5-2-2018

Towards Overhead-Free Interface Theory for Compositional Hierarchical Real-Time Systems

Jin Hyun Kim

University of Pennsylvania, jinhyun@seas.upenn.edu

Kygong Hoon Kim

Gyeongsang National University, khkim@gnu.ac.kr

Arvind Easwaran

Nanyang Technological University, arvinde@ntu.edu.sg

Insup Lee

University of Pennsylvania, lee@cis.upenn.edu

Follow this and additional works at: https://repository.upenn.edu/cis_reports

Part of the [Computer Engineering Commons](#), and the [Computer Sciences Commons](#)

Recommended Citation

Jin Hyun Kim, Kygong Hoon Kim, Arvind Easwaran, and Insup Lee, "Towards Overhead-Free Interface Theory for Compositional Hierarchical Real-Time Systems", . May 2018.

MS-CIS-18-02

This paper is posted at ScholarlyCommons. https://repository.upenn.edu/cis_reports/1019
For more information, please contact repository@pobox.upenn.edu.

Towards Overhead-Free Interface Theory for Compositional Hierarchical Real-Time Systems

Abstract

Much recent research has been conducted on compositional real-time scheduling framework as the framework becomes a useful fundamental theory for real-time OS-Hypervisor. Much recent research has been conducted on compositional real-time scheduling as the framework becomes a useful fundamental theory for real-time OS-Hypervisor. However, compositional frameworks suffer from abstraction overheads in composing components. In this paper, we classify the composition overheads into i) supply abstraction overhead associated with the supply from a resource provider, and ii) demand abstraction overhead associated with the component workload. Then, we provide sufficient conditions for each abstraction overhead to be eliminated. In addition, this paper provides a heuristic technique that transforms any component to satisfy the sufficient conditions so that the abstraction overheads are minimized. In our comparison, we showed our technique outperforms the prior overhead-reducing CF about 10% at average and other combination techniques about 8% in reducing the overhead.

Disciplines

Computer Engineering | Computer Sciences

Comments

MS-CIS-18-02

Contents

1	Introduction	2
2	Preliminaries	4
2.1	Models and Definitions	4
2.2	Underlying CF	6
2.3	Assumptions and Problem Description	7
3	Overheads of Compositional Framework	7
3.1	Overhead Classification	7
3.2	Supply Abstraction Overhead	8
3.2.1	Implementing IPRM	9
3.3	Demand Abstraction Overhead	11
3.3.1	Removing bDAO	12
3.4	Composition Overhead-Free Components	12
4	Hierarchical Scheduling Systems with Sync	14
4.1	Harmonic Composition of a Component under RM	14
4.2	Divisible Composition of a Component under EDF	14
4.3	Overhead-Free Hierarchical Composition	15
5	Compositional Framework-SYNC for Non-Harmonic Tasks	15
5.1	Proximation of Non-Harmonic Workload Tasks	16
5.2	Harmonic Interface Generation	17
5.3	Utilization Bound Computation in using Proxy Tasks	19
6	Performance Evaluation	21
6.1	Utilization Bound of Proximation	21
6.2	Composition Overhead of Independent Components	22
6.3	Composition Overhead	23
6.4	Runtime Overhead of HSS	24
7	Discussion and Conclusions	25

Towards Overhead-Free Interface Theory for Compositional Hierarchical Real-Time Systems

Jin Hyun Kim¹, Kyong Hoon Kim², Arvind Easwaran³, and Insup Lee⁴

¹ Univ. of Pennsylvania, Philadelphia, USA, jinhyun@seas.upenn.edu

² Gyeongsang National Univ., Jinju, Korea, khkim@gnu.ac.kr

³ Nanyang Technological Univ., Nanyang, Singapore, arvinde@ntu.edu.sg

⁴ Univ. of Pennsylvania, Philadelphia, USA, lee@cis.upenn.edu

Abstract

Much recent research has been conducted on compositional real-time scheduling framework as the framework becomes a useful fundamental theory for real-time OS-Hypervisor. Much recent research has been conducted on compositional real-time scheduling as the framework becomes a useful fundamental theory for real-time OS-Hypervisor. However, compositional frameworks suffer from abstraction overheads in composing components. In this paper, we classify the composition overheads into i) *supply abstraction overhead* associated with the supply from a resource provider, and ii) *demand abstraction overhead* associated with the component workload. Then, we provide sufficient conditions for each abstraction overhead to be eliminated. In addition, this paper provides a heuristic technique that transforms any component to satisfy the sufficient conditions so that the abstraction overheads are minimized. In our comparison, we showed our technique outperforms the prior overhead-reducing CF about 10% at average and other combination techniques about 8% in reducing the overhead.

1998 ACM Subject Classification Dummy classification – please refer to <http://www.acm.org/about/class/ccs98-html>

Keywords and phrases Compositional Framework, Hierarchical System, Overhead Minimization

Digital Object Identifier 10.4230/LIPICs...

1 Introduction

A hierarchical scheduling system (HSS) is a component-based scheduling system wherein a component distributes to sub-components the resource that it receives from its parent component or the hardware platform. Nowadays, various applications of such a system can be found in avionics, automotive, virtual machines (Xen hypervisor, VMWare), etc. [10, 19, 6, 15]. The compositional framework (CF) is a development and analysis framework for HSS based on the component-based approach. The study of CF has been developed in several directions: new resource models for characterizing resource supply on single and multi-cores [18, 23, 7, 5, 21, 3], optimization for improving system efficiency [14, 24, 8, 6, 4], share-resource communication [2], cache-aware scheduling and implementation overheads, and so on. A resource model of CF is a supply abstraction model and an interface of CF is a demand abstraction of a given workload. They are prone to the abstraction overheads, *i.e.*, the resource is under-utilized due to the abstractions. In the following motivating problem, we give an example.

© Jin Hyun Kim and Kyong Hoon Kim;
licensed under Creative Commons License CC-BY

Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

■ **Figure 1** An example of hierarchical real-time systems

Motivating Problem

Let us consider an OS-Hypervisor operating two virtual machines using the scheduling algorithm RM (Rate Monotonic), as shown in Figure 1a. The virtual machines schedule tasks using RM and EDF (Earliest Deadline First), respectively. Here, the system can be modeled by the compositional scheduling framework using a Periodic Resource Model (PRM) [24], which provides resource periodically. Since the framework considers the worst-case for resource supply, PRM computes sufficient resource budgets of C_1 and C_2 as 3 units for 4 time units ($I_1 = (4,3)$) and 5 units for 8 time units ($I_2 = (8,5)$), respectively. In this case, the compositional scheduling framework does not provide a feasible solution. However, the system is schedulable due to the harmonic property of resource and task periods, as shown in Figure 1c. In particular, Simon *et al.* [12] observed that around 1000-1500 runnables of automotive applications can be divided into 10 groups, each of which uses a common period and these group periods are mostly harmonic with each other. It turns out that tasks in a group using the same period benefit from the harmonicity. The above observation motivates this work, so that harmonicity is applied to the CF, that prior studies have not paid much attention to, for improving resource utilization of HSS.

To improve resource utilization of HSS, Easwaran *et al.* [7] introduced a new resource model, Explicit Deadline Period (EDP) resource model, by extending the periodic resource model (PRM) of Shin and Lee [23, 22] with deadline. The EDP resource model identifies the minimum resource needed to schedule component's tasks using a periodic resource model. Easwaran *et al.* [7, 6], however, do not present how to implement the resource supply model in parent component to satisfy the demand abstraction of component interfaces. Chen *et al.* [4] proposed a way of using the same period for every interface and thus eliminated abstraction overheads from intermediate components. However, they didn't take into account the increase in runtime overheads such as context switches, as a result of using a common period that must be smaller than or equal to the smallest period of component tasks [20]. Recently, Guo *et al.* [11] presented how to apply harmonicity to independent scheduling components to remove interface abstraction overheads, wherein component tasks adjust their periods to be harmonic with the component interface. However, Guo *et al.* haven't shown how to apply harmonicity to dependent components of HSS in a parent-child component relationship.

This paper presents a novel CF where a demand abstract represents workload's demand completely independent from a resource model and analyze the root cause of abstraction

overheads in the CF. Based on the analysis, we present sufficient conditions to remove these overheads. Using the sufficient conditions, we then present a hierarchical scheduling system, called Hierarchical Scheduling System-Sync (HSSS), wherein no abstraction overhead is incurred when composing interface and resource supply tasks. In addition, we present how to utilize our novel CF for non-harmonic tasks to benefit from harmonicity. As a result, we show our novel CF improves resource utilization using harmonic property of real-time tasks, compared with the prior work of CF.

