SIRAP: A Synchronization Protocol for Hierarchical Resource Sharing in Real-Time Open Systems[*]

Moris Behnam, Insik Shin[†], Thomas Nolte, Mikael Nolin
Mälardalen Real-Time Research Centre (MRTC)
Västerås, Sweden
{moris.behnam, insik.shin, thomas.nolte, mikael.nolin}@mdh.se

ABSTRACT

This paper presents a protocol for resource sharing in a hierarchical real-time scheduling framework. Targeting real-time open systems, the protocol and the scheduling framework significantly reduce the efforts and errors associated with integrating multiple semi-independent subsystems on a single processor. Thus, our proposed techniques facilitate modern software development processes, where subsystems are developed by independent teams (or subcontractors) and at a later stage integrated into a single product. Using our solution, a subsystem need not know, and is not dependent on, the timing behaviour of other subsystems; even though they share mutually exclusive resources. In this paper we also prove the correctness of our approach and evaluate its efficiency.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-Purpose and Application based System—*Real-time and embedded systems*; D.4.1 [**Operating System**]: Process Management—*Concurrency, Mutual exclusion, Scheduling, Synchronization.*

General Terms

Design, Performance, Theory.

Keywords

Hierarchical scheduling, Real-time open systems, Real-time subsystem integration, Resource-sharing, SIRAP, Synchronization protocol.

[*]The work in this paper is supported by the Swedish Foundation for Strategic Research (SSF), via the research programme PROGRESS.
[†]Corresponding author.

1. INTRODUCTION

In many industrial sectors integration of electronic and software subsystems (to form an integrated hardware and software system), is one of the activities that is most difficult, time consuming, and error prone [2, 14]. Almost any system, with some level of complexity, is today developed as a set of semi-independent subsystems. For example, cars consist of multiple subsystems such as antilock braking systems, airbag systems and engine control systems. In the later development stages, these subsystems are integrated to produce the final product. Product domains where this approach is the norm include automotive, aerospace, automation and consumer electronics.

It is not uncommon that these subsystems are more or less dependent on each other, introducing complications when subsystems are to be integrated. This is especially apparent when integrating multiple software subsystems on a single processor. Due to these difficulties inherent in the integration process, many projects run over their estimated budget and deadlines during the integration phase. Here, a large source of problems when integrating real-time systems stems from subsystem interference in the time domain.

To provide remedy to these problems we propose the usage of a real-time scheduling framework that allows for an easier integration process. The framework will preserve the essential temporal properties of the subsystem both when the subsystem is executed in isolation (unit testing) and when it is integrated together with other subsystems (integration testing and deployment). Most importantly, the deviation in the temporal behaviour will be bounded, hence allowing for predictable integration of hard real-time subsystems. This is traditionally targeted by the philosophy of open systems [9], allowing for the independent development and validation of subsystems, preserving validated properties also after integration on a common platform.

In this paper we present the Subsystem Integration and Resource Allocation Policy (SIRAP), which makes it possible to develop subsystems individually without knowledge of the temporal behaviour of other subsystems. One key issue addressed by SIRAP is the resource sharing between subsystems that are only semi-independent, i.e., they use one or more shared logical resources.

Problem description.

A software system $\mathcal{S}$ consists of one or more subsystems to be executed on one single processor. Each subsystem $S_s \in \mathcal{S}$, in turn, consists of a number of tasks. These subsystems can be developed independently and they have their own local scheduler (scheduling the subsystem's tasks). This approach by isolation of tasks within subsystems, and allowing for their own local scheduler, has several advantages [19]. For example, by keeping a subsystem isolated from other subsystems, and by keeping the subsystem local scheduler, it is possible to re-use a complete subsystem in a different application from where it was originally developed. However, as subsystems are likely to share logical resources, an appropriate resource sharing protocol must be used. In order to facilitate independent subsystem development, this protocol should not require information from all other subsystems in the system. It should be enough with only the information of the subsystem under development in isolation.

Contributions.

The main contributions of this paper include the presentation of SIRAP, a novel approach to subsystem integration in the presence of shared resources. Moreover, the paper presents the deduction of bounds on the timing behaviour of SIRAP together with accompanying formal proofs. In addition, the cost of using this protocol is thoroughly evaluated. The cost is investigated as a function of various parameters including: cost as a function of the length of critical sections, cost depending on the priority of the task sharing a resource, and cost depending on the periodicity of the subsystem. Finally, the cost of having an independent subsystem abstraction, which is suitable for open systems, is investigated and compared with dependent abstractions.

Organization of the paper.

Firstly, related work on hierarchical scheduling and resource sharing is presented in Section 2. Then, the system model is presented in Section 3. SIRAP is presented in Section 4. In Section 5 schedulability analysis is presented, and SIRAP is evaluated in Section 6. Finally, the paper is summarized in Section 7.

2. RELATED WORK

Hierarchical scheduling.

For real-time systems, there has been a growing attention to hierarchical scheduling frameworks [1, 7, 9, 10, 15, 16, 17, 20, 23, 25, 26].

Deng and Liu [9] proposed a two-level hierarchical scheduling framework for open systems, where subsystems may be developed and validated independently in different environments. Kuo and Li [15] presented schedulability analysis techniques for such a two-level framework with the fixed-priority global scheduler. Lipari and Baruah [16, 18] presented schedulability analysis techniques for the EDF-based global schedulers.

Mok *et al.* [21] proposed the bounded-delay resource partition model for a hierarchical scheduling framework. Their model can specify the real-time guarantees that a parent component provides to its child components, where the parent and child components have different schedulers. Feng and Mok [10] and Shin and Lee [26] presented schedulability analysis techniques for the hierarchical scheduling framework that employs the bounded-delay resource partition model.

There have been studies on the schedulability analysis with the periodic resource model. This periodic resource model can specify the periodic resource allocation guarantees provided to a component from its parent component [25]. Saewong *et al.* [23] and Lipari and Bini [17] introduced schedulability conditions for fixed-priority local scheduling, and Shin and Lee [25] presented a schedulability condition for EDF local scheduling. Davis and Burns [7] evaluated different periodic servers (Polling, Deferrable, and Sporadic Servers) for fixed-priority local scheduling.

Resource sharing.

