of methods and survey of tools,” *Trans. on Embedded Computing Sys.*, vol. 7, no. 3, pp. 1–53, 2008.

[25] I. Bate and A. Burns, “An integrated approach to scheduling in safety-critical embedded control systems,” *Real-Time Syst.*, vol. 25, no. 1, pp. 5–37, 2003.

[26] J. Mäki-Turja and M. Nolin, “Efficient implementation of tight response-times for tasks with offsets,” *Real-Time Systems Journal*, vol. 40, no. 1, pp. 77–116, October 2008.

[27] M. Hendriks and M. Verhoef, “Timed automata based analysis of embedded system architectures,” in *The 20th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2006*, April 2006.

[28] R. Cardell-Oliver, “Conformance tests for real-time systems with timed automata specifications,” *Formal Aspects of Computing*, pp. 350–371, December 2000.

[29] T. Bøgholm, H. Kragh-Hansen, P. Olsen, B. Thomsen, and K. G. Larsen, “Model-based schedulability analysis of safety critical hard real-time java programs,” in *JTRES ’08: Proceedings of the 6th international workshop on Java technologies for real-time and embedded systems*. New York, NY, USA: ACM, 2008, pp. 106–114.

```
#define IOQSIZE 12

int nofEvents = 0;
int gstate1_ctrl = 0;

void RTSSim_ENV_IO(TCB* tcb)
{
    nofEvents += 2;
}

void RTSSim_IO(TCB* tcb)
{
    int eventsToProcess = 0;
    int k = 0;  // parameter k

    if (nofEvents > 6)
    {
        eventsToProcess = 6;
    }else{
        eventsToProcess = nofEvents;
    }

    gstate1_ctrl = nofEvents;

    while(eventsToProcess-- > 0) // V section starts
    {
        nofEvents--;
        sendMessage(tcb, IOQ, 1, 0);
        k++;
    }                             // V section ends
}

void RTSSim_CTRL(TCB* tcb)
{
    int ioevent = 0;
    int i = 0; // parameter i
    int j = 0; // parameter j

    do{                          // V section starts
        ioevent = recvMessage(tcb, IOQ, 0);

        if (ioevent > -1)
        {
            i++;
        }
    }while (ioevent > -1);      // V section ends

    if(gstate1_ctrl > 6)        // V section starts
    {
        execute(tcb, 100, 10);
        j++;
    }                            // V section ends.
}
```

Figure 5. The RTSSim model used in the evaluation.