The contribution of this paper are to

- Propose a novel CF and analyze the root causes of the CF's overheads,
- Identify sufficient conditions to eliminate demand and supply abstractions overheads, respectively,
- Develop overhead-free hierarchical composition based on the harmonicity property, and
- Develop algorithms to approximate non-harmonic task by harmonic task and to generate harmonic interface

Organization

Section 2 presents the underlying model and problems that we address in this paper. In Section 3, we present sufficient conditions to remove demand and supply abstraction overheads. Section 4 provides a new hierarchical system, called Hierarchical Scheduling System-Sync (HSSS) removing abstraction overheads. In the following, we show how to apply HSSS for components comprising non-harmonic tasks such that the abstraction overheads are minimized. In Section 6, we compare our framework with various prior techniques and their combinations. Section 7 discusses the lessons for this work and conclude this paper.

2 Preliminaries

This section discusses preliminaries underlying this work; the setting and definitions of the underlying system, and the problem that we address in this paper.

2.1 Models and Definitions

The notations we use in this paper are defined in Table 1. We use the periodic task model [17], $\tau = (p, e)$, where p is the period and e is the worst-case execution time. A resource model $\mathcal{R}$ captures a resource supply pattern. For instance, the periodic resource model (PRM) $\Gamma(\Pi, \Theta)$ of [23] models a periodic resource supply that supplies Θ resources every Π time units. A resource model can be regarded as a resource supply task of a component when it is used to operate a sub-component, *i.e.*, when the task is scheduled by the scheduling algorithm in the component, the sub-component associated with the task can run. A hierarchical scheduling system (HSS) is defined by a tuple $HSS = (M, Supply)$, comprising the set M of components and a function of $Supply : T \mapsto M$ mapping a resource supply task to a component. A component M is composed of a workload $W \subset T$ and an algorithm, and HSS associates a task of W with a M in supply-demand relation. A task is called *elementary task* if it is mapped to no sub-component. Similarly, a component enclosing only elementary tasks is called an *elementary component* or *leaf component*. A component exposes a collective resource requirement of its workload through a *periodic interface* $I(\mathbb{P}, \mathbb{B})$ meaning that I requires $\mathbb{B}$ resource every $\mathbb{P}$ time units.

We use harmonic properties of real-time tasks [13]. For harmonic properties, we define the following two functions:

■ **Table 1** Symbol definitions

Symbols	Definitions
A	Scheduling algorithm
$\mathcal{T} = \{\tau_i \mid \tau_i(p_i, e_i)\}$	The set of tasks, where p is a period and e is the worst-case execution time
$\mathcal{W} = \{\tau_i \mid \tau_i(p_i, e_i)\}$	Workload of a component, a set of tasks, where p is a period and e is a worst-case execution time
$\mathcal{M} = \{\mathcal{C}(\mathcal{W}, \mathcal{A})\}$	The set of components, where $\mathcal{C}$ is a component comprising of $\mathcal{W}$ and $\mathcal{A}$
$Supply : \mathcal{T} \mapsto \mathcal{M}$	Map from a resource-supply task to a sub-component
$\Gamma(\Pi, \Theta)$	Periodic Resource Model (PRM) with Π period and Θ budget
$\mathcal{H}(\Pi, \Theta)$	Ideal Periodic Resource Model (IPRM) with Π period and Θ budget
$\mathcal{I}(\mathbb{P}, \mathbb{B})$	Periodic interface with $\mathbb{P}$ period and $\mathbb{B}$ budget
$\mathcal{D}(\mathbb{P}, \mathbb{E})$	Periodic demand abstract with $\mathbb{P}$ period and $\mathbb{E}$ worst-case execution time
$U_{\mathcal{W}}, U_i$	Utilization of workload $\mathcal{W}$ and Utilization of a task τ_i
$UB_{\mathcal{R}}$	Resource utilization bound of the resource model $\mathcal{R}$
UB_{τ}	Task resource utilization bound

► **Definition 1** (Harmonic Tasks). For tasks τ_i and τ_j ,

$$\mathcal{H}(\tau_i, \tau_j) \begin{cases} 1 & \text{if } p_i \bmod p_j = 0 \vee p_j \bmod p_i = 0 \\ 0 & \text{Otherwise} \end{cases}$$

Two tasks τ_i and τ_j are said to be *harmonic* if and only if $\mathcal{H}(\tau_i, \tau_j) = 1$. $\mathcal{H}$ can be used by other component with periods, such as resource model $\Gamma(\Pi, \Theta)$. For instance, $\mathcal{H}(\Gamma, \tau) = 1$ if Π and p are harmonic.

► **Definition 2** (Harmonic Workload). A workload or task set $\mathcal{W}$ is harmonic if every pair of tasks in the set is harmonic, *i.e.*,

$$\forall \tau_i, \tau_j \in \mathcal{W}, \quad \mathcal{H}(\tau_i, \tau_j) = 1$$

We define two metrics to evaluate the overhead of resource supply against resource demand as follows:

► **Definition 3** (Composition Overhead). Given a resource model $r \in \mathcal{R}$ and a workload $\mathcal{W}$, composition overhead is defined by

$$O^{\text{com}}(\mathcal{R}, \mathcal{C}) = \frac{U_{\mathcal{R}} - U_{\mathcal{W}}}{U_{\mathcal{R}}}$$

where $U_{\mathcal{R}}$ is the utilization of the resource model r and $\mathcal{W}$ is the workload of $\mathcal{C}$. For instance, $U_{\Gamma(\in \mathcal{R})} = \Theta/\Pi$ for $\Gamma(\Pi, \Theta)$.

For a given HSS, the system overhead denotes the sum of all component abstraction overheads.

► **Definition 4** (System Overhead). For a given HSS, the system overhead is defined by

$$O^{\text{sys}} = \sum_{\mathcal{C}_i \in \mathcal{M}, \tau_j = \text{Supply}^{-1}(\mathcal{C}_i)} O^{\text{com}}(\tau_j, \mathcal{C}_i) = U_{\mathcal{W}_{top}} - \sum_{\mathcal{W}_i \in \{\mathcal{W}_{elem}\}} U_{\mathcal{W}_i}$$

where W_{top} is the workload of the top (root) component and W_{elem} is an elementary component.

2.2 Underlying CF

Basically, the component interface of CF represents the demand of a component and abstracts details of the component, such as workload, task types, scheduling algorithm. The resource model represents the resource supply and abstracts the detail on what and how to supply resource, such as configuration of a resource supplying component. The interface and resource model can be captured using various models, such as sporadic task model, implicit or explicit periodic task model [1].

The CF we consider satisfies *compositionality* and *composability*. For any given two components, a CF is compositional if it follows the compositional rules:

$$\text{Composition Rule : } \frac{C_1 \hookrightarrow I_1, C_2 \hookrightarrow I_2, \mathcal{R}_1 \vdash I_1, \mathcal{R}_2 \vdash I_2, \\ C_{12}[\mathcal{R}_1 \parallel \mathcal{R}_2] \hookrightarrow I_{12}, \mathcal{R}_{12} \vdash I_{12}}{\mathcal{R}_{12} \vdash C_1 \parallel_C C_2}$$

where $\hookrightarrow$ is an abstraction function from a component C to an interface I , $\vdash$ is a verifier that checks if the demand of a given interface I is less than or equal to the supply of a given $\mathcal{R}$, $C[\]$ denotes the workload that the component C comprises, and $\parallel_C$ is a composition of components. For the two components C_1 and C_2 , the abstraction function $\hookrightarrow$ abstracts their workloads into interface I_1 and I_2 , respectively. If there exist two resource models $\mathcal{R}_1$ and $\mathcal{R}_2$ that satisfy I_1 and I_2 , they are composed into an interface I_{12} under a component C_{12} , and there exist a resource model $\mathcal{R}_{12}$ that satisfies the composite interface I_{12} , then the two components C_1 and C_2 are composed such that they are schedulable by the supply of $\mathcal{R}_{12}$. According to the composition rule, CF needs to provide two operators: An abstraction operator that abstracts the workload of a component into an interface and a verifier that checks if a given resource model satisfies an interface.

The CF meets composability for interface composition. The interface of CF needs to be able to be interpreted under a standard scheduling algorithm, such as deadline-driven, fixed-priority, etc. In other words, the interface should be able to be converted into a task schedulable under a scheduling algorithm. For instance, a periodic task of components in [23] is abstracted by interface, which is interpreted into a periodic task by parent component. However, the static resource partition model in [18] is not composable since the resource partition cannot be abstracted by an interface that can be interpreted by a standard

The CF we consider meets the two constraints: syntactic and semantic constraints. The syntactic constraint restricts the representation of interface and resource model, which should be able to specify what to demand and supply. For instance, PRM $\Gamma(\Pi, \Theta)$ specifies a supply amount of resources Θ and a supply period Π . Similarly, a periodic interface $I(\mathbb{P}, \mathbb{B})$ states the demand amount $\mathbb{B}$ of resources and the demand period $\mathbb{P}$ of resources. The resource model and interface in CF should satisfy given syntactic constraints.

The semantics constraint restricts the interpretation of interface and resource model's representations. Given an interface and resource model, CF should be able to give a formal interpretation on how to demand and supply resource in specific behaviors. For instance, the interface, a demand representation, can be interpreted by **dbf** as if the demand of a workload is represented by **dbf**. The resource model, a supply representation, can be interpreted by **sbfc**.

