SIRAP: A Global Resource Sharing Protocol Facilitating Integration of Semi-independent Real-Time Systems*

Moris Behnam†, Insik Shin, Thomas Nolte, Mikael Nolin
Mälardalen Real-Time Research Centre (MRTC)
Västerås, SWEDEN

Abstract

This paper presents a protocol for resource sharing in a hierarchical real-time scheduling framework. Together, 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.

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 [3, 12]. 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.

In this paper we present the Subsystem Integration and Resource Allocation Policy (SIRAP), that 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

*The work in this paper is supported by the Swedish Foundation for Strategic Research (SSF), via the research programme PROGRESS.
†contact author: moris.behnam@mdh.se

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 [17]. 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 a critical section, cost depending on the priority of the task sharing a resource, and cost depending on the periodicity of the subsystem.

Organisation 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. Section 4 presents some general issues of resource sharing in hierarchical scheduled systems. SIRAP is presented in Section 5 and formally evaluated in Section 6. In Section 7 schedulability analysis is presented, and SIRAP is evaluated in Section 8. Finally, the paper is summarised in Section 9.

2 Related work

Hierarchical scheduling For real-time systems, there has been a growing attention to hierarchical scheduling frameworks [2, 6, 8, 9, 13, 14, 15, 19, 22, 24, 25].

Deng and Liu [8] proposed a two-level hierarchical scheduling framework for open systems, where subsystems may be developed and validated independently in different environments. Kuo and Li [13] presented schedulability analysis techniques for such a two-level framework with the fixed-priority global scheduler. Lipari and Baruah [14, 16] presented schedulability analysis techniques for the EDF global scheduler, using the Constant bandwidth Server (CBS) [1] and the Bandwidth Sharing Server (BSS) [27], respectively.

Mok *et al.* [20] 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 [9] and Shin and Lee [25] 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 [24]. Saewong *et al.* [22] and Lipari and Bini [15] introduced schedulability conditions for fixed-priority local scheduling, and Shin and Lee [24] presented a schedulability condition for EDF local scheduling. Davis and Burns [6] 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) [23], the Priority Ceiling Protocol (PCP) [21], and the Stack Resource Policy (SRP) [4].

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 work by Kuo and Li [13] used SRP and they showed that it is very suitable for sharing of local resources in a hierarchical scheduling framework. However, SRP is not very good for sharing global resources. This was addressed by introducing a common server for all globally shared resource accesses. However, having such a server can be very costly [7].

Deng and Liu [8] proposed the usage of non-preemptive global resource access, which bounds the maximum blocking time that a task might be subject to. However, as resource access is non-preemptive, all tasks in a subsystem are affected by blocking.

Almeidia and Pedreiras [2] considered the issue of supporting mutually exclusive resource sharing within a subsystem. Matic and Henzinger [19] considered supporting interacting tasks with data dependency within a subsystem and between subsystems, respectively.

More recently, Davis and Burns [7] presented the Hierarchical Stack Resource Policy (HSRP), allowing their work on hierarchical scheduling [6] 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 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.

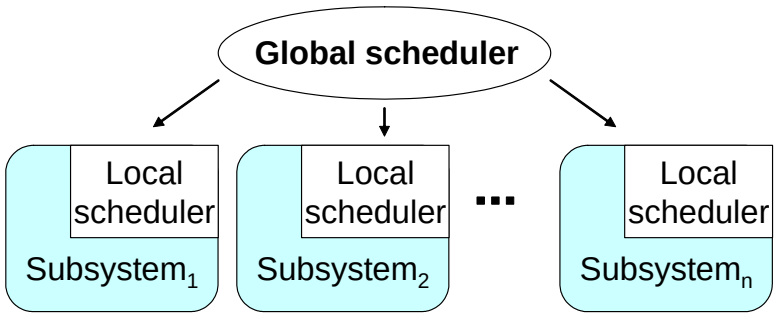

Figure 1. Two-level hierarchical scheduling framework.

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 [8], 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 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 is a matter of the local scheduler of each subsystem.

