IEICE TRANS. INF. & SYST., VOL.E99–D, NO.8 AUGUST 2016

LETTER

Hierarchical System Schedulability Analysis Framework Using UPPAAL

So Jin AHN[†a)], ***Student Member***, **Dae Yon HWANG**[†], ***Nonmember***, **Miyoung KANG**[†], ***Member***, ***and*** **Jin-Young CHOI**[†b)], ***Nonmember***

SUMMARY Analyzing the schedulability of hierarchical real-time systems is difficult because of the systems' complex behavior. It gets more complicated when shared resources or dependencies among tasks are included. This paper introduces a framework based on UPPAAL that can analyze the schedulability of hierarchical real-time systems.

key words: *hierarchical system, schedulability analysis, real-time systems, formal methods, UPPAAL*

1. Introduction

Virtualization technology has been attracting increasing attention for its advantages like isolation and easy maintenance. However, it is difficult to analyze the schedulability of a system using virtualization technology because the schedulers in the system are hierarchical. One of the more important requirements of real-time systems is that all the real-time tasks in the system complete their execution within the allotted period. When a task related to safety misses a deadline, it could lead to a system failure causing considerable casualties and property damage. Therefore, it is essential to verify that all the real-time tasks in a system are always schedulable.

[1], [2], and [3] formally specify and verify two-level hierarchical systems under the rate-monotonic (RM) or earliest-deadline-first (EDF) scheduling policy. These studies analyze in detail the system behavior because all schedulers of the system are analyzed together. However, thus far, no method that can handle more than two hierarchical levels has been developed.

[4]–[6], and [7] formally analyze the schedulability of hierarchical real-time systems with the compositional analysis approach. The compositional analysis approach is used for partitioning a huge hierarchical system into subsystems, and each partition is separately verified as whether it is always schedulable. They prove that if all subsystems are always schedulable, then the entire system is always schedulable. [6] and [7] specify shared resources such as semaphore or message passing and verify a system that includes them. The compositional analysis approach can prevent state explosion and analyze more than two-level hierarchical systems efficiently. Although this approach has its advantages, it is difficult to check the behaviors such as preemptions and priority inversion problem in detail, and to specify the shared resources or dependencies among tasks intuitively.

In this paper, a formal specification and verification framework based on UPPAAL [8] to analyze the schedulability of a hierarchical real-time system is proposed. The advantages of this framework are as follows:

1) A hierarchical real-time system can be formally specified with time requirements.
2) Shared resources and dependencies among tasks can be specified intuitively.
3) The framework helps to update worst case execution time (WCET) with the overhead time related to preemption by counting the preemptions of each task.
4) A user-defined scheduling policy can be included in the framework and verified.

This work advances hierarchical real-time system analysis because it can present detailed system behaviors, which reduces the number of false-positive results and yields accurate WCET.

2. Hierarchical Schedulability Analysis Framework

2.1 Hierarchical Real-Time System

A real-time system and a hierarchical real-time system are defined as follows [9]:

Definition 1. Real-time system (RS)
A real-time system RS can be defined as follows:
RS = (W, R, S), where W represents a set of tasks, such as real-time preemptive tasks; R, a resource model; and S, a scheduling policy.

Definition 2. Hierarchical real-time system (HS)
A hierarchical real-time system HS can be recursively defined as follows:
1) a real time system RS, or
2) HS = (W_H, R, S), where W_H denotes a set of real-time preemptive tasks and hierarchical systems; R, a resource model; and S, a scheduling policy.