2.3 Assumptions and Problem Description

In this paper, we present a new periodic abstract demand, based on the periodic task model [17], analyze the overhead of the periodic demand abstractions, and present techniques for their minimization and elimination. In a prior CF [22], a resource supply model is computed based on details of a given component modeled by `dbf`, not an abstract interface, and a component interface is generated from a resource supply that is sufficient to satisfy the demand of the component. In such a framework, the interface, which represents the demand of a component, relies on a particular resource model being used at its parent level, and thus, could include a supply-dependent overhead. One of our goals is to have explicitly separate demand abstraction and supply abstraction.

Based on the following assumptions: (1) HSS that we focus on is an open system, where the analysis of component is independent from other components; (2) the demand and the supply are not synchronized, thus the supply can occur when no demand appears; and (3) the demand and supply abstraction in this paper is based on the periodic task model [16], this paper analyzes the root cause of CF's abstraction overheads.

Our goal of this work is to minimize O^{sys} of a HSS. To the end, we answer the following questions: (1) How can we achieve an ideal supply for a periodic supply model? (2) What are sufficient conditions to remove the overhead of a periodic demand abstract? (3) How can we utilize the ideal periodic resource supply models and sufficient conditions removing the demand abstract overhead for any given task sets?

3 Overheads of Compositional Framework

In this section, we limit a CF where the overhead we discuss in this paper. Afterwards, discusses two abstraction overheads of CF and ways to tackle them.

3.1 Overhead Classification

The CF is prone to abstraction overheads since the demand of a component is abstracted by an interface and the supply is abstracted by a resource model. Such abstractions give rise to overheads, *i.e.*, the resulting HSS requires additional resources. The causes of the overheads are various, such as abstraction of the preemption by interleaving components and asynchronous demand and supply, and so on.

Besides, scheduling algorithms have different schedulability. For instance, EDF schedules task sets under 100% resource utilization, regardless of number of tasks, on a single-core processor. RM has different CPU utilization bounds according to the number of tasks, *i.e.*, $n(2^{1/n} - 1)$ where n is the number of tasks. Compared to EDF, RM can guarantee the scheduling of tasks who total CPU utilization less than required by EDF. We call such a difference in the CPU utilization bound as scheduling algorithm overhead.

The runtime overhead is critical in an HSS since an HSS schedules recursively-enclosed sub-components. The runtime overheads are critical in an HSS since an HSS schedules recursively-enclosed sub-components. Nevertheless, the scheduling algorithm overhead and the runtime overheads are out of scope of this paper. In this paper, we focus on the overheads associated with demand and supply abstraction in CF.

The subsections describe demand and supply abstraction overheads in CF and provide sufficient conditions to remove each of them.

3.2 Supply Abstraction Overhead

Under the assumption of asynchronous supply and demand, supply abstraction overhead (SAO) of CF is an overhead incurred for the following reasons:

■ **Figure 2** A PRM supply

Figure 2 shows the worst-case supply delay for PRM [23]. Here, the demand arrives just after the supply of the first period has finished, because the demand and the supply are not necessarily synchronized. Further, the supply of PRM can be delayed for at most $\Pi - \Theta$ since it may be preempted by other interleaving components. The interval $\Pi - \Theta$ of the second period in Figure 2 depicts the largest preemption time. The worst-case starvation of PRM is known to be $2(\Pi - \Theta)$, which may happen because the demand and the supply are not synchronized ($\Pi - \Theta$ of the first period of Figure 2) and the supply is delayed as much as possible ($(\Pi - \Theta$ of the second period of Figure 2)) by being preempted.

We now define an ideal periodic resource model (iPRM) $H(\Pi, \Theta)$ as follows:

► **Definition 5.** For a given period Π and execution time Θ , a *ideal periodic resource model* (iPRM), denoted by $H(\Pi, \Theta)$, is a periodic resource model that guarantees Θ resources for any time interval of duration Π .

Note that, by definition, the worst-case starvation interval (WCSI) of iPRM $H(\Pi, \Theta)$ is $\Pi - \Theta$. As mentioned early, The WCSI of the supply of PRM $\Gamma(\Pi, \Theta)$ is $2(\Pi - \Theta)$ [23]. The WCSI of Explicit Deadline Periodic (EDP) resource model is $\Pi + \Delta - 2\Theta$, where Δ is the deadline of a periodic supply [7]. Therefore, for a given period, iPRM is an idea periodic resource model among the resource model based on the periodic task model. The *sb*f of iPRM from [7] is defined as follows:

$$\text{sb}_{\text{iPRM}}(H, t) = \left\lfloor \frac{t}{\Pi} \right\rfloor \cdot \Theta + \max(0, t - (\Pi - \Theta) - \left\lfloor \frac{t}{\Pi} \right\rfloor \cdot \Pi) \quad (1)$$

One choice of an optimal iPRM is processor sharing. We call it, $\Omega(\Pi, \Theta)$.

We now define the supply abstraction overhead (SAO) as follows:

► **Definition 6** (Supply Abstraction Overhead (SAO)). For a periodic resource model $\mathcal{R}(\Pi, \Theta)$ and a time interval t , the supply abstraction overhead of $\mathcal{R}$ is the difference between worst-case supplies of $\Omega(\Pi, \Theta)$ and $\mathcal{R}(\Pi, \Theta)$.

For a periodic resource model $\mathcal{R}$, the SAO is computed by

$$\text{O}^{\text{sup}}(\mathcal{R}, t) = \int_0^t \text{sb}_{\text{iPRM}}(\Omega, x) - \text{sb}_{\mathcal{R}}(\mathcal{R}, x) dx \quad (2)$$

where $\mathcal{R}$ is a periodic resource model for given Π and Θ . Periodic resource models may depend on different supply bound functions for the same parameters. For instance, the PRM of [23] and the iPRM depend on different *sb*fs.

The resource model with less SAO makes more tasks feasible. For instance, Figure 3 shows the SAOs of iPRM $H(10, 3)$ and PRM $\Gamma(10, 3)$. The shaded regions show that iPRM is more resource efficient than PRM. In the following, we present sufficient conditions for a PRM to be an iPRM for a given period.

■ **Figure 3** Supply abstraction overhead

3.2.1 Implementing iPRM

The static partitioned resource model [18] guarantees a fraction of supply resource with a bounded delay. In the static resource model, the supply pattern repeats in the same manner every period. The first sufficient condition to obtain an iPRM from a PRM is related to the regularity of periodic supply. In this paper, a regular PRM Γ is defined as follows:

► **Definition 7.** A PRM $\Gamma(\Pi, \Theta)$ is regular if it provides resource at the same time points within every period.

Here, we have a sufficient condition for a PRM Γ to be an iPRM in following lemma:

► **Lemma 8.** If a PRM $\Gamma(\Pi, \Theta)$ is regular, it is an iPRM.

Proof. Let $\text{supply}(\Gamma, t_0, t)$ compute the amount of resources supplied by Γ during $[t_0, t_0 + t]$, i.e., an interval t starting at t_0 . For $\Gamma(\Pi, \Theta)$ and $0 < t_i \leq \Pi$ and $n \in \mathbb{N}_{>0}$, $\text{supply}(\Gamma, (n-1) \cdot \Pi, t_i) + \text{supply}(\Gamma, (n-1) \cdot \Pi + t_i, \Pi - t_i) = \Theta$ by the definition of supply . For any two consecutive periods of a regular Γ , the Γ supplies resources such that $\text{supply}(\Gamma, (n-1) \cdot \Pi, t_i) = \text{supply}(\Gamma, n \cdot \Pi, t_j)$ and $\text{supply}(\Gamma, (n-1) \cdot \Pi + t_i, \Pi - t_i) = \text{supply}(\Gamma, n \cdot \Pi + t_j, \Pi - t_j)$, where $t_i + \Pi = t_j$. Then, $\text{supply}(\Gamma, (n-1) \cdot \Pi + t_i, \Pi - t_i) + \text{supply}(\Gamma, n \cdot \Pi, t_j) = \Theta$ since $\text{supply}(\Gamma, (n-1) \cdot \Pi, t_i) + \text{supply}(\Gamma, (n-1) \cdot \Pi + t_i, \Pi - t_i) = \Theta$. Hence, a regular Γ always provides Θ during a time interval Π , thus Γ is an iPRM. ◀

Now, we present a sufficient condition to enforce a PRM to be regular. Under RM, tasks with harmonic periods are scheduled at the same points in each period. If harmonic period tasks are scheduled under EDF and their ties are identically broken, then each task also executes at the same points in each period. Thus, a PRM implemented by a task under the above harmonic conditions is regular if the PRM is serving the top (root) component fully utilizing resources.

► **Lemma 9.** If tasks are harmonic under RM, then each task has the execution at the same time within its period.