3.3 Virtual processor model

The notion of real-time virtual processor (resource) model was fist introduced Mok *et al.* [20] 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 provides. 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 [24] 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:

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

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

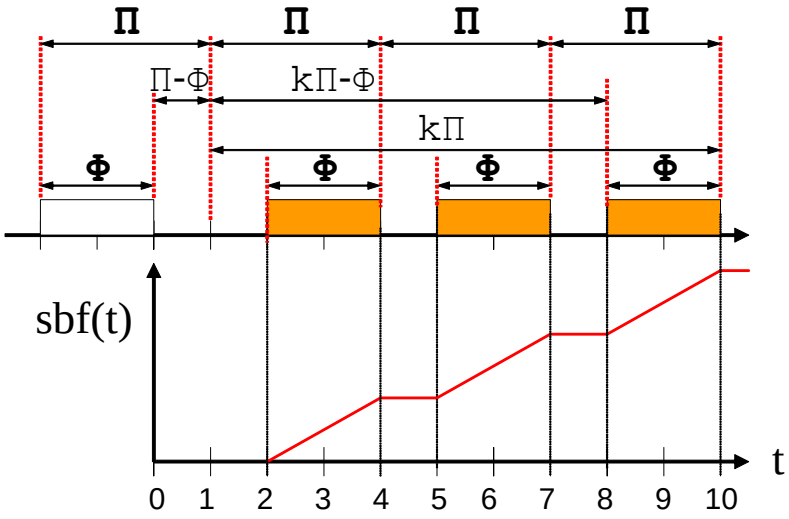

Figure 2. The supply bound function of a periodic virtual processor model $\Gamma(\Pi, \Theta)$ for $k = 3$.

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

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

where $k = \max\Big(\big\lceil (t - (\Pi - \Theta))/\Pi \big\rceil, 1\Big)$. Here, we first note that an interval of length t may not start 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 $\mathtt{sbf}_\Gamma(t)$ 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 follows the periodic resouce model and are represented by a corresponding $\Gamma_s(\Pi_s, \Theta_s)$, where Π_s and Θ_s are the subsystem period and budget respectively.

Task model We consider a periodic task model $\tau_i(T_i, C_i, X_i)$[1], where T_i is a period, C_i is a worst-case execution time requirement (WCET), and X_i is a WCET within a critical section; C_i includes X_i. We only consider properly nested shared resource accesses, and X_i is the length of the outer critical section.

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

4 Synchronization and hierarchical scheduling

In this section we present the implications of introducing shared logical resources in the context of a hierarchical scheduling framework. First, we define the terminology (also depicted in Figure 3) used throughout the remainder of paper as follows:

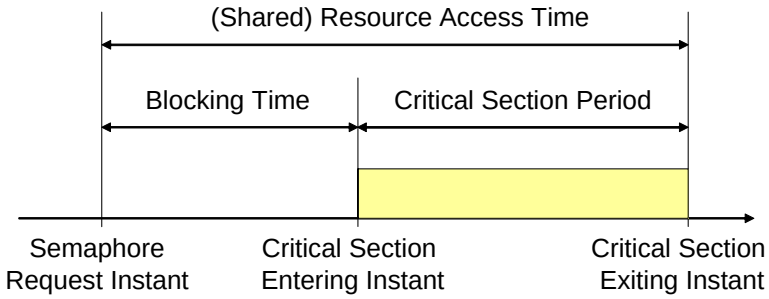

Figure 3. Shared resource access time.

- *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.
- *Blocking time:* a duration from a semaphore request time to a critical section entering time.
- *Critical section period:* a duration from a critical section entering instant to a critical section exiting instant.
- *(Shared) resource access time:* a duration from a semaphore request instant to a critical section exiting time.

Also, we refer *task-level context switch* to the context switch that happens between tasks within a subsystem and *subsystem-level context switch* to the context switch that happens between subsystems.

Critical section period In a priority scheduled system we can calculate an upper bound for the interference a job J_i (with priority i) may experience from other higher priority jobs J_k (where $k < i$) during an interval t. We denote this worst case interference by $I(i,t)$, given by

$$I(i,t) = \sum_{k=1}^{i-1} \left\lceil \frac{t}{T_k} \right\rceil C_k \tag{3}$$

[1]The task model is refined to include multiple shared resource accesses in Section 7.4.