When several tasks are sharing a logical resource, typically only one task is allowed to use the resource at a time. Thus the logical resource requires mutual exclusion of tasks that uses it. To achieve this a *mutual exclusion protocol* is used. The protocol provides rules about how to gain access to the resource, and specifies which tasks should be blocked when trying to access the resource.

To achieve predictable real-time behaviour, several protocols have been proposed including the Priority Inheritance Protocol (PIP) [24], the Priority Ceiling Protocol (PCP) [22], and the Stack Resource Policy (SRP) [3].

When using SRP, a task may not preempt any other tasks until its priority is the highest among all tasks that are ready to run, and its preemption level is higher than the system ceiling. The preemption level of a task is a static parameter assigned to the task at its creation, and associated with all instances of that task. A task can only preempt another task if its preemption level is higher than the task that it is to preempt. Each resource in the system is associated with a resource ceiling and based on these resource ceilings, a system ceiling can be calculated. The system ceiling is a dynamic parameter that changes during system execution.

The duration of time that a task lock a resource, is called Resource Holding Time (RHT). Fisher *et al.* [4, 11] proposed algorithms to minimize RHT for fixed priority and EDF scheduling with SRP as a resource synchronization protocol. The basic idea of their proposed algorithms is to increase the ceiling of resources as much as possible without violating the schedulability of the system under the same semantics of SRP.

Deng and Liu [9] proposed the usage of non-preemptive global resource access, which bounds the maximum blocking time that a task might be subject to. The work by Kuo and Li [15] used SRP and they showed that it is very suitable for sharing of local resources in a hierarchical scheduling framework. Almeida and Pedreiras [1] considered the issue of supporting mutually exclusive resource sharing within a subsystem. Matic and Henzinger [20] considered supporting interacting tasks with data dependency within a subsystem and between subsystems, respectively.

More recently, Davis and Burns [8] presented the Hierarchical Stack Resource Policy (HSRP), allowing their work on hierarchical scheduling [7] to be extended with sharing of logical resources. However, using HSRP, information on all tasks in the system must be available at the time of subsystem integration, which is not suitable for an open systems development environment, and this can be avoided by the SIRAP protocol presented in this paper.

3. SYSTEM MODEL

3.1 Hierarchical scheduling framework

A hierarchical scheduling framework is introduced to support CPU time sharing among applications (subsystems) under different scheduling services. Hence, a system $\mathcal{S}$ consists of one or more subsystems $S_s \in \mathcal{S}$. The hierarchical scheduling framework can be generally represented as a two-level tree of nodes, where each node represents a subsystem with its own scheduler for scheduling internal tasks (threads), and CPU time is allocated from a parent node to its children nodes, as illustrated in Figure 1.

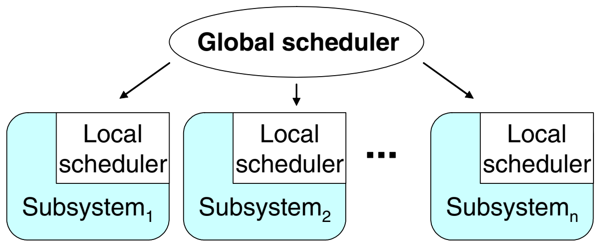

Figure 1: Two-level hierarchical scheduling framework.

The hierarchical scheduling framework provides *partitioning* of the CPU between different subsystems. Thus, subsystems can be isolated from each other for, e.g., fault containment, compositional verification, validation and certification and unit testing.

The hierarchical scheduling framework is also useful in the domain of open systems [9], where subsystems may be developed and validated independently in different environments. For example, the hierarchical scheduling framework allows a subsystem to be developed with its own scheduling algorithm internal to the subsystem and then later included in a system that has a different global level scheduler for scheduling subsystems.

3.2 Shared resources

For the purpose of this paper a shared (logical) resource, r_i, is a shared memory area to which only one task at a time may have access. To access the resource a task must first lock the resource, and when the task no longer needs the resource it is unlocked. The time during which a task holds a lock is called a *critical section*. Only one task at a time may lock each resource.

A resource that is used by tasks in more than one subsystem is denoted a *global shared resource*. A resource only used within a single subsystem is a *local shared resource*. In this paper we are concerned only with global shared resources and will simply denote them by shared resources. Management of local shared resources can be done by using any synchronization protocol such as PIP, PCP, and SRP.

3.3 Virtual processor model

The notion of real-time virtual processor (resource) model was first introduced Mok *et al.* [21] to characterize the CPU allocations that a parent node provides to a child node in a hierarchical scheduling framework. The *CPU supply* of a virtual processor model refers to the amounts of CPU allocations that the virtual processor model can provide. The *supply bound function* of a virtual processor model calculates the minimum possible CPU supply of the virtual processor model for a time interval length t.

Shin and Lee [25] proposed the periodic virtual processor model $\Gamma(\Pi, \Theta)$, where Π is a period ($\Pi > 0$) and Θ is a periodic allocation time ($0 < \Theta \leq \Pi$). The capacity U_Γ of a periodic virtual processor model $\Gamma(\Pi, \Theta)$ is defined as Θ/Π. The periodic virtual processor model $\Gamma(\Pi, \Theta)$ is defined to characterize the following property:

$$\text{supply}_\Gamma\big(k\Pi, (k+1)\Pi\big) = \Theta, \quad \text{where } k = 0, 1, 2, \ldots, \quad (1)$$

where the supply function $\text{supply}_{R_s}(t_1, t_2)$ computes the amount of CPU allocations that the virtual processor model R_s provides during the interval $[t_1, t_2)$.

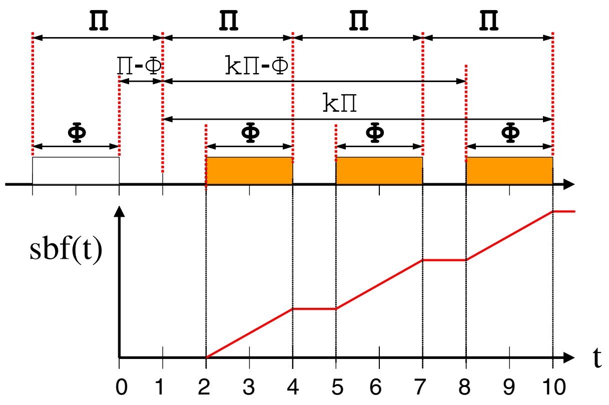

Figure 2: The supply bound function of a periodic virtual processor model $\Gamma(\Pi, \Theta)$ for $k = 3$.