Proof. Let τ_1 and τ_2 harmonic tasks under RM such that a shorter period task is given to τ_1 and a larger period is given to τ_2 . Since τ_1 has higher priority over τ_2 under RM and they are harmonic, τ_2 includes n executions of τ_1 , where $n \geq \mathbb{N}_{>0}$. The highest priority task τ_1 always begins jobs at the beginning of its period and finishes without preemption. Thus, the highest priority task has the same execution at every period. For the second highest priority task τ_2 , τ_1 always preempts at the same points for the above reason whenever τ_2 runs. τ_2 runs at the remaining time after τ_1 finishes jobs at every the period of τ_1 . Since τ_1 executes jobs at the same point every period, the remaining execution time is always the same at every period of τ_1 . Thus, τ_2 has the same execution at every its period, which executes at

■ **Figure 4** An HSS using iPRM

the remaining time of τ_1 . For the rest of the tasks, they execute at the same points for the above same reason. ◀

► **Lemma 10.** *If tasks are harmonic under EDF and their ties are broken identically such that a shorter period task is given a higher priority and one of two tasks with the same period precedes the other, then each task has the execution at the same time within its period.*

Proof. Let τ_1 and τ_2 harmonic tasks under EDF of which ties are broken identically such that a shorter period task (τ_1) is given a higher priority. τ_2 includes n executions of τ_1 , where $n \geq \mathbb{N}_{>0}$ and τ_1 always precedes τ_2 since they are harmonic and τ_1 is given priority over ties. Thus, τ_1 always performs the same execution every its period. Thus, it has the same execution points at every period. τ_2 can execute at the remaining time during the period of τ_1 after τ_1 finishes job. Since τ_1 has the same execution points at every its period for the above reasons, τ_2 executes at the same remaining time after τ_1 finishes job, and τ_1 executes exactly n times during the period of the τ_2 , thus τ_2 also has the same execution points. For the rest of tasks under EDF, they have the same executions every period for the same reason above. ◀

For instance, the resource supply tasks τ_1 and τ_2 in Figure 4 are harmonic at the top component, thus they are regular. Then, C_1 and C_2 supplied by τ_1 and τ_2 can utilize as much resource as individual resource utilization of τ_1 and τ_2 .

In the following lemma, we provide conditions that a resource supply task τ_i becomes regular when it is also supplied by a parent component Γ :

► **Lemma 11.** *For a given regular $\Gamma(\Pi, \Theta)$ supplying resources for W and each $\tau_i \in W$, τ_i is regular if*

$$(\forall \tau_i \neq \tau_j, \tau_j \in W, \mathcal{H}(\tau_i, \tau_j) = 1 \wedge \mathcal{H}(\tau_i, \Gamma) = 1) \wedge U_W \leq U_\Gamma \quad (3)$$

Proof. Since W is harmonic, the base-case and worst-case response time of each task of W is $e_i + \sum_{k \in HP(i)} e_k/p_i$. The resource supply of regular Γ always is proven to be an iPRM by Lemma 8, so the delay of its resource supply is always the same. Suppose that the supply delay of Γ is ϵ , the best-case and worst-case response time of each task of W supplied by Γ cannot be smaller or larger than $e_i + \sum_{k \in HP(i)} e_k/p_i + \epsilon$. ◀

Figure 4 shows that all resource supply tasks are harmonic in each component that their supply follows a iPRM. C_1 enclosing τ_3 and τ_4 is supplied with resources by τ_1 . Then, τ_3

and τ_4 supply resources to C_3 and C_4 . Since τ_1 , τ_3 and τ_4 are harmonic, the supply tasks, τ_3 and τ_4 , are regarded as iPRM.

3.3 Demand Abstraction Overhead

Demand abstraction overhead (DAO) is generated when a component workload is transformed into an abstract form. The CF's interface, a demand abstract, such as [23, 7], includes SAO since it is computed using sbf of resource models. We define a *periodic demand abstract* $D(P, E)$, based on the periodic task model [17], such that it does not incur supply-related overheads like WCSI. It is defined as follows:

► **Definition 12** (Periodic Demand Abstract (PDA)). For a workload W under scheduling algorithm A , we say (P, E) is a periodic demand abstract, denoted by $D(P, E)$, if whenever a resource supply can schedule D , it can also schedule W .

PDA is a supply-independent interface different from interfaces [23, 7] that rely on a resource supply model. Hence, it does not incur any supply-relevant overheads. To represent the demand of the PDA, we adopt the dbf of [9] as follows: For a PDA $D(P, E)$ and a time interval t , the dbf of PDA is defined by $\text{dbf}_{\text{PDA}}(D, t) = \lceil t/P \rceil E$.

■ **Figure 5** $W_{EDF} = \{\tau_1(10, 2), \tau_1(20, 4), \tau_3(40, 4)\}$ abstracted by $D(5, 2.5)$ and $D(10, 5)$

Figure 5 shows that two PDAs, $D(5, 2.5)$ and $D(10, 5)$, suffice to represent the demand of τ_1, τ_2 , and τ_3 under EDF, because any supply satisfying the demand of the PDAs can satisfy the demand of W . Nonetheless, some supply satisfying the demand of $D(10, 5)$ may not satisfy the demand of $D(5, 2.5)$ even though they have the same resource utilization. For instance, $H(10, 5)$ can satisfy $D(10, 5)$, but not $D(5, 2.5)$. Thus, we can say that $D(10, 5)$ demands relatively less resources than $D(5, 2.5)$. Based on the above observation, DAO is formally defined as follows:

► **Definition 13** (Demand Abstraction Overhead (DAO)). For a given workload W and a PDA $D(P, E)$, the demand abstraction overhead is defined as the difference between the demand of D and the original demand of W . For given W , D , and t , the DAO of D w.r.t. W is computed by

$$O^{\text{dem}}(W, D, t) = \int_0^t (\text{dbf}_D(D, x) - \text{dbf}_A(W, x)) dx$$

Note that DAO is independent from SAO as it is in terms of dbf. DAO incorporates the overhead incurred by abstracting a workload into a singleton task.

As can see in Figure 5, it is usually unavoidable to have non-zero DAO when a workload is abstracted by a singleton task. Now, we define a bandwidth-DAO (bDAO) of PDA in

terms of resource utilization. For a workload W and a PDA $D(P, E)$, the bandwidth-DAO is defined by $O^{\text{bDEM}}(W, D) = U_W - E/P$.

A PDA with no DAO is said to be zero-DAO and a PDA with no bDAO is said to be zero-bDAO.

3.3.1 Removing bDAO

In [14], it is known that a workload under EDF scheduled by a resource supply is schedulable if the component interface period divides workload's periods and the total utilization of the workload is less than or equal to that of the resource supply. We apply this condition to remove bDAO from PDA abstraction of a workload under EDF.

For a workload W under EDF, a PDA has zero-bDAO, if it requires the same resource bandwidth as U_W and PDA's period divides periods of all the tasks in W .

► **Lemma 14** (from [14]). *For a workload W under EDF, a PDA $D(P, E)$ is zero-bDAO if*

$$(U_W = \frac{E}{P}) \wedge (\forall \tau_i \in W, p_i \bmod P = 0)$$

Now, we present a condition to eliminate bDAO from PDA abstraction of a workload scheduled under RM. The harmonic task set under RM can make full use of resources [13]. The following lemma shows that if all the tasks in a workload are harmonic with each other and PDA, and the PDA has the same resource utilization as the workload, then the PDA has no abstraction overhead.

► **Lemma 15.** *For a W under RM, a PDA $D(P, E)$ is zero-bDAO if*

$$(U_W = \frac{E}{P}) \wedge (\forall \tau_i \in W, \mathcal{H}(\tau_i, P) = 1)$$

Proof. Since W is under RM and harmonic, it requires resources at the rate of U_W . Since $\forall \tau_i \in W, \mathcal{H}(\tau_i, D) = 1, \forall \tau_i \in W, \frac{p_i}{P} \in \mathbb{N}$. τ_i requires the amount $\frac{e_i}{p_i} \cdot P$ resources at every P . Thus, all $\tau_i \in W$ require the amount $\sum_{\tau_i \in W} \frac{e_i}{p_i} \cdot P$ resources at every P . Let $E = \sum_{\tau_i \in W} \frac{e_i}{p_i} \cdot P$. W requires $\frac{E}{P}$ at every P . Thus, $D = (P, E)$ sufficiently represents all resource requirements of W if W is harmonic and $U_W = \frac{E}{P}$. ◀

In Figure 4, since the periods of τ_7 and τ_8 under RM are harmonic with 5, 15, and 45, $D = (5, 1)$, $D = (15, 3)$, and $D = (45, 9)$ can be demand abstraction tasks with zero-bDAO. Similarly, the periods of τ_9 and τ_{10} under EDF can be divided by 5, 15, and 30, thus $D = (5, 2)$, $D = (15, 6)$, and $D = (30, 12)$ are demand abstraction tasks with zero-bDAO.

3.4 Composition Overhead-Free Components

In this section, we present sufficient conditions for a PDA to be schedulable by an interface, which is in turn schedulable by a iPRM resource model, so that an HSS can be built without abstraction overheads.