Manuscript received January 7, 2016.
Manuscript revised March 28, 2016.
Manuscript publicized May 6, 2016.
[†]The authors are with Korea University, Seoul, Korea.
a) E-mail: sjspirit@formal.korea.ac.kr
b) E-mail: choi@formal.korea.ac.kr (Corresponding author)
DOI: 10.1587/transinf.2016EDL8003

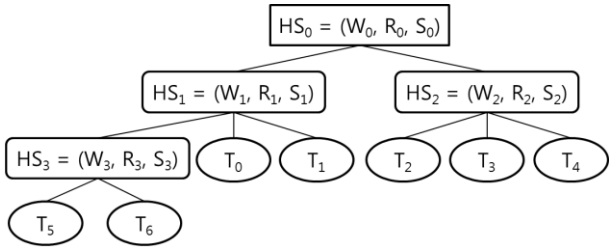

Fig. 1 Example of hierarchical real-time system.

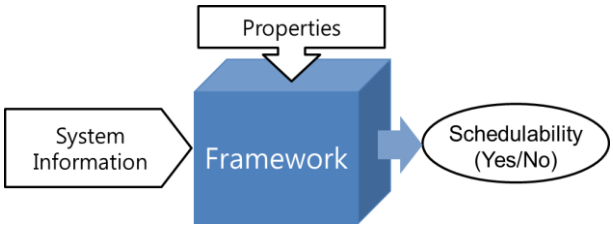

Fig. 2 Overview of proposed framework.

Figure 1 shows an example of a hierarchical real-time system. T_n ($0 \leq n \leq 6$) in Fig. 1 represent real-time preemptive tasks. Each HS has a scheduler and a workload, that is, a set of HSs and tasks that the scheduler schedules. For example, in the case of $HS_1 = (W_1, R_1, S_1)$ in Fig. 1, the workload is $W_1 = \{HS_3, T_0, T_1\}$. HS_1 does not recognize that HS_3 is a system; the scheduler of HS_1 schedules HS_3 as it schedules T_0 and T_1.

A resource model is an abstraction of resources such as physical CPUs or virtual CPUs. In this study, a resource model of the root HS, for example, R_0 of HS_0 in Fig. 1, is considered an abstraction of a physical resource. The other resource models, for example, R_1 of HS_1 and R_2 of HS_2 represent an abstraction of a virtual resource that is a virtualized R_0. R_3 of HS_3 is a model of virtualized R_1.

A scheduling policy is an algorithm that describes the order in which HS assigns an abstracted resource to the elements of the workload. RM and EDF are the most popular scheduling policies for real-time systems; therefore, the framework includes the RM and EDF policies by default.

The proposed framework will be explained with the following notations: T represents a real-time preemptive task, the root HS is denoted as P, and HSs that are not root HS are indicated as V.

2.2 Framework for Schedulability Analysis of Hierarchical Real-Time System

The proposed framework specifies and verifies HSs that have shared resources and dependencies among tasks without partitions. Figure 2 shows an overview of the framework. The system information is about the schedulers and tasks and includes workloads, scheduling policies, and time requirements. The framework inputs are as follows:

- P = (W, R_P, S), where $R_P = (\infty, \infty)$
- V = (W, R_V, S), where $R_V = (Pe, B)$
- T = (Pe, E, D, O, SR, Dep)

Workloads (W) of P and V are the same as the workloads explained in 2.1, and so are the *scheduling policies (S)*. RM, EDF, and preemptive FP are built into the framework, and user-defined scheduling policies can be added to the framework.

To specify resource models of P and V, information of period and time budget is needed. A HS is given a specific amount of time budget that can be used every period by the parent system. *Pe* represents the length of one period, and *B* denotes the budget time given to HS for using a physical resource in this period. P manages physical resource directly which means P has infinite *Pe* and *B* by default. Therefore, R_P, the resource model of P is (∞, ∞) by default, and R_V, the resource model of V need inputs of *Pe* and *B*.

A task has to use the physical resource for the *execution time (E)* every period. The first period of a task starts at the *offset* time *O* after the system is started. *D* denotes the deadline of the task, by which it has to complete its job. If there is any *shared resource (SR)* among tasks, the id of the shared resource needs to be included in the task information. *Dep* denotes the dependency variable. It is optional information that specifies a set of tasks that must completed before the given task can be executed. The proposed framework can specify tasks that are not in the same scheduler as *SR* and *Dep*.