For the periodic model $\Gamma(\Pi, \Theta)$, its supply bound function $\texttt{sbf}_\Gamma(t)$ is defined to compute the minimum possible CPU supply for every interval length t as follows:

$$\texttt{sbf}_\Gamma(t) = \begin{cases} t - (k+1)(\Pi - \Theta) & \text{if } t \in [(k+1)\Pi - 2\Theta, \\ & \quad (k+1)\Pi - \Theta], \\ (k-1)\Theta & \text{otherwise,} \end{cases} \quad (2)$$

where $k = \max\Big(\big\lceil\big(t - (\Pi - \Theta)\big)/\Pi\big\rceil, 1\Big)$. Here, we first note that an interval of length t may not begin synchronously with the beginning of period Π. That is, as shown in Figure 2, the interval of length t can start in the middle of the period of a periodic model $\Gamma(\Pi, \Theta)$. We also note that the intuition of k in Eq. (2) basically indicates how many periods of a periodic model can overlap the interval of length t, more precisely speaking, the interval of length $t - (\Pi - \Theta)$. Figure 2 illustrates the intuition of k and how the supply bound function $\texttt{sbf}_\Gamma(t)$ is defined for $k = 3$.

3.4 Subsystem model

A subsystem $S_s \in \mathcal{S}$, where $\mathcal{S}$ is the whole system of subsystems, consists of a task set and a scheduler. Each subsystem S_s is associated with a periodic virtual processor model abstraction $\Gamma_s(\Pi_s, \Theta_s)$, where Π_s and Θ_s are the subsystem period and budget respectively. This abstraction $\Gamma_s(\Pi_s, \Theta_s)$ is supposed to specify the collective temporal requirements of a subsystem, in the presence of global logical resource sharing.

Task model.

We consider a periodic task model $\tau_i(T_i, C_i, \mathcal{X}_i)$, where T_i and C_i represent the task's period and worst-case execution time (WCET) respectively, and $\mathcal{X}_i$ is the set of WCETs within critical sections belonging to τ_i. Each element $x_{i,j}$ in $\mathcal{X}_i$ represents the WCET of a particular critical section $cx_{i,j}$ executed by τ_i. Note that C_i includes all $x_{i,j} \in \mathcal{X}_i$.

The set of critical sections cover for the following two cases of multiple critical sections within one job:

1. sequential critical sections, where $\mathcal{X}_i$ contains the WCETs of all sequential critical sections, i.e. $\mathcal{X}_i = \{x_{i,1}, ..., x_{i,o}\}$ where o is the number of sequential shared resources that task τ_i may lock during its execution.
2. nested critical sections, where $x_{i,j} \in \mathcal{X}$ being the length of the outer critical section.

Note that in the remaining paper, we use x_i rather than $x_{i,j}$ for simplicity when it is not necessary to indicate j.

Scheduler.

In this paper, we assume that each subsystem has a fixed-priority preemptive scheduler for scheduling its internal tasks.

4. SIRAP PROTOCOL

4.1 Terminology

Before describing the SIRAP protocol, we define the terminology (also depicted in Figure 3) that are related to hierarchical logical resource sharing.

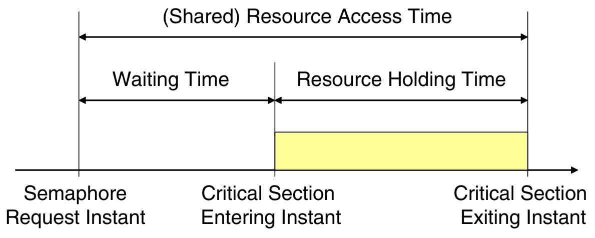
(Shared) Resource Access Time
Waiting Time
Resource Holding Time
Semaphore Request Instant
Critical Section Entering Instant
Critical Section Exiting Instant

Figure 3: Shared resource access time.

- *Semaphore request instant:* an instant at which a job tries to enter a critical section guarded by a semaphore.
- *Critical section entering (exiting) instant:* an instant at which a job enters (exits) a critical section.
- *Waiting time:* a duration from a semaphore request time to a critical section entering time.
- *Resource holding time:* a duration from a critical section entering instant to a critical section exiting instant. Let $h_{i,j}$ denote the resource holding time of a critical section $cx_{i,j}$ of task τ_i.
- *(Shared) resource access time:* a duration from a semaphore request instant to a critical section exiting time.

In addition, a context switch is referred to as *task-level context switch* if it happens between tasks within a subsystem, or as *subsystem-level context switch* if it happens between subsystems.

4.2 SIRAP protocol description

The subject of this paper is to develop a synchronization protocol that can address global resource sharing in hierarchical real-time scheduling frameworks, while aiming at supporting independent subsystem development and validation. This section describes our proposed synchronization protocol, SIRAP (Subsystem Integration and Resource Allocation Policy).

Assumption.

SIRAP relies on the following assumption:

- The system's global scheduler schedules subsystems according to their periodic virtual processor abstractions $\Gamma_s(\Pi_s, \Theta_s)$. The subsystem budget is consumed every time when an internal task within a subsystem executes, and the budget is replenished to Θ_s every subsystem period Π_s. Similar to traditional server-based scheduling methods [6], the system provides a runtime mechanism such that each subsystem is able to figure out at any time t how much its remaining subsystem budget Θ_s is, which will be denoted as $\Theta'_s(t)$ in the remaining of this section.

The above assumption is necessary to allow run-time checking whether or not a job can potentially enter and execute a whole critical section before a subsystem-budget expire. This is useful particularly for supporting independent abstraction of subsystem's temporal behavior in the presence of global resource accesses.

In addition to supporting independent subsystem development, SIRAP also aims at minimizing the resource holding time and bounding the waiting time at the same time. To achieve this goal, the protocol has two key rules as follows:

R1 When a job enters a critical section, preemptions from other jobs within the same subsystem should be bounded to keep its resource holding time as small as possible.

R2 When a job wants to enter a critical section, it enters the critical section at the earliest instant such that it can complete the critical section before the subsystem-budget expires.

The first rule R1 aims at minimizing a resource holding time so that the waiting time of other jobs, which want to lock the same resource, can be minimized as well. The second rule R2 prevents a job J_i from entering a critical section $cx_{i,j}$ at any time t when $\Theta'(t) < h_{i,j}$. This rule guarantees that when the budget of a subsystem expires, no task within the subsystem locks a global shared resource.