For a given iPRM, $H(\Pi, \Theta)$, a PDA schedulable by this iPRM satisfies the following conditions:

► **Lemma 16.** *A PDA $D(P, E)$ is schedulable by iPRM $H(\Pi, \Theta)$ if*

$$(\frac{E}{P} = \frac{\Theta}{\Pi}) \wedge (P \bmod \Pi = 0)$$

Proof. For $H(\Pi, \Theta)$, $\text{sbfiPRM}(H, t)$ is such that

$$\text{sbfiPRM}(H, k\Pi) = k\Theta \quad (4)$$

where $k \in \mathbb{N}_{>0}$ for the following reason: Since $t = k\Pi$ and $k \in \mathbb{N}_{>0}$,

$$\begin{aligned} \text{sbfiPRM}(H, k\Pi) &= \left\lfloor \frac{k\Pi}{\Pi} \right\rfloor \cdot \Theta + \max(0, k\Pi - (\Pi - \Theta) - \left\lfloor \frac{k\Pi}{\Pi} \right\rfloor \cdot \Pi) \\ &= \left\lfloor k \right\rfloor \cdot \Theta + \max(0, k\Pi - (\Pi - \Theta) - \left\lfloor k \right\rfloor \cdot \Pi) \\ &= k \cdot \Theta + \max(0, k\Pi - (\Pi - \Theta) - k\Pi) \\ &= k \cdot \Theta + \max(0, -(\Pi - \Theta)) \\ &= k \cdot \Theta \end{aligned} \quad (5)$$

■ **Figure 6** $\text{dbf}_{EDF}(D, t)$ and $\text{sbfiPRM}(H, t)$

Now, we prove the lemma as follows: $k\Pi = P$ since $P \bmod \Pi = 0$. $\frac{E}{P} = \frac{k\Theta}{k\Pi}$ since $\frac{E}{P} = \frac{\Theta}{\Pi}$ and $k\Pi = P$. $\text{sbfiPRM}(H, k\Pi) = k\Theta$ since $\forall t, O^{sup}(\Gamma, t) = 0$. Since $\frac{E}{P} = \frac{k\Theta}{k\Pi}$ holds as shown in Figure 6, D is schedulable by H . ◀

To generate an interface I incurring *zero-bDAO*, we have the following theorem and corollary.

► **Theorem 17.** *For an iPRM $H(\Pi, \Theta)$ and zero-bDAO $D(P, E)$, the interface $I(P, \mathbb{B})$ is schedulable if*

$$\left(\frac{E}{P} = \frac{\mathbb{B}}{P} = \frac{\Theta}{\Pi} \right) \wedge (P \bmod P = 0) \wedge (P \bmod \Pi = 0)$$

Proof. In Lemma 16, we show that D is schedulable by Γ such that $(\frac{E}{P} = \frac{\Theta}{\Pi}) \wedge (P \bmod \Pi = 0)$. Similarly, since $(\frac{E}{P} = \frac{\mathbb{B}}{P}) \wedge (P \bmod P = 0)$, I is a periodic demand abstract for D . Since I is also a periodic demand abstract, we can use Lemma 16 to show that I is schedulable by Γ such that $(\frac{\mathbb{B}}{P} = \frac{\Theta}{\Pi}) \wedge (P \bmod \Pi = 0)$. Thus, $I = (P, \mathbb{B})$ is schedulable if $(\frac{E}{P} = \frac{\mathbb{B}}{P} = \frac{\Theta}{\Pi}) \wedge (P \bmod P = 0) \wedge (P \bmod \Pi = 0)$. ◀

► **Corollary 18.** *For an iPRM $H(\Pi, \Theta)$ and a zero-bDAO $D(P, E)$, a component is composed with no abstraction overheads if*

$$\frac{E}{P} = \frac{\Theta}{\Pi} \wedge P \bmod \Pi = 0$$

Suppose that C_3 in Figure 4 has $D(45, 9)$ as a demand abstraction task and $\Gamma(5, 1)$ is given to C_3 as a resource model. Then, C_3 picks up one of $I(5, 1)$, $I(15, 3)$, and $I(45, 9)$ as its interface. Then, C_3 is composed with no abstraction overheads.

4 Hierarchical Scheduling Systems with Sync

Now, we show how an HSS free from abstraction overheads and component scheduling overheads is built, depending on scheduling algorithms, RM and EDF.

4.1 Harmonic Composition of a Component under RM

This section, we discuss a composition of a component under RM such that the composition has no abstraction overhead. First, we define a harmonic component as follows:

► **Definition 19.** A component with a PDA $D(P, E)$ is *harmonic* if the task periods in its workload and the PDA's period are harmonic, *i.e.*, all task periods in the workload and P are harmonic.

As indicated in Theorem 17, a PDA or an interface is zero-bDAO if $E = P \cdot U_W$ or $\mathbb{B} = \mathbb{P} \cdot U_W$. Let us assume that an IPRM $H(\Pi, \Theta)$ provides resources to a harmonic component. If the parameters of H are identical to D of the component, we can say the component abstraction incurs no abstraction overhead or is overhead-free. Thus, the following lemma presents overhead-free composition of a component with RM scheduling algorithm.

► **Lemma 20.** *For a given component C with RM scheduling algorithm, if the resource model $\Gamma(P, E)$ is an IPRM and $D(P, E)$ is harmonic with the workload, then the interface $I(P, E)$ has no abstraction overhead.*

Proof. Since a PDA $D(P, E)$ is harmonic with the workload, the PDA is zero-bDAO. If the same parameter (P, E) is used for H, D , and I , then they satisfy the three conditions of Theorem 17. ◀

The interface I derived from the zero-bDAO PDA is said to be *harmonic interface*.

4.2 Divisible Composition of a Component under EDF

A harmonic component with EDF algorithm can achieve overhead-free composition from Lemma 20. However, in case of EDF, we can use divisible constraint rather than the stricter harmonic constraint for abstraction overhead elimination.

► **Definition 21.** A component using PDA $D(P, E)$ is *divisible* if all task periods are individually divided by P .

As indicated in Theorem 17, the divisible interface produces no demand abstraction overhead if $E = P \cdot U_W$ under EDF algorithm. Thus, the following lemma presents overhead-free composition of a component with EDF algorithm.

► **Lemma 22.** *For a given component C with EDF scheduling algorithm, if the resource model $\Gamma(P, E)$ is an IPRM and $D(P, E)$ is divisible with the workload, then the interface $I(P, E)$ has no abstraction overhead.*

Proof. Since a PDA $D(P, E)$ is divisible with the workload under EDF algorithm, the PDA is zero-bDAO. If the same parameter (P, E) is used for H, D , and I , then they satisfy the three conditions of Theorem 17. ◀

The interface I derived from the zero-bDAO divisible PDA is said to be *divisible interface*.

■ **Figure 7** An example of HSSS

4.3 Overhead-Free Hierarchical Composition

Now, we define an overhead-free compositional framework with harmonic or divisible components. Lemmas 20 and 22 present overhead-free compositions of individual components under RM and EDF, respectively, using iPRM resource models. Then, we can define an overhead-free compositional framework with no abstraction overheads under the following conditions: If 1) the interface of a component under RM is harmonic or 2) the interface of a component under EDF is divisible, then the interface is zero-bDAO, *i.e.*, the interface has no demand abstraction overhead. If 3) a resource model is a resource supply task of a harmonic component, then the resource model becomes an iPRM, which has no supply abstraction overhead. If a CF satisfies the above 3 conditions, it has no abstraction overheads. For example, the system in Figure 4 meets the harmonicity condition for RM components and the divisibility condition for EDF components.

However, the resource model of an EDF component with a divisible interface is not an iPRM. In Figure 7, the component C_2 is a divisible component. However, tasks τ_5 and τ_6 cannot provide resources as iPRM because they are not regular under EDF.

► **Corollary 23.** *If each component in a hierarchical system is harmonic and the resource supply is the same as the interface, the composition has no abstraction overhead.*

In case of Corollary 23, the harmonicity condition includes the divisibility condition so that all components have no abstraction overheads. We call such a hierarchical system *Hierarchical Scheduling System-Sync* (HSSS) and use it for composing components with non-harmonic tasks. Next problem is how to make all component interfaces satisfy the above three conditions of HSS with no abstraction overhead. For a given hierarchical system, only task information and hierarchical structure are initially provided. In most cases, a workload is not harmonic. Thus, we describe a method of deriving component interfaces for the purpose of minimizing the abstraction overhead using HSSS.

5 Compositional Framework-SYNC for Non-Harmonic Tasks

In this section, we discuss a compositional framework for general periodic tasks, where given tasks are adapted to an HSSS so as to remove abstraction overheads of intermediate components. To this end, we present 1) an adaptation of non-harmonic component into a

harmonic component, and 2) an algorithm that determines interface period such that it is harmonic with the underlying component's workload tasks.

5.1 Proximation of Non-Harmonic Workload Tasks

For a non-harmonic task set, we dedicate a periodic resource server [25] to each task. We call the dedicated periodic server *proxy task* and denote it by $v(\Pi, \Theta)$. A given task can run only when the associated proxy task runs. In our framework, we force the proxy tasks in a component to be harmonic with each other and the component interface as well, so that the interface incurs no SAO and becomes an IPRM. Lemma 24 presents a computation that creates a proxy task $v(\Pi, \Theta)$ for a given task.