2.3 Resource Request and Allocation Procedure for the Proposed Framework

Physical resource scheduler is a scheduler of HS_0. It sorts the elements of its workload that are requesting resources with its scheduling policy every time an element sends a new request or an element completes its execution. After sorting the elements, the scheduler selects the first element of the workload and allocates the resource.

Virtual resource scheduler is a scheduler of V. It schedules its workload in the same manner as the physical resource scheduler except for some limitations. A virtual resource is not always available; it is available only when it occupies a physical resource. Therefore, the virtual resource scheduler can only schedule and allocate its resource when it is given a physical resource. V has a predefined time period and time budget; therefore, it cannot request more than the assigned budget per period, but this does not mean that V has to spend its complete budget every time; if no elements of V's workload request a resource, then the virtual resource will be idle. Because V has no elements to allocate its resource, a physical resource will be wasted if there are any other Vs or Ts that are waiting for the physical resource. Therefore, in the proposed framework, we assume that the resources in the system will not be idle if there is any V or T requesting for the resource; the V that is idle will yield its use of the physical resource. This implies that any V that cannot consume its entire budget is not related to the scheduling problem.

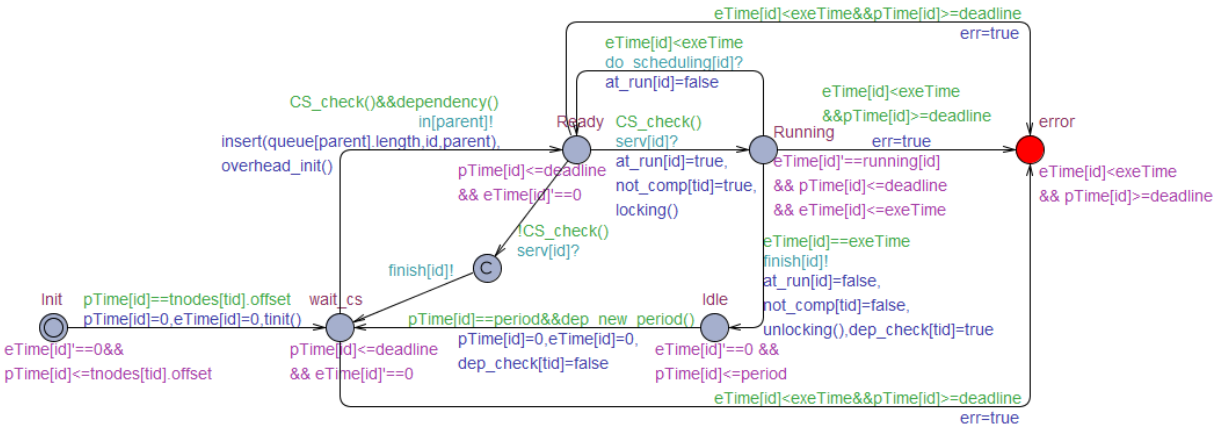

Fig. 3 Task model.

Figure 3 shows a timed automaton modelling a task. Tasks begin their first period after their start offset time O. From the second period, T starts its new period right after a period ends. T checks the availability of the shared resource and the completion of the dependent tasks before requesting a resource at the beginning of every period. If a shared resource is available (or there is no shared resource) and the execution of every dependent task has been completed (or there is no dependent task), then T requests a resource and remains in the Ready state. Before T transits its state from Ready to Running, which implies that a resource is allocated to T, T checks the state of the shared resource and the dependent tasks again. If any conditions to execute T are not satisfied, then T yields the resource.

Every task must get the resource of P for its execution time E per its period time Pe. The schedulability of the system will fail if any of the tasks cannot complete its execution within its period time Pe. In Fig. 3, the Error station implies that the task failed to complete its job in time.