SIRAP : preemption management.

The SRP [3] is used to enforce the first rule R1. Each subsystem will have its own system ceiling and resources ceiling according to its jobs that share global resources. According to SRP, whenever a job locks a resource, other jobs within the same subsystem can preempt it if the jobs have higher preemption levels than the locked resource ceiling, so as to bound the blocking time of higher-priority jobs. However, such task-level preemptions generally increase resource holding times and can potentially increase subsystem utilization. One approach to minimize $h_{i,j}$ is to allow no task-level preemptions, by assigning the ceiling of global resource equal to the maximum preemption level. However, increasing the resource ceiling to the maximum preemption level may affect the schedulability of a subsystem. A good approach is presented in [4], which increases the ceiling of shared global resources as much as possible while keeping the schedulability of the subsystem.

SIRAP : self-blocking.

When a job J_i tries to enter a critical section, SIRAP requires each local scheduler to perform the following action. Let t_0 denote the semaphore request instant of J_i and $\Theta'(t_0)$ denote the subsystem's budget at time t_0.

- If $h_{i,j} \leq \Theta'(t_0)$, the local scheduler executes the job J_i. The job J_i enters a critical section at time t_0.
- Otherwise, i.e., if $h_{i,j} > \Theta'(t_0)$, the local scheduler delays the critical section entering of the job J_i until the next subsystem budget replenishment. This is defined as *self-blocking*. Note that the system ceiling will be equal to resource ceiling at time t_0, which means that the jobs that have preemption level greater than system ceiling can only execute during the self blocking interval[1]. This guarantees that when the subsystem of J_i receives the next resource allocation, the subsystem-budget will be enough to execute job J_i inside the critical section[2].

5. SCHEDULABILITY ANALYSIS

5.1 Local schedulability analysis

Consider a subsystem S_s that consists of a periodic task set and a fixed-priority scheduler and receives CPU allocations from a virtual processor model $\Gamma_s(\Pi_s, \Theta_s)$. According to [25], this subsystem is schedulable if

$$\forall \tau_i, 0 < \exists t \leq T_i \quad \texttt{dbf}_{\texttt{FP}}(i,t) \leq \texttt{sbf}_{\Gamma}(t). \tag{3}$$

The goal of this section is to develop the demand bound function $\texttt{dbf}_{\texttt{FP}}(i,t)$ calculation for the SIRAP protocol. $\texttt{dbf}_{\texttt{FP}}(i,t)$ is computed as follows;

$$\texttt{dbf}_{\texttt{FP}}(i,t) = C_i + I_S(i) + I_H(i,t) + I_L(i), \tag{4}$$

where C_i is the WCET of τ_i, $I_S(i)$ is the maximum self blocking for τ_i, $I_H(i,t)$ is the maximum possible interference imposed by a set of higher-priority tasks to a task τ_i during an interval of length t, and $I_L(i)$ is the maximum possible interference imposed by a set of lower-priority tasks that share resources with preemption level (ceiling) greater than or equal to the priority of task τ_i.

[1]With simple modifications to the SRP protocol, the execution of tasks can be allowed within the self blocking interval if they do not access global resources even though their preemption levels are less than the system ceiling. However this is off the point of this paper.

[2]The idea of self-blocking has been also considered in different contexts, for example, in CBS-R [6] and zone based protocol (ZB) [12]. Our work is different from those in the sense that CBS-R used a similar idea for supporting soft real-time tasks, and ZB used it in a pfair-scheduling environment, while we use it for hard real-time tasks under hierarchical scheduling. This difference inherently requires the development of different schedulability analysis, including Eqs. (5), (6), and (7).

The following lemmas shows how to compute $I_S(i)$, $I_H(i,t)$ and $I_L(i)$.

LEMMA 1. *Self-blocking imposes to a job J_i an extra processor demand of at most $\sum_{j=1}^{o} h_{i,j}$ if a job access multiple shared resources.*

PROOF. When the job J_i self-blocks itself, it consumes the processor of at most $h_{i,j}$ units being idle. If the job access shared resources then the worst case will happen when the job block itself whenever it tries to enter a critical section. □

LEMMA 2. *A job J_i can be interfered by a higher-priority job J_j that access shared resources, at t time units for a duration of at most $\lceil \frac{t}{T_j} \rceil (C_j + \sum_{k=1}^{o} h_{j,k})$ time units.*

PROOF. Similar to classical response time analysis [13], we add $\sum_{k=1}^{o} h_{j,k}$ to C_j which is the worst case self blocking from higher priority tasks, the lemma follows. □

LEMMA 3. *A job J_i can be interfered by only one lower-priority job J_j by at most $2 \cdot max(h_{j,k})$, where k=1,...,o.*

PROOF. A higher-priority job J_i can be interfered by a lower-priority job J_j. This occurs only if J_i is released after J_j tries to enter a critical section but before J_j exits the critical section. When J_i is released, only one job can try to enter or be inside a critical section. That is, a higher-priority job J_i can then be interfered by at most a single lower-priority job. The processor demand of J_j during a critical section period is bounded by $2 \cdot max(h_{j,k})$ for the worst case. The lemma follows. □

From Lemma 1, the self-blocking $I_S(i)$ is given by;

$$I_S(i) = \sum_{k=1}^{o} h_{i,k} \tag{5}$$

According to Lemma 2 and taking into account the interference from higher priority tasks, $I_H(i,t)$ is computed as follows;

$$I_H(i,t) = \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil (C_j + \sum_{k=1}^{o} h_{j,k}). \tag{6}$$

The maximum interference from lower priority tasks can be evaluated according to Lemma 3 according to;

$$I_L(i) = \max_{j=i+1,\ldots,n} (2 \cdot \max_{k=1,\ldots,o} (h_{j,k})). \tag{7}$$

Based on Eq. (5) and (6) and (7), the processor demand bound function is given by Eq. (4).

The resource holding time $h_{i,j}$ of a job J_i that access a global resource is evaluated as the maximum critical section execution time $x_{i,j}+$ the maximum interference from the tasks that have preemption level greater than the ceiling of the logical resource during the execution $x_{i,j}$. $h_{i,j}$ is computed [4] using $W_{i,j}(t)$ as follows;

$$W_{i,j}(t) = x_{i,j} + \sum_{l=ceil(x_{i,j})+1}^{u} \lceil \frac{t}{T_l} \rceil C_l, \tag{8}$$

where $ceil(x_{i,j})$ is the ceiling of the logical resource accessed within the critical section $x_{i,j}$, and C_l, T_l are the worst case execution time and the period of job that have higher preemption level than $ceil(x_{i,j})$, and u is the maximum ceiling within the subsystem.