► **Lemma 24** (Budget for Proxy Task). *For a given task $\tau(p, e)$ and period Π of $v(\Pi, \Theta)$, the resource supply Θ of the proxy task $v(\Pi, \Theta)$ is computed by*

$$\Theta = \begin{cases} \frac{e}{k} & \text{if } 0 \leq \alpha \leq \Pi - \frac{e}{k} \\ \frac{e + \Pi - \alpha}{k + 1} & \text{if } \Pi - \frac{e}{k} < \alpha \leq \Pi \end{cases} \quad (6)$$

where $p = k\Pi + \alpha$.

(a) Case 1

(b) Case 2

■ **Figure 8** Different cases for the relation between a given task and its proxy task

Proof. Let $\text{dbf}_{RM}(W, t, i) = e_i + \sum_{k \in HP(i)} \lceil t/p_k \rceil e_k$, where $HP(i)$ is a set of tasks whose priority is higher than i . Given a task $\tau = (p, e)$ and an IPRM $H = (\Pi, \Theta)$ such that $\exists t \leq p. \text{dbf}_{RM}(\{\tau\}, t, -) \leq \text{sbfi}_{IPRM}(H, t)$, let $p = k\Pi + \alpha$. Then, α is in two cases: $0 \leq \alpha \leq \Pi - \Theta$ (The case of Figure 8a) and $\Pi - \Theta < \alpha \leq \Pi$ (The case of Figure 8b). For each case, Θ is computed according to the two cases:

Case 1 $0 \leq \alpha \leq \Pi - \Theta$,

■ **Figure 9** An example of HSSS for non-harmonic tasks

Let $k\Pi \leq p \leq k\Pi - \Theta$ such that $0 \leq \alpha \leq \Pi - \Theta$ and $p = k\Pi + \alpha$. Then, the minimum value of $k\Theta$ is equal to e such that $\text{dbf}_{RM}(\{\tau\}, t, -) \leq \text{sbf}_H(t)$ where $k\Pi \leq t \leq k\Pi - \Theta$. Thus, $\Theta = \frac{e}{k}$.

Case 2 $\Pi - \Theta < \alpha \leq \Pi$,

Let $(k+1)\Pi - \Theta < p \leq (k+1)\Pi$ such that $\Pi - \Theta < \alpha \leq \Pi$. Then $k\Theta < e \leq (k+1)\Theta$ such that $\text{dbf}_{RM}(\{\tau\}, t, -) \leq \text{sbf}_H(t)$ where $k\Pi - \Theta < t < k\Pi$. Thus, for α , $e + \Pi - \alpha = (k+1)\Theta$, then $\Theta = \frac{e + \Pi - \alpha}{k+1}$. ◀

► **Example 25.** Suppose that a task $\tau(16, 5)$ is given. If the period Π of $v(\Pi, \Theta)$ is 5, then the task's period p is $16 = 3 \times 5 + 1$ where $\alpha = 1$. $\Theta = 5/3 = 1.667$ such that $0 \leq \alpha = 1 \leq 5 - \Theta$. With a different period $\Pi = 6$, the period p is $16 = 2 \times 6 + 4$ where $\alpha = 4$. Thus, $\Theta = \frac{5+6-4}{2+1} = 7/3 = 2.667$ such that $6 - \Theta < \alpha \leq 6$.

Figure 9 shows the mappings from tasks ($\tau_7 \sim \tau_{12}$) to the corresponding proxy tasks ($v_7 \sim v_{12}$). We call such a creation of proxy tasks *proximation* in this paper. It may also result in a cost, called *proximation overhead*. While the abstraction overhead of a PRM based CF is incurred by every component, the proximation overhead is incurred only by tasks in leaf components. Using numerical analysis and simulation experiments in Section 6, we show that the proximation overheads are much smaller than component abstraction overheads.

5.2 Harmonic Interface Generation

This section presents an algorithm that determines interfaces of components and proxy tasks for a given system. The challenging issues in deriving interfaces are 1) to meet harmonicity of interface periods and 2) to reduce of proximation overhead. The component interface's period should be harmonic with all the workload tasks'. In addition, proxy tasks in leaf components are also to be harmonic. This harmonicity condition results in no abstraction overhead in hierarchical real-time systems.

The proximation overhead depends on the proxy task period which can be any number less than or equal to the task period. We prefer a bigger period of proxy tasks that is close to original period of a given application task due to extra overheads for context switching and interrupt handling of smaller-period proxy tasks [4].

The first step of the algorithms is to decide the base period for harmonic composition of the system. When we derive the interface of a component, the period of the interface should

```

findCompInterface ( $H, C = (W, A)$ )
/*
-  $H$ : a set of base periods
-  $C$ : a component with workload  $W$  and algorithm  $A$  */
*/
1:  $H_C \leftarrow H$ 
2: for  $\tau_i \in W$  do
3:   if  $\text{type}(\tau_i) == \text{component}$  then
4:      $(p_i^H, e_i^H) \leftarrow \text{findCompInterface}(H_C, \text{comp}(\tau_i))$ 
5:   else  $\text{type}(\tau_i) == \text{task}$  */
6:      $(p_i^H, e_i^H) \leftarrow \text{findVirtualTaskInterface}(H_C, \tau_i)$ 
7:      $H_C \leftarrow H_C \cup \{p_i^H\}$ 
8:   endfor
9:  $P_C \leftarrow \min_{\tau_i \in W} p_i^H$ 
10:  $E_C \leftarrow P_C \times \sum_{\tau_i \in W} \frac{e_i^H}{p_i^H}$ 
11: Set the interface  $I_C = (P_C, E_C)$  of the component  $C$ .
12: return  $(P_C, E_C)$ 

```

■ **Figure 10** Algorithm of component interface generation

be harmonic with both its workload tasks and the interfaces of other sibling components. This base period, denoted as h_{base} , can be any number between one and $\lfloor p_{\min}/2 \rfloor$, where $p_{\min}$ is the smallest task period in the system. 2 is selected because any period from one to $\lfloor p_{\min}/2 \rfloor$ does not need more resources than the original period of a task [4] in single task abstraction.

```

findVirtualTaskInterface ( $H, T_i = (p_i, c_i)$ )
1:  $p_i^H \leftarrow 1$ 
2:  $e_i^H \leftarrow 1$ 
3: for  $p$  from  $\lfloor p_i/2 \rfloor + 1$  downto 1 do
4:   if  $p$  is harmonic with  $H$  then
5:      $e \leftarrow \text{calculateBudget}(p, p_i, e_i)$ 
6:     if  $e/p < e_i^H/p_i^H$  then
7:        $p_i^H \leftarrow p$ 
8:        $e_i^H \leftarrow e$ 
9:     endif
10:  endif
11: endfor
12: Set the proxy task of  $\tau_i$  as  $v_i = (p_i^H, e_i^H)$ .
13: return  $(p_i^H, e_i^H)$ 

```

■ **Figure 11** Algorithm of proxy task interface generation

Since the component interface is derived from the workload, the generation procedure is accomplished from leaves to the root in the component hierarchy graph. Figure 10 shows the pseudo-code of deriving a component interface. We use the depth-first-search mechanism to compute the interface, so that the function **findCompInterface** calls itself recursively if the workload task is a component (line 3-4). For example, the interface determination

```

deriveSystemInterface ( $C_0 = (W_0, A_0)$ )
/*  $C_0$ : the top-layer component or the root component in
the system */
1:  $p_{min} \leftarrow$  the minimum period in all tasks in the system
2:  $h_{base} \leftarrow$  any number between 1 and  $\lfloor p_{min}/2 \rfloor$ 
3:  $(P_{C_0}, E_{C_0}) \leftarrow \mathbf{findCompInterface}(\{h_{base}\}, C_0, A_0)$ 

```

■ **Figure 12** Algorithm of system interface generation

sequence of components in Figure 9 is $C_3-C_4-C_1-C_2-C_0$.

The proxy task interface of a task is selected as one with the minimum utilization among proxy tasks harmonic with the given set of periods. As shown in Figure 11, the proxy task period is selected between one and half of the task period (line 3). If the period is harmonic with the given set H of periods, we calculate the required budget for the task by Eq. 6 (line 5). The new proxy task interface is updated so that the minimum-utilization interface is selected (line 4-10). For example, for a given $h_{base} = 9$, possible proxy task periods of task $task_7$ are 1, 3, 9, and 18. Since the proxy period 9 provides with the minimum utilization, the proxy task of τ_7 is determined by $v_1(9, 1)$.

After the interface of a task or component is determined, the next task or component should harmonic with this new interface period. For example, when the proxy task period of v_9 in Figure 9 is determined by 27, the proxy task period of task τ_{10} should be harmonic with 9 and 27. Thus, we add the new interface period in the set of harmonic periods H_C in order for the next task or component interface to consider it (line 7 of Figure 10). When the function **findVirtualTaskInterface** is called for task τ_{10} , for instance, the harmonic period set H is given by $\{9, 27\}$. The possible proxy task periods less than half of the period of τ_{10} are 1, 9, 27, 54, 81, and 108. The algorithm selects one among possible harmonic periods.