2.4 Preemption Counting

Overheads extend the execution time E of tasks beyond the required time. The context switching (CXS) overhead and the cache migration and preemption delay (CMPD) overhead are introduced when a task starts or is preempted. To calculate these overheads, we need to count how many times the task is preempted. The proposed framework counts each task's preemption occurrence per its period, respectively.

In this study, we classify preemption into two types: intra-scheduler preemption and inter-scheduler preemption.

A running task is intra-scheduler-preempted when an element in the same workload preempts the task, and is inter-scheduler-preempted when the HS scheduling the task is preempted or runs out of budget before the period ends. In other words, inter-scheduler preemption occurs if a task is preempted by another element that belongs to another workload.

After counting the preemptions of each T, we can calculate the new execution times of each T. In this paper, we assume that the execution time of a task is delayed for a certain amount of CXS overhead time and the CMPD overhead time when a task is preempted. The new execution time of the task is then calculated as follows:

$$E_t = E_t' + (\Delta^{\mathrm{CXS}} + \Delta^{\mathrm{CMPD}}) * \mathrm{PRM_t}$$

where t denotes the id of a task; E_t, the new execution time of the task; E_t', the current execution time; Δ^{CXS}, the context switching overhead time; Δ^{CMPD}, the CMPD overhead time; and $\mathrm{PRM_t}$, the sum of the maximum intra-scheduler preemption number and the maximum inter-scheduler preemption number per period of the task.

3. Case Study

In this section, we demonstrate the proposed framework by specifying and verifying a target system. The information of the target system is presented in Table 1.

After inserting the information into the proposed framework, the following verification property written in TCTL [8] is checked formally.

A[] not err

The TCTL sentence means the variable *err* never be true in any execution path of the system. When a task misses its deadline, the variable *err* is set to true. If there is no path that leads to a state where *err* is true then the task can always be completed before its deadline.

The UPPAAL tool automatically verifies the property whether it is "Satisfied" or "Unsatisfied" when the target system information and the property are given to the proposed framework. "Satisfied" implies that the target system is schedulable, and "Unsatisfied" means that there are one or more cases in which the target system misses a deadline.

The verification result of our case study is "Satisfied."

Figure 4 illustrates a case of the target system specified in Table 1 with a shared resource between T_1 and T_3. The verification result is "Unsatisfied"; the counter example shows that the shared resource SR_0 can cause a priority inversion problem leading to a deadline miss for T_4.

Table 2 shows the preemption counts of the target system specified in Table 1 without any shared resource. We

Table 1 Requirements of target system without SR.

	S	*W*	*Pe*	*B*
P	RM	$\{V_0,V_1\}$	∞	∞
V_0	EDF	$\{T_0,T_1\}$	10	4
V_1	RM	$\{T_2,T_3,T_4\}$	5	3
	Pe	*E*	*D*	*O*
T_0	15	1	15	0
T_1	15	4	15	0
T_2	10	2	10	0
T_3	15	4	15	0
T_4	20	1	20	0

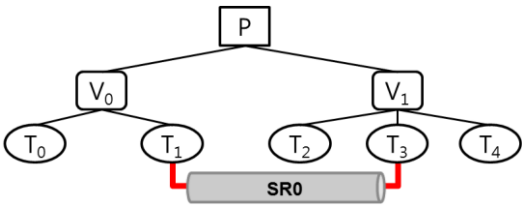

Fig. 4 Topology of target system with SR.

Table 2 Preemption count.

	Intra-scheduler preemption	Inter-scheduler preemption	Total preemption
T_0	1	2	3
T_1	1	3	4
T_2	1	0	1
T_3	2	1	3
T_4	3	1	4

Table 3 Updated information of target system

	S	*W*	*Pe*	*B*
V_0	EDF	$\{V_2,T_0,T_1\}$	10	4
V_2	RM	$\{T_5,T_6\}$	20	1
	Pe	*E*	*D*	*O*
T_5	100	1	100	0
T_6	50	1	50	0

set the CXS time = 86.917 μs and CMPD time = 139.12 μs as in [10] to derive a new E. The verification result is "Unsatisfied"; the counter example shows that the given budget of V_0 is not sufficient to schedule T_1 and T_2.