The resource holding time $h_{i,j}$ is the smallest time t_i^* such that $W_{i,j}(t_i^*) = t_i^*$.

5.2 Global schedulability analysis

Here, issues for global scheduling of multiple subsystems are dealt with. For a subsystem S_s, it is possible to derive a periodic virtual processor model $\Gamma_s(\Pi_s, \Theta_s)$ that guarantees the schedulability of the subsystem S_s according to Eq.(3).

The local schedulability analysis presented for subsystems is not dependent on any specific global scheduling policy. The requirements for the global scheduler, are as follows: i) it should schedule all subsystems according to their virtual processor model $\Gamma_s(\Pi_s, \Theta_s)$, ii) it should be able to bound the waiting time of a task in any subsystem that wants to access global resource.

To achieve those global scheduling requirements, preemptive schedulers such as EDF and RM together with the SRP [3] synchronization protocol can be used. So when a subsystem locks a global resource, it will not be preempted by other subsystems that have preemption level less than or equal to the locked resource ceiling. Each subsystem, for all global resources accessed by tasks within a subsystem, should specify a list of pairs of all those global resources and their maximum resource holding times $\{(r_1, H_{r_1}), ..., (r_p, H_{r_p})\}$. However it is possible to minimize the required information that should be provided for each subsystem by assuming that all global resources have the same ceiling equal to the maximum preemption level $\hat{\pi}_s$ among all subsystems. Then for the global scheduling, it is enough to provide virtual processor model $\Gamma_s(\Pi_s, \Theta_s)$ and the maximum resource holding times among all global resources $\hat{H}_s = max(H_{R_1}, ..., H_{R_p})$ for each subsystem S_s. On the other hand, assigning the ceiling of all global resources to the maximum preemption level of the subsystem that access these resources is not as efficient as using the original SRP protocol, this since we may have resources with lower ceiling which permit more preemptions from the higher preemption level subsystems.

Under EDF global scheduling, a set of n subsystems is schedulable [3] if

$$\forall k_{k=1,\dots,n} (\sum_{i=1}^{k} \frac{\Theta_i}{\Pi_i}) + \frac{B_k}{\Pi_k} \leq 1, \qquad (9)$$

where B_k of subsystem S_k is the duration of the longest resource holding time among those belonging to subsystems with preemption level lower than π_k.

For RM global scheduling, the schedulability test based on tasks' response time is

$$W_i = \Theta_i + B_k + \sum_{j=1}^{i-1} \left\lceil \frac{W_i}{\Pi_j} \right\rceil (C_j). \qquad (10)$$

It is also possible to use a non-preemptive global scheduler together with the SIRAP protocol. In this case, no subsystem-level context switch happens when there is a task inside a critical section. That is, whenever a task tries to lock a global resource, it is guaranteed that the global resource is not locked by another task from other subsystems. This way provides a clean separation between subsystems in accessing global shared resources. Then, we can achieve a more subsystem abstraction, i.e., subsystems do not have to export information about their global shared resource accesses, for example, which global shared resources they access and the maximum resource holding time. In fact, it will require more system resources to schedule subsystems under non-preemptive global scheduling rather than under preemptive global scheduling. Hence, we can see a tradeoff between abstraction and efficiency. Exploring this tradeoff is a topic of our future work.

5.3 Local resource sharing

So far, only the problem of sharing global resource between subsystems has been considered. However, many real time applications may have local resource sharing within subsystem as well. Almeida and Pedreiras [1] showed that some traditional synchronization protocols such as PCP and SRP can be used for supporting local resource sharing in a hierarchical scheduling framework by including the effect of local resource sharing in the calculation of $\mathtt{dbf_{FP}}$. That is, to combine SRP/PCP and the SIRAP protocol for synchronizing both local and global resources sharing, Eq. (7) should be modified to

$$I_L(i) = \max(\max(2 \cdot x_{j,k}), b_i), \quad \text{where } j = i+1, \dots, n. \qquad (11)$$

where b_i is the maximum duration for which a task i can be blocked by its lower-priority tasks in critical sections from local resource sharing.

6. PROTOCOL EVALUATION

In this section, the cost of using SIRAP is investigated in terms of extra CPU utilization (U_Γ) required for subsystem schedulability guarantees. We assume that all global resource ceilings can be equal to the maximum preemption level, which means that no tasks within a subsystem preempt a task inside a critical section, and therefore $h_{i,j} = x_{i,j}$. Supporting logical resource sharing is expected to increase subsystem utilizations U_Γ. This increment in U_Γ depends on many factors such as the maximum WCET within a critical section $x_{i,j}$, the priority of the task sharing a global resource, and the subsystem period Π_s.

Sections 6.1, 6.2, and 6.3 investigate the effect of those factors under the assumption that task i accesses a single critical section. In Section 6.4, this assumption is relaxed so as to investigate the effect of the number of critical sections. Section 6.5 compares independent and dependent abstractions in terms of subsystem utilization.

6.1 WCET within critical section

One of the main factors that affect the cost of using SIRAP is the value of $x_{i,j}$. It is clear from Eqs. (4), (6), and (7) that whenever $x_{i,j}$ (which equals to $h_{i,j}$) increases, $\mathtt{dbf_{FP}}$ will increase as well, potentially causing U_Γ to increase in order to satisfy the condition in Eq. (3). Figure 4 shows the effect of increasing x_i on two different task sets. Task set 1 is sensitive for small changes in x_i whilst task set 2 can tolerate the given range of x_i without showing a big change in U_Γ. The reason behind the difference is that task set 1 has a task with period very close to Π_s while the smallest task period in task set 2 is greater than Π_s by more than 4 times. Hence,

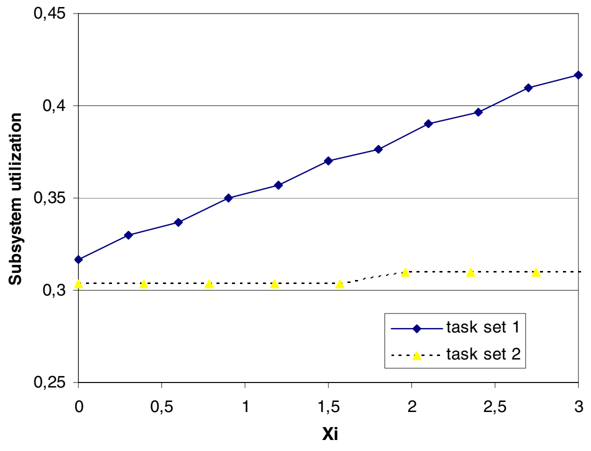

Figure 4: U_Γ as a function of x_i for two task sets where only the lowest priority task share a resource.