When all the tasks or sub-components in the component workload decide their interfaces, the component interface is determined easily. The interface period becomes the minimum among tasks' periods. Since we need the interface period to be harmonic with all the workload tasks, the minimum period satisfies this condition (line 9 of Figure 10). The execution time of the interface is determined by the utilization of the workload due to the harmonicity (line 10 of Figure 10).

As shown in Figure 12, we derive all the interfaces by calling the function **findCompInterface** with the root component and the base harmonic period. The time complexity of Figure 11 requires pseudo-polynomial time because it searches all integer-value periods less than half of the task period. Thus, the proposed algorithm runs in pseudo-polynomial time.

5.3 Utilization Bound Computation in using Proxy Tasks

In this section, we analyze the utilization bound (UB) of a component under the proposed framework. First, we derive the utilization bound of a proxy task which guarantees the schedulability of a given task in a leaf component. Then, we extend it to the utilization bound of a component workload.

► **Lemma 26.** (*Proxy Task Utilization Bound: UB_v*) A task $\tau_i(p_i, e_i)$ is schedulable by a proxy task $v_i(\Pi_i, \Theta_i)$, if

$$U_{v_i} = \frac{\Theta_i}{\Pi_i} \geq UB_v = \frac{(k+1)U_i}{k + U_i}, \quad (7)$$

■ **Figure 13** Schedulable resource utilization bound of a single task

where $p_i = k\Pi_i + \alpha$ ($k \geq 1, 0 \leq \alpha < \Pi_i$).

Proof. Let $\text{ldbf}_{\text{IPRM}}(\mathbf{v}_i, t)$ be $\frac{k\Theta_i}{k\Pi_i + (\Pi_i - \Theta_i)}t$, which is the linear function passing through the point $(k\Pi_i + (\Pi_i - \Theta_i), k\Theta_i)$ as shown in Figure 13. Then, for $k\Pi_i \leq t \leq (k+1)\Pi_i$, $\text{ldbf}_{\text{IPRM}}(\{\tau_i\}, t)$ is less than or equal to $\text{sbf}_{\text{IPRM}}(\mathbf{v}_i, t)$.

Suppose that $p_i = k\Pi_i + \alpha$ and $\frac{(k+1)U_i}{k+U_i} \leq U_{\mathbf{v}_i}$. Then,

$$\begin{aligned} (k+1)U_i &\leq (k+U_i)U_{\mathbf{v}_i} \\ U_i &\leq \frac{kU_{\mathbf{v}_i}}{k+1-U_{\mathbf{v}_i}} \\ &= \frac{k\Theta_i}{\Pi_i(k+1-\frac{\Theta_i}{\Pi_i})} = \frac{k\Theta_i}{k\Pi_i + \Pi_i - \Theta_i} \\ e_i &\leq \frac{k\Theta_i}{k\Pi_i + \Pi_i - \Theta_i} p_i \\ \text{dbf}_A(\{\tau_i\}, p_i) = e_i &\leq \text{ldbf}_{\text{IPRM}}(\{\tau_i\}, p_i) \leq \text{sbf}_{\text{IPRM}}(\mathbf{v}_i, p_i) \end{aligned}$$

Since $\text{dbf}_A(\{\tau_i\}, p_i) = e_i$ for any scheduling algorithm A , the demand of the task at time p_i is less than or equal to the resource supply by the proxy task, which guarantees the schedulability of the task. ◀

Now, we extend the proxy task utilization bound in Lemma 26 to a workload as shown in the following theorem.

► **Theorem 27.** (*Utilization Bound of a Workload: UB_W*) A workload W is schedulable by $\text{IPRM } \mathcal{H}(\Pi, \Theta)$, if

$$U_\Gamma = \frac{\Theta}{\Pi} \geq \text{UB}_W = \sum_{i=1}^n \frac{(k_i+1)U_i}{k_i+U_i}, \quad (8)$$

where $p_i = k_i\Pi + \alpha_i$ ($k_i \geq 1, 0 \leq \alpha_i < \Pi$).

Proof. In the proposed framework, we assign a proxy task $\mathbf{v}_i(\Pi, \Theta_i)$ for each task $\tau_i \in W$. Since all proxy tasks have the same period as the interface period, the component becomes harmonic so that there is no composition overhead among Γ and $\mathbf{v}_i$'s. Now, let $\Theta = \Theta_1 + \Theta_2 + \dots + \Theta_n$, where each $\Theta_i \geq \Pi \frac{(k_i+1)U_i}{k_i+U_i}$. Then, such a proxy resource distribution satisfies Eq. 8. Suppose that the component workload with such a proxy resource allocation is not schedulable. However, each task is schedulable by the proxy task (Π, Θ_i) according to Lemma 26 because $U_{\mathbf{v}_i} \geq \frac{(k_i+1)U_i}{k_i+U_i}$. Therefore, this contradicts the assumption that the workload is not schedulable. ◀

■ **Figure 14** Proxy task utilization bounds

■ **Figure 15** Utilization bound difference of PRM and HPRM

6 Performance Evaluation

6.1 Utilization Bound of Proximity

In this subsection, we numerically analyze the utilization bound of a single task proximation. Figure 14 shows the schedulable utilization bound of the proxy task for a given task according to Eq. 7 of Lemma 26 with regards to k . As shown in Figure 14, the utilization bounds become closer to given utilizations of task as k becomes larger. This is because the smaller resource period reduces the compositional overhead.

We compare the utilization bound overhead of a single task proximation with that of PRM [24]. In [24], they provided the utilization bound of n tasks for a given resource. We use the utilization bound of PRM with $n = 1$ since the task proximation is the same as a component with one task. Figure 15 shows the utilization bound difference of the proposed scheme and PRM. These improvements result from the supply bound function with no abstraction overhead in the proposed framework.

In this section, we compare the proposed scheme with other related schemes for hierarchical real-time systems. Specifically, we consider the following schemes:

1. PRM (*PRM*) [24]: CF based on PRM.
2. PRM with a single global period (*PRM-single*) [4]: Every interface is given a single period, which is less than or equal to p_{min} .
3. A combination technique - Harmonic component by using a shrinking period (*PRM-sync-Trans*): Components are harmonic and task periods are shrunken to be harmonic with interface periods.
4. Our technique (*PRM-sync-Proxy*).

Figure 16 Resource bounds of a component with i.i.t.u.

Figure 17 Average resource bounds of a component

6.2 Composition Overhead of Independent Components

In this comparison, we want to show the efficiency of the proposed proximation based composition approach. The proposed framework of this paper does not produce any abstraction overheads in intermediate components, but the proximation overhead incurred by leaf components remain. For this reason, we show the efficiency of proximation by comparing the resource utilization bound of our approach against those of a PRM based CF for a single component.

Since the resource bound in Eq. 8 depends on individual task utilizations, we analyze it in two ways: First, we analyze it under the assumption of *individually identical task utilization* (*i.i.t.u.*). Second, we measure the average resource bound for randomly generated task sets.

If we assume that each task has the same utilization, Eq. 8 becomes $(k+1)U_W/(k+U_W/n)$, where $k = \min_{\tau_i \in W} k_i$ and n is the number of tasks in W . We can then compare the resource bound of the proximation based CF against that of PRM under RM and EDF based on the equations in [24]. Figure 16 shows the resource bounds of three schemes when $n = 4$. Note that the results for other n show similar tendency.

The proposed scheme generally shows lower resource bound in lower utilization. In case

of $k = 1$, the proposed scheme under i.i.t.u. is worse than PRM-RM (PRM under RM). When $k \geq 2$, it performs better than PRM-RM. As k increases, the resource bounds of the proposed scheme and PRM-EDF algorithm become close.

Next, we analyze the resource bounds for randomly generated tasks. We generate n tasks for each workload utilization range, where n is randomly selected from 2 to 6. The individual task utilization is generated using a normal distribution with mean U_W/n and standard deviation 50% of the utilization range (i.e. 0.05). We also randomly select k from 1 to 10. We measure the average resource bounds of 500 random task sets for each utilization range and show the results in Figure 17. The resource bound overhead of Figure 17(b) is obtained by $(UB_W - U_W)/U_W$ for a given resource bound UB_W .

The resource bound of the proposed scheme performs better than PRM when the workload utilization is less than about 70%. However, for higher workload utilizations, the proposed scheme has higher resource bounds than PRM-EDF. This is because the resource bound of the proposed scheme depends on individual task proximation overheads, as shown in Figure 17(b).

6.3 Composition Overhead

In simulation experiments, we use a system structure with a depth of three as in Figure 18. Each leaf component has three tasks, resulting in a system that contains eighteen tasks in all. The task period is randomly generated between p_{min} and p_{max} for each component as shown in Table 2.