For a subsystem that runs over a virtual processor model Γ in a hierarchical scheduling framework, we present a method $\texttt{csp}(J_i)$ that can compute the worst-case critical section period of a job J_i within the subsystem as follows:

$$\texttt{csp}(J_i) = \min\left\{t | I(i,t) + X_i \leq \texttt{sbf}_\Gamma(t)\right\} \tag{4}$$

Note that, as given by Eq. (3), the worst case critical section period $\texttt{csp}(J_i)$ is relying on the number of higher priority tasks and their execution times, and might be potentially much larger that the length of the critical section in isolation. This is an important observation, as it complicates logical resource sharing among subsystems in a hierarchical scheduling framework. Specifically, if a subsystem is allowed to be preempted, the interference among subsystems sharing a logical resource must be bounded. This can be achieved by the usage of a synchronization protocol such as SIRAP, described in Section 5.

Blocking time Looking at Figure 3, we now consider how to compute the blocking time of a job J_i on a semaphore ϕ. Let J_ϕ denote the job that is currently within the critical section guarded by ϕ and $Q(\phi)$ denote a set of jobs that are supposed to enter the critical section ahead of J_i.

$$\texttt{bt}(J_i, \phi) = \texttt{csp}(J_\phi) + \sum_{J_k \in Q(\phi)} \texttt{csp}(J_k) \tag{5}$$

What is expressed by Equation 5 is that the blocking time, i.e., the time before the job is allowed to enter the critical section, might be potentially large. Here, appropriate mechanisms could improve the blocking time of a job substantially. Issues that a synchronization protocol for a hierarchical scheduling framework must deal with include (1) bounding the critical section period and (2) bounding the blocking time. In the next section, we propose a resource access protocol that minimizes the critical section time and bounds the maximum blocking time of each task.

5 SIRAP: Protocol description

Recalling the problem description in the introduction of this paper, it is desirable that a synchronization protocol for hierarchical scheduling allows for independent development of subsystems. In this section, we describe the protocol proposed in this paper: SIRAP (Subsystem Integration and Resource Allocation Policy).

Assumptions SIRAP relies on the following assumptions:

A1 The global scheduler can inform the local scheduler about how much time its has to execute. More precisely, the global scheduler must be able to answer questions of the form "can I be guaranteed to get at least t more time-units of execution before the next system-level context switch?". Fulfilling this assumption is trivial when the global scheduler is a non-preemptive scheduler or a TDMA scheduler, but many other schedulers can be used as well.

A2 Each local scheduler knows when its internal task tries to enter a critical section and exits the critical section respectively.

A3 The unit of CPU-time allocation, with which the global scheduler provides subsystems, is no smaller than the maximum of X_i for all jobs J_i across subsystems.

The reason for A1 and A2 is to allow for run-time checking whether or not a job can potentially enter and execute a whole critical section before a subsystem-level context switch. This is useful when allowing for independent subsystem development. Assumption A3 is needed to make sure that any critical section actually fits within the duration of execution of a subsystem before its corresponding subsystem-level context switch. Moreover, a runtime mechanism is required to enforce that a job J_i does not spend more time than specified by X_i inside a critical section. However, the implementation of such a runtime mechanism is beyond the scope of this paper.

Preemption level As introduced in [4], we use the notion of *preemption level*. Each job J_i is associated with its own preemption level $\pi(J_i)$, which is defined as a positive integer. A job J_i is said to have a higher preemption level than another job J_k if $\pi(J_k) < \pi(J_i)$. A job J_i is not allowed to preempt another job J_k unless $\pi(J_k) < \pi(J_i)$. In this paper, let π^* denote the highest preemption level of the current subsystem.

Protocol outline SIRAP aims at minimizing the critical section period and bounding the blocking time at the same time as allowing for independent subsystem development. To achieve this goal, the protocol has two key rules as follows:

R1 When a job tries to enter a critical section, its preemption level becomes the highest preemption level within the same subsystem. The job has the highest preemption level until it exits the critical section. This way guarantees that the job does not get interfered by higher-priority jobs within the same subsystem during a resource access time.