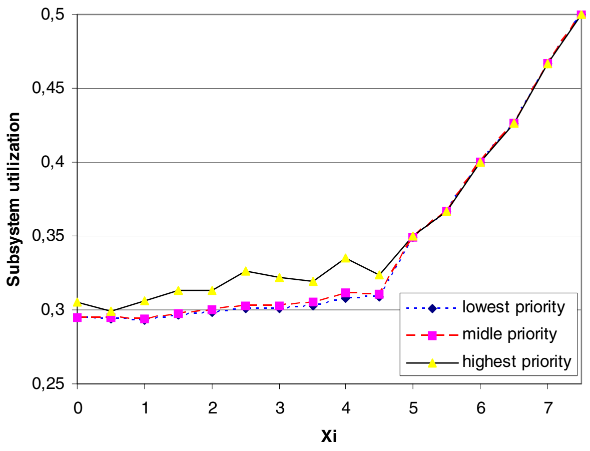

Figure 5: Average utilization for 100 task sets as a function of x_i, when low, medium and high priority task share a resource respectively, $\Pi_s = 15$.

SIRAP can be more or less sensitive to x_i depending on the ratio between task and subsystem period.

For the remaining figures (Figure 5 and 6), simulations are performed as follows. We randomly generated 100 task sets, each containing 5 tasks. Each task set has a utilization of 25%, and the period of the generated tasks range from 40 to 1000. For each task set, a single task accesses a global shared resource; the task is the highest priority task, the middle priority task, or the lowest priority task. For each task set, we use 11 different values of x_i ranging from 10% to 50% of the subsystem period.

6.2 Task priority

From Eqs. (4), (6) and (7), looking how tasks sharing global logical resources affect the calculations of $\mathtt{dbf}_{\mathtt{FP}}$, it is clear that task priority for these tasks is of importance. The contribution of low priority tasks on $\mathtt{dbf}_{\mathtt{FP}}$ is fixed to a specific value of x_i (see Eq. (7)), while the increase in $\mathtt{dbf}_{\mathtt{FP}}$ by higher priority tasks depends on many terms such as higher priority task period T_k and execution time C_k (see Eq. (6)). It is fairly easy to estimate the behaviour of a subsystem when lower priority tasks share global resources; on one hand, if the smallest task period in a subsystem is close to Π_s, U_Γ will be significantly increased even for small values of x_i. As the value of `sbf` is small for time intervals close to Π_s, the subsystem needs a lot of extra resources in order to fulfil subsystem schedulability. On the other hand, if the smallest task period is much larger than Π_s then U_Γ will only be affected for large values of x_i, as shown in Figure 4.

Figure 5 shows U_Γ as a function of x_i for when the highest, middle and lowest priority task are sharing global resources, respectively, where $\Pi_s = 15$. The figure shows that the highest priority task accessing a global shared resource needs in average more utilization than other tasks with lower priority. This observation is expected as the interference from higher priority task is larger than the interference from lower priority tasks (see Eq. (6) and (7)). However, note that in the figure this is true for x_i within the range of [0,5]. If the value of x_i is larger than 5, then U_Γ keeps increasing rapidly without any difference among the priorities of tasks accessing the global shared resource. This can be explained as follows. When using SIRAP, the subsystem budget Θ_s should be no smaller than x_i to enforce the second rule R2 in Section 4.2. Therefore, when $x_i \geq 5$, Θ_s should also become greater than 5 even though subsystem period is fixed to 15. This essentially results in a rapid increase of U_Γ with the speed of $x_i/15$.

6.3 Subsystem period

The subsystem period is one of the most important parameters, both in the context of global scheduling and `sbf` calculations for a subsystem. As Π_s is used in the `sbf` calculations, Π_s will have significant effect on U_Γ (see Eq. (3)).

Figure 6 compares average subsystem utilization for different values of subsystem period, i.e., for $\Pi_s = 20$ and $\Pi_s = 40$ for the same task sets. Here, only the highest priority task accesses a global shared resource. It is interesting to see that the lower value of Π_s, i.e, $\Pi_s = 20$, results in a lower subsystem utilization when x_i is small, i.e., $x_i \leq 6$, and then a higher subsystem utilization when x_i gets larger from $x_i = 6$. That is, x_i and Π_s are not dominating factors one to another, but they collectively affect subsystem utilization. It is also interesting to see in Figure 6 that the subsystem utilization of $\Pi_s = 40$ behaves in a similar way by increasing rapidly from $x_i = 14$.

Hence, in general, Π_s should be less than the smallest task period in a subsystem, as in hierarchical scheduling without resource sharing, the lower value of Π_s gives better results (needs less utilization). However, in the presence of global resources sharing, the selection of the subsystem period depends also on the maximum value of x_i in the subsystem.

6.4 Multiple critical sections

We compare the case when a task i accesses multiple critical sections (MCS) with the case when a task j accesses a single critical section (SCS) within duration $x_j = \sum_{k=1}^{o} x_{i,k}$ according to the demand bound function calculations in Eq. (4). The following shows the effect of accessing MCS by a task on itself and on higher and lower priority tasks;

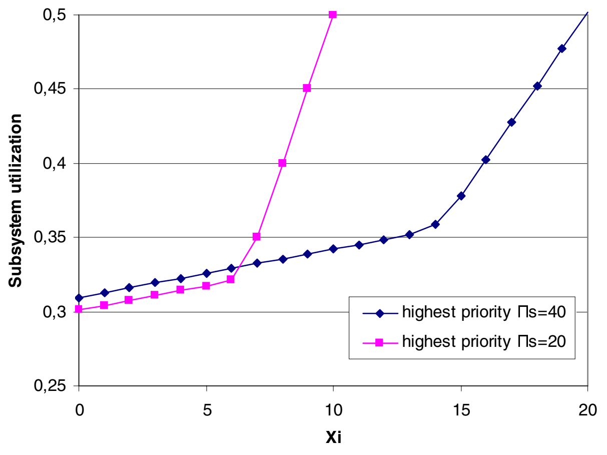

Figure 6: Average utilization for 100 task sets as a function of x_i, when only the highest priority tasks share a resource and the subsystem period is $\Pi_s = 20$ and $\Pi_s = 40$.