■ **Table 2** Period ranges of tasks in the simulations

Component	C_4	C_5	C_6	C_7	C_8	C_3
$[p_{min}, p_{max}]$	[50, 110]	[60, 120]	[70, 130]	[80, 140]	[90, 150]	[100, 160]

For a given system utilization range of $[U_{min}, U_{max}]$, we generate each task period and execution time such that the total utilization resides in a utilization range. As shown in Figure 19, we varied the utilization range from 0.2 to 0.6 in an interval of 0.05. We define the utilization overhead as the difference between the top-layer workload utilization by composition and the actual utilization of eighteen tasks. For each utilization range, we generate 100 task sets and measure the average overhead.

■ **Figure 18** A system structure for simulations

Figure 19(a) shows the average utilization overhead for each utilization range of task sets, while Figure 19(b) shows the overheads in percentile compared to the task set utilization. As shown in Figure 19, the proposed scheme shows the lowest overhead of about 5 %. Although both *PRM-sync-Trans* and *PRM-single* have no intermediate overhead, the difference comes from the additional reduction due to the harmonicity or divisibility between leaf components and their proxy tasks. Since *PRM-sync-Trans* incurs more overhead than *PRM-sync-Proxy*,

Figure 19 Utilization Overhead

the task proximation seems to be more efficient than just reducing task period for harmonicity or divisibility.

Figure 20 Component Overhead in PRM

Figure 20 shows the utilization overhead in each component of *PRM* and *PRM-sync-Proxy*. Since *PRM* does not consider intermediate component overheads, it generates additional composition overhead in C_1 and C_2 as shown in Figure 20. The proposed framework removes all intermediate composition overheads. Let us note that *PRM-single* is also a special case of the proposed framework because all components use a common period, resulting in harmonicity. Another improvement comes from the reduction of overhead in leaf components with RM scheduling algorithm. For example, C_4 , C_6 , and C_8 in Figure 20 show high overheads because of RM algorithm. However, the proposed framework only incurs proximation overheads regardless of scheduling algorithm since the proxy task enforces tasks under RM to be harmonic.

6.4 Runtime Overhead of HSS

The smaller the interface period, the more is the runtime overhead such as context switch and scheduling[20]. For instance, the common period of *PRM-single* [4] needs to be smaller or equal to p_{min} of the system, implying that it may induce frequent context switches. The proxy task of our framework also imposes such a runtime overhead. So this section compares the runtime overhead of the proposed technique against prior techniques.

For a given component or proxy task with period P , it may incur one context switch every P time units. Thus, in this paper, we approximately measure the number of context

switches of each component or proxy task by counting one per its period. The approximate number of context switches in a system is defined by the summation of those numbers of all components and proxy tasks.

Figure 21 Average number of approximate context-switches

Figure 21 shows the average number of context switches of four schemes in the simulation results of Figure 19. *PRM-single* has many context switches due to the small interface period. On the contrary, *PRM-sync-Trans* shows the least context switch overhead. Since *PRM-sync-Proxy* has an additional layer due to task proximation, it shows relatively more context switch overhead than *PRM-sync-Trans*, but still significantly lower than *PRM-single*.

In reality, it is not big efficiency degradation since, as shown in [12], most actual tasks in real applications can be divided into several task groups of which each uses a common period.

7 Discussion and Conclusions

OS-Hypervisor engineers can better utilize resources using our technique as follows:

- OS-Hypervisor can better utilize the resources by using harmonic interfaces.
- If the host system manages the virtual system to be harmonic, regardless of host's scheduling algorithm, it can guarantee Θ/Π bandwidth of resource supply for the virtual system's resource request of (Π, Θ) .
- If the host system running EDF manages the virtual system to be divisible, it can guarantee Θ/Π bandwidth of resource supply for the virtual system's resource request of (Π, Θ) .

This paper discussed 2 abstraction overheads of prior compositional framework and presented sufficient conditions for scheduling components to remove these overheads. In addition, we provided novel techniques and algorithms that allow tasks to be composed in an HSS with low overheads. We showed that our technique outperforms a prior overhead-reducing CF [4] by about 10% on average and other combination techniques by about 8% on average.

Our proximation technique is not optimal since the base period of harmonic periods is randomly selected such that it is less than the minimal period among all workload. More study needs to be done to develop an optimal selection technique for harmonic periods. In the future work, we plan to devise an optimal algorithm to compute harmonic periods of components and investigate how to reduce the runtime overheads when applying the techniques in practical systems.

References

- 1 Madhukar Anand. *Conditional models for compositional design of real-time embedded systems*. PhD thesis, University of Pennsylvania, 2008.
- 2 A. Biondi, G. C. Buttazzo, and M. Bertogna. Schedulability analysis of hierarchical real-time systems under shared resources. *IEEE Transactions on Computers*, 65(5):1593–1605, May 2016. doi:10.1109/TC.2015.2444833.
- 3 Yang Chang, Robert Davis, and Andy Wellings. Schedulability analysis for a real-time multiprocessor system based on service contracts and resource partitioning. *University of York, Tech. Rep. YCS*, 432, 2008.
- 4 Sanjian Chen, Linh TX Phan, Jaewoo Lee, Insup Lee, and Oleg Sokolsky. Removing abstraction overhead in the composition of hierarchical real-time systems. In *RTAS, 2011 17th IEEE*, pages 81–90. IEEE, 2011.
- 5 W. J. Chen, P. C. Huang, Q. Leng, A. K. Mok, and S. Han. Regular composite resource partition in open systems. In *2017 IEEE Real-Time Systems Symposium (RTSS)*, pages 34–44, Dec 2017. doi:10.1109/RTSS.2017.00011.
- 6 A. Easwaran, I. Lee, O. Sokolsky, and S. Vestal. A compositional scheduling framework for digital avionics systems. In *RTCSA*, pages 371–380, Aug 2009. doi:10.1109/RTCSA.2009.46.
- 7 Arvind Easwaran, Madhukar Anand, and Insup Lee. Compositional analysis framework using edp resource models. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, pages 129–138, 2007.
- 8 Arvind Easwaran, Madhukar Anand, Insup Lee, and Oleg Sokolsky. On the complexity of generating optimal interfaces for hierarchical systems. In *CRTS*, 2008.
- 9 Nathan Fisher and Sanjoy Baruah. Approximation algorithms for feasibility analysis in real-time static-priority systems. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, pages 117–126, 2005.
- 10 Green Hills Software. Safety Critical Products: ARINC 653 Partition Scheduler . https://www.ghs.com/products/safety_critical/arinc653.html/, 2017. [Online; accessed 27-Sept-2017].
- 11 Chunhui Guo, Xiayu Hua, Hao Wu, Douglas Lautner, and Shangping Ren. Best-harmonically-fit periodic task assignment algorithm on multiple periodic resources. *IEEE Transactions on Parallel and Distributed Systems*, 27(5):1303–1315, 2016.
- 12 Simon Kramer, Dirk Ziegenbein, and Arne Hamann. Real world automotive benchmark for free. In *WATER 2015*.
- 13 T. W. Kuo and A. K. Mok. Load adjustment in adaptive real-time systems. In *[1991] Proceedings Twelfth Real-Time Systems Symposium*, pages 160–170, Dec 1991. doi:10.1109/REAL.1991.160369.
- 14 Tei-Wei Kuo and Ching-Hui Li. A fixed-priority-driven open environment for real-time applications. In *RTSS*, pages 256–267. IEEE, 1999.
- 15 J. Lee, S. Xi, S. Chen, L. T. X. Phan, C. Gill, I. Lee, C. Lu, and O. Sokolsky. Realizing compositional scheduling through virtualization. In *RTAS*, pages 13–22, April 2012. doi:10.1109/RTAS.2012.20.
- 16 C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973. URL: <http://doi.acm.org/10.1145/321738.321743>, doi:10.1145/321738.321743.
- 17 Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.
- 18 Aloysius K. Mok, Xiang (Alex) Feng, and Deji Chen. Resource partition for real-time systems. In *Proceedings of RTAS '01*, pages 75–84. IEEE Computer Society, 2001.

- 19 Malcolm S Mollison, Jeremy P Erickson, James H Anderson, Sanjoy K Baruah, and John A Scoredos. Mixed-criticality real-time scheduling for multicore systems. In *Computer and Information Technology (CIT)*, pages 1864–1871. IEEE, 2010.
- 20 Linh TX Phan, Meng Xu, Jaewoo Lee, Insup Lee, and Oleg Sokolsky. Overhead-aware compositional analysis of real-time systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*, pages 237–246. IEEE, 2013.
- 21 Insik Shin, Arvind Easwaran, and Insup Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *Real-Time Systems, 2008. ECRTS'08. Euromicro Conference on*, pages 181–190. IEEE, 2008.
- 22 Insik Shin, Arvind Easwaran, and Insup Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *ECRTS*, pages 181–190. IEEE Computer Society, 2008.
- 23 Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *RTSS*, pages 2–13, 2003.
- 24 Insik Shin and Insup Lee. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embedded Comput. Syst.*, 7(3), 2008.
- 25 Brinkley Sprunt, Lui Sha, and John Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, 1(1):27–60, 1989.