R2 After trying to enter the critical section, the job enters the critical section at the earliest instant such that it can complete the critical section prior to the earliest subsystem-level context switch.

The first rule R1 allows avoiding a task-level preemption caused by the local scheduler, and the second rule R2 allows to avoid that any task are within a critical section during a subsystem-level context switch.

As the second rule R2 prevents subsystem-level context switches from occurring while a job is within a critical section, it guarantees that every lock is always free whenever a subsystem-level context switch happens.

SIRAP : preemption level management SIRAP requires a simple preemption level management. It just requires modification of the preemption level of a job when it tries to enter a critical section and exits the critical section.

- When a job J is released, its preemption level $\pi(J)$ is initialized to its priority $p(J)$, i.e., $\pi(J) = p(J)$.
- When the job J tries to enter a critical section, its preemption level is raised to the highest preemption level, i.e., $\pi(J) = \pi^*$.
- When the job J exits the critical section, its preemption level goes down back to its priority $p(J)$, i.e., $\pi(J) = p(J)$.

Note that a more refined preemption level management could be used. However, for globally shared logical resources this simple approach has been used before [7], and it significantly decreases predictability complexity.

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 t_1 denote the earliest instant at which a subsystem-level context switch occurs such that $t_0 < t_1$.

- If $X_i \leq t_1 - t_0$, the local scheduler executes the job J_i. The job J_i enters a critical section at time t_0.
- Otherwise, i.e., if $X_i > t_1 - t_0$, the local scheduler delays the critical section entering of the job J_i until t_1. This is defined as *self-blocking*, which means that the subsystem is running the CPU idle[2]. Note that the preemption level of the job J_i is raised to the highest preemption level at time t_0, but its execution is delayed until t_1. This guarantees that when the subsystem of J_i receives the next resource allocation, the job J_i still has the highest preemption level and the local scheduler executes the job J_i so that it can enter the critical section.

The implication of this approach is that a job J_i will in the worst case spend $2X_i$ time units for shared resource access time, i.e., looking at Figure 3, both blocking time and critical section period have each a duration of X_i.

6 Properties of the protocol

In this section, we present the main properties of SIRAP. These properties are then used to analyze the schedulability of a subsystem.

Lemma 1 *No job will be preempted by other jobs within the same subsystem for the duration in between the time when it tries to enter a critical section until the time when it exits the critical section.*

Proof. When a job tries to enter a critical section, its preemption level becomes the highest preemption level of its belonging subsystem until it exits the critical section. The lemma follows. □

[2]One can think of utilizing this idle time by executing other jobs within the same component or by yielding the processor. However, this is off the point of this paper.

Lemma 2 (No within-subsystem blocking) *Whenever a job J_i tries to enter a critical section, no job within the same subsystem is inside the critical section.*

Proof. If another job J_k is inside the critical section it will have the highest preemtion level, hence preventing J_i from executing. When J_i tries to enter a critical section it is executing, hence no other job is inside the critical section. □

Lemma 3 *No subsystem-level context switch happens inside a critical section.*

Proof. A job can enter a critical section only if it can complete its critical section prior to the earliest subsystem-level context switch. The lemma follows. □

Lemma 4 (No between-subsystems blocking) *Whenever a job tries to enter a critical section, no job in another subsystem is inside the critical section.*

Proof. The lemma follows from Lemma 3. □

Theorem 5 *Whenever a job tries to enter a critical section, no job is inside the critical section.*

Proof. The theorem follows from Lemma 2 and Lemma 4. □

Lemma 6 *SIRAP is deadlock-free.*

Proof. Since a job does not get preempted while it is inside a critical section, there can be no deadlock. □

Timing properties

Lemma 7 *The critical section period of a job J_i is the same as the duration of its critical section X_i.*

Proof. Since there are no task-level and subsystem-level context switches while the job is inside critical section, it takes as much as X_i to complete the critical section even in a hierarchical scheduling framework. □

Lemma 8 *Self-blocking imposes to a job J_i an extra processor demand of at most X_i.*

Proof. When the job J_i self-blocks itself, it consume the processor of at most X_i units being idle. □