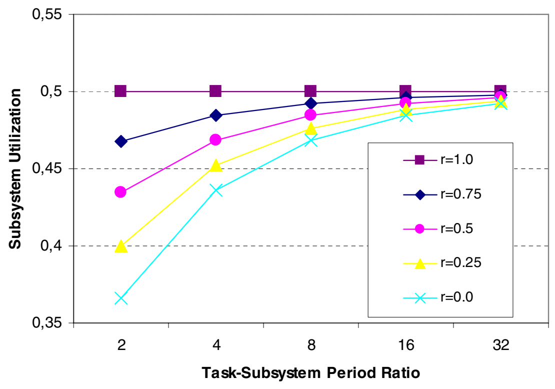

Figure 7: Comparison between independent and dependent abstractions in terms of subsystem utilization.

- Self blocking, Eq. (5) shows that both accessing MCS and SCS by a task gives the same result.
- Higher priority task, the effect from higher priority task accessing MCS or SCS can be evaluated by Eq. (6). I_H will be the same for both cases also.
- Lower priority task, Eq. (7) shows that I_L for MCS is less than SCS case because in MCS the maximum of $x_{i,j}$ will be less than x_i for SCS.

We can conclude that the required subsystem utilization for MCS case will be always less than or equal to the case of SCS having $x_j = \sum_{k=1}^{o} x_{i,k}$, which means that our proposed protocol is scalable in terms of the number of critical sections.

6.5 Independent abstraction

In this paper, we have proposed a synchronization protocol that supports independent abstraction of a subsystem, particularly, for open systems. Independent abstraction is desirable since it allows subsystems to be developed and validated without knowledge about temporal behavior of other subsystems. In some cases, subsystems can be abstracted *dependently* of others when some necessary information about all the other subsystems is available. However, dependent abstraction has a clear limitation to open systems where such information is assumed to be unavailable. In addition, dependent abstraction is not good for dynamically changing systems, since it may be no longer valid when a new subsystem is added. Despite of the advantages of independent abstraction vs. dependent abstraction, however, one may wonder what costs look like in using independent abstraction in comparison with using dependent abstraction. In this section, we discuss this issue in terms of resource efficiency (subsystem resource utilization).

One of the key differences between independent and dependent abstractions is how to model a resource supply provided to a subsystem, more specifically, how to characterize the longest *blackout duration* during which no resource supply is provided. Under independent abstraction, the longest blackout duration is assumed to be the worst-case (maximum) one. Whereas, it can be exactly identified by some techniques [7, 5] under dependent abstraction. This difference inherently yields different subsystem resource utilizations, as illustrated in Figure 7. Before explaining this figure, we need to establish some notions and explain how to obtain this figure.

We first extend the periodic resource model $\Gamma(\Pi, \Theta)$ by introducing an additional parameter, *blackout duration ratio (r)*. We define r as follows. Let L_{min} and L_{max} denote the minimum and maximum possible blackout duration, and

$$L_{min} = \Pi - \Theta \text{ and } L_{max} = 2(\Pi - \Theta).$$

When exactly computed, the longest blackout duration can then be represented as $r \cdot (L_{max} - L_{min}) + L_{min}$. We generalize the supply bound function of Eq. (2) with the blackout duration ratio r as follows:

$$\texttt{sbf}_\Gamma(t) = \begin{cases} t - (k+1)(\Pi - \Theta) & \text{if } t \in [k\Pi - \Theta \\ & +r(\Pi - \Theta), \\ & k\Pi + r(\Pi - \Theta)], \\ (k-1)\Theta & \text{otherwise,} \end{cases} \quad (12)$$

where $k = \max\Big(\lceil(t - (\Pi - \Theta))/\Pi\rceil, 1\Big)$.

We here explain the notion of *task-subsystem period ratio*, which is the x-axis of the figure. Suppose a periodic resource model $\Gamma_1(\Pi_1, \Theta_1, r_1)$ is an abstraction that guarantees the schedulability of a subsystem S. According to Eq. (3), there then exists a time instant t_i^*, where $0 < t_i^* \le T_i$, for each task τ_i within the subsystem S such that

$$\forall \tau_i, \quad \texttt{dbf}_{\texttt{FP}}(i, t_i^*) \le \texttt{sbf}_{\Gamma_1}(t_i^*). \quad (13)$$

In fact, given the values of subsystem period Π and blackout duration ratio r, we can find a smallest value of Θ, denoted as Θ_i^*, that can satisfy Eq. (13) at t_i^* for each task τ_i. The value of budget Θ is then finally determined as the maximum value among all Θ_i^*. This way makes sure that Θ is large enough to guarantee the timing requirements of all tasks. Let T^* denote a time instant t_k^* such that Θ_k^* is the maximum among the ones. We can see that $T^* \in [T_{min}, T_{max}]$, where T_{min} and T_{max} denote the minimum and maximum task periods within subsystem, respectively. We define the *task-subsystem period ratio* as T^*/Π.

Given a periodic abstraction Γ_1 of the subsystem S, another periodic resource model $\Gamma_2(\Pi_2, \Theta_2, r_2)$ can be also an abstraction of S, if

$$\forall \tau_i, \quad \mathtt{sbf}_{\Gamma_1}(t_i^*) \leq \mathtt{sbf}_{\Gamma_2}(t_i^*), \tag{14}$$

since Eq. (3) can be satisfied with S and Γ_2 as well. More specifically, $\Gamma_2(\Pi_2, \Theta_2, r_2)$ can be an abstraction of S, if

$$\mathtt{sbf}_{\Gamma_1}(T^*) \leq \mathtt{sbf}_{\Gamma_2}(T^*). \tag{15}$$

That is, given Γ_1 and the values of Π_2 and r_2, we can find the minimum value of Θ_2 that satisfies Eq. (15).