More than two-level hierarchical system can be specified and verified in a similar way. For example, the target system can be updated to a three-level hierarchical system as follow. A virtual scheduler V_2, has two tasks as its workload, is added to the workload of V_0. The workload of V_0 is updated from $\{T_0, T_1\}$ to $\{V_2, T_0, T_1\}$, and the information of V_2, T_5, and T_6 is added as is presented in Table 3. The system topology is same as Fig. 1. After update, the framework verifies the schedulability of the system using the TCTL sentence introduced above. The verification result of extended target system is "Satisfied."

4. Conclusion

In this paper, we proposed a schedulability analysis framework that can specify systems intuitively and verify whether they are always schedulable. The advantage of the proposed framework is that it can check the entire system; therefore, it is possible to specify the shared resources intuitively and count the number of preemption occurrences.

Further, in this paper, we presented a simple case study to show how the framework specifies a system and analyzes system schedulability. In the future, we intend to analyze avionic systems introduced in [5] and [9] with the proposed framework.

In a future study, we will upgrade the proposed framework to specify a system for multi-core and implement protocols for the shared resources to prevent priority inversion problems. Moreover, the proposed framework will be upgraded to automatically add the overhead time to the execution time when preemption occurs in order to obtain a more accurate result.

Acknowledgments

This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2015-H8501-15-1012) supervised by the IITP (Institute for Information & Communications Technology Promotion) and Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2012R1A1A2009354).

References

[1] T.-W. Kuo and C.-H. Li, "A fixed-priority-driven open environment for real-time applications," Proc. 20th IEEE Real-Time Systems Symposium 1999, pp.256–267, IEEE, 1999.

[2] G. Lipari and S.K. Baruah, "Efficient scheduling of real-time multi-task applications in dynamic systems," Proc. IEEE, pp.166–175, 2000.

[3] G. Lipari, J. Carpenter, and S. Baruah, "A framework for achieving inter-application isolation in multiprogrammed, hard real-time environments," Proc. 21st IEEE Real-Time Systems Symposium, pp.217–226, IEEE, 2000.

[4] I. Shin and I. Lee, "Compositional real-time scheduling framework with periodic model," ACM Trans. Embedded Computing Systems (TECS), vol.7, no.3, p.30, 2008.

[5] J. Boudjadar, et al., "Hierarchical scheduling framework based on compositional analysis using UPPAAL," The 10th International Symposium on Formal Aspects of Component Software, LNCS 8348, pp.61–77, 2013.

[6] L. Carnevali, A. Pinzuti, and E. Vicario, "Compositional verification for hierarchical scheduling of real-time systems," IEEE Trans. Softw. Eng., vol.39, no.5, pp.638–657, 2013.

[7] A. Easwaran, et al., "A compositional framework for avionics (ARINC-653) systems," Technical Report, Department of Computer & Information Science, University of Pennsylvania, no.MS-CIS-09-04, 2009.

[8] G. Behrmann, A. David, and K.G. Larsen, "A Tutorial on Uppaal," Formal Methods for the Design of Real-Time Systems, Lecture Notes in Computer Science, vol.3185, pp.200–236, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[9] J. Park, et al., "A process algebraic approach to the schedulability analysis and workload abstraction of hierarchical real-time systems,"

Journal of Logical and Algebraic Methods in Programming (submitted).

[10] L.T.X. Phan, M. Xu, J. Lee, I. Lee, and O. Sokolsky, "Overhead-aware compositional analysis of real-time systems," 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), pp.237–246, 2013.