Lemma 9 *Self-blocking can increase the resource access time of a job J_i by at most X_i.*

Proof. The lemma immediately follows from Lemma 8. □

Lemma 10 *In a duration of t time units, a job J_i can be interfered by a higher-priority job J_k for a duration of at most $\lceil \frac{t}{T_k} \rceil (C_k + X_k)$ time units.*

Proof. Similar to classical response time analysis [10], the lemma follows. □

Lemma 11 *A job J_i can be interfered by only one lower-priority job J_k by at most $2X_k$.*

Proof. A higher-priority job J_i can be interfered by a lower-priority job J_k when J_k has a higher preemption level than J_i, i.e., $\pi(J_i) < \pi(J_k)$. This occurs only if J_i is released after J_k tries to enter a critical section but before J_k 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_k during a critical section period is bounded by $2X_k$. The lemma follows. □

7 Schedulability analysis

7.1 Local schedulability analysis

From Lemma 10, we present a method $I_H(i,t)$ that can compute the maximum possible interference imposed by a set of higher-priority tasks to a task τ_i during an interval of length t, as follows:

$$I_H(i,t) = \sum_{k=1}^{i-1} \left\lceil \frac{t}{T_k} \right\rceil (C_k + X_k) \tag{6}$$

From Lemma 11, we also present another method $I_L(i)$ that can compute the maximum possible interference imposed by a set of lower-priority tasks to a task τ_i, as follows:

$$I_L(i) = \max(2 \cdot X_k), \quad \text{where } k = i+1, \ldots, n \tag{7}$$

From Lemma 8, it follows that the maximum possible processor demand of a task τ_i imposed by itself is $C_i + X_i$.

Based on Eq. (6) and Eq. (7), we now present a schedulability condition as follows:

Theorem 12 *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 $\Gamma_s(\Pi_s, \Theta_s)$. This subsystem is schedulable in using SIRAP if*

$$\forall \tau_i, 0 < \exists t \leq T_i \;\; \mathtt{dbf}_{\mathsf{FP}}(i,t) \leq \mathtt{sbf}_{\Gamma}(t), \tag{8}$$

where

$$\mathtt{dbf}_{\mathsf{FP}}(i,t) = C_i + X_i + I_H(i,t) + I_L(i). \tag{9}$$

Proof. The demand bound function $\mathtt{dbf}_{\mathsf{FP}}(i,t)$ is immediately derived from Lemma 11 and 8. The condition in Eq. (8) then follows from [24]. □

7.2 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 Theorem 12.

Looking at a system with a non-preemptive EDF or RM global scheduler, the schedulability condition for non-preemptive EDF scheduling [11] can be extended for SIRAP as follows:

Theorem 13 *A set of subsystems S_s, each of which is schedulable with a periodic virtual processor model $\Gamma_s(\Pi_s, \Theta_s)$, is schedulable under a non-preemptive EDF global scheduler, if*

- $\sum U_{\Gamma_s} \leq 1$
- $\forall t > 0 \quad \mathtt{dbf}'_{\mathsf{EDF}}(t) + \max_i\{\Theta_s\} \leq t$,

where

$$\mathtt{dbf}'_{\mathsf{EDF}}(t) = \sum \left\lfloor \frac{t}{\Pi_i} \right\rfloor \cdot \Theta_i. \tag{10}$$

Moreover, the schedulability condition for non-preemptive fixed-priority scheduling [26] can be extended for SIRAP as follows:

Theorem 14 *A set of subsystems S_s, each of which is schedulable with a periodic virtual processor model $\Gamma_s(\Pi_s, \Theta_s)$, is schedulable under a non-preemptive fixed-priority global scheduler, if*

$$\forall s \quad \exists t_s \in [0, \Pi_s] \quad \mathtt{dbf}'_{\mathtt{FP}}(t_s, s) + \max_{S_k \in \mathtt{LP}(s)} \{\Theta_k\} \leq t_s,$$

where

$$\mathtt{dbf}'_{\mathtt{FP}}(t, s) = \Theta_s + \sum_{S_k \in \mathtt{HP}(s)} \left\lceil \frac{t}{\Pi_k} \right\rceil \cdot \