Figure 7 shows subsystem utilizations of periodic abstractions under different values of blackout duration ratio r, when they have the same subsystem period in abstracting the same subsystem. In general, it shows that dependent abstraction, which can exactly identify the value of r, would produce more resource-efficient subsystem abstractions. Specifically, for example, when $r = 0$, i.e., when the subsystem has the highest priority under fixed-priority global scheduling, a subsystem can be abstracted with 15% less subsystem utilization than in the case of independent abstraction ($r = 1$). The figure also shows that differences in subsystem utilization generally decrease when the task-subsystem period ratio increases and/or the blackout duration ratio increases. For example, when $r = 0.5$, i.e., when the system has a moderately high utilization and subsystems have medium or low priorities under fixed-priority global scheduling or subsystems are scheduled under global EDF scheduling, differences are shown to be smaller than 8%.

7. CONCLUSION

In this paper we have presented the novel Subsystem Integration and Resource Allocation Policy (SIRAP), which provides temporal isolation between subsystems that share logical resources. Each subsystem can be developed, tested and analyzed without knowledge of the temporal behaviour of other subsystems. Hence, integration of subsystems, in later phases of product development, will be smooth and seamless.

We have formally proven key features of SIRAP such as bounds on delays for accessing shared resources. Further, we have provided schedulability analysis for tasks executing in the subsystems; allowing for use of hard real-time application within the SIRAP framework.

Naturally, the flexibility and predictability offered by SIRAP comes with some costs in terms of overhead. We have evaluated this overhead through a comprehensive simulation study. From the study we can see that the subsystem period should be chosen as much smaller than the smallest task period in a subsystem and take into account the maximum value of h_i in the subsystem to prevent having high subsystem utilization. Future work includes investigating the effect of context switch overhead on subsystem utilization together with the subsystem period and the maximum value of h_i.

8. REFERENCES

[1] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *Proc. of the Fourth ACM International Conference on Embedded Software*, September 2004.

[2] D. Andrews, I. Bate, T. Nolte, C. M. O. Pérez, and S. M. Petters. Impact of embedded systems evolution on RTOS use and design. In G. Lipari, editor, *Proceedings of the 1st International Workshop Operating System Platforms for Embedded Real-Time Applications (OSPERT'05) in conjunction with the 17th Euromicro International Conference on Real-Time Systems (ECRTS'05)*, pages 13–19, Palma de Mallorca, Balearic Islands, Spain, July 2005.

[3] T. P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, March 1991.

[4] M. Bertogna, N. Fisher, and S. Baruah. Static-priority scheduling and resource hold times. In *Proceedings of the 15th International Workshop on Parallel and Distributed Real-Time Systems*, Long Beach, CA, March 2007.

[5] R. J. Bril, W. F. J. Verhaegh, and C. C. Wust. A cognac-glass algorithm for conditionally guaranteed budgets. In *RTSS '06: Proceedings of the 27th IEEE International Real-Time Systems Symposium*, pages 388–400, 2006.

[6] M. Caccamo and L. Sha. Aperiodic servers with resource constraints. In *IEEE Real-Time Systems Symposium*, pages 161–170, 2001.

[7] R. I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS'05)*, December 2005.

[8] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, December 2005.

[9] Z. Deng and J. W.-S. Liu. Scheduling real-time applications in an open environment. In *Proc. of IEEE Real-Time Systems Symposium*, pages 308–319, December 1997.

[10] X. Feng and A. Mok. A model of hierarchical real-time virtual resources. In *Proc. of IEEE Real-Time Systems Symposium*, pages 26–35, December 2002.

[11] N. Fisher, M. Bertogna, and S. Baruah. Resource-locking durations in edf-scheduled systems. In *13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07)*, pages 91–100, 2007.

[12] P. Holman and J. H. Anderson. Locking under pfair scheduling. *ACM Trans. Comput. Syst.*, 24(2):140–174, 2006.

[13] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal (British Computer Society)*, 29(5):390–395, October 1986.

[14] H. Kopetz, R. Obermaisser, P. Peti, and N. Suri. From a federated to an integrated architecture for dependable embedded real-time systems. Technical Report 22, Technische Universität at Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2004.

[15] T.-W. Kuo and C. Li. A fixed-priority-driven open environment for real-time applications. In *Proc. of IEEE Real-Time Systems Symposium*, pages 256–267, December 1999.

[16] G. Lipari and S. Baruah. Efficient scheduling of real-time multi-task applications in dynamic systems.

In *Proc. of IEEE Real-Time Technology and Applications Symposium*, pages 166–175, May 2000.

[17] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *Proc. of Euromicro Conference on Real-Time Systems*, July 2003.

[18] G. Lipari, J. Carpenter, and S. Baruah. A framework for achieving inter-application isolation in multiprogrammed hard-real-time environments. In *Proc. of IEEE Real-Time Systems Symposium*, December 2000.

[19] G. Lipari, P. Gai, M. Trimarchi, G. Guidi, and P. Ancilotti. A hierarchical framework for component-based real-time systems. In *Component-Based Software Engineering*, volume LNCS-3054/2004, pages 209–216. Springer Berlin / Heidelberg, May 2005.

[20] S. Matic and T. A. Henzinger. Trading end-to-end latency for composability. In *Proc. of IEEE Real-Time Systems Symposium*, pages 99–110, December 2005.

[21] A. Mok, X. Feng, and D. Chen. Resource partition for real-time systems. In *Proc. of IEEE Real-Time Technology and Applications Symposium*, pages 75–84, May 2001.

[22] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of the* 9^{th} *IEEE International Real-Time Systems Symposium (RTSS'88)*, pages 259–269, December 1988.

[23] S. Saewong, R. Rajkumar, J. Lehoczky, and M. Klein. Analysis of hierarchical fixed-priority scheduling. In *Proc. of Euromicro Conference on Real-Time Systems*, June 2002.

[24] L. Sha, J. P. Lehoczky, and R. Rajkumar. Task scheduling in distributed real-time systems. In *Proceedings of the International Conference on Industrial Electronics, Control, and Instrumentation*, pages 909–916, Cambridge, MA, USA, November 1987.

[25] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proc. of IEEE Real-Time Systems Symposium*, pages 2–13, December 2003.

[26] I. Shin and I. Lee. Compositional real-time scheduling framework. In *Proc. of IEEE Real-Time Systems Symposium*, December 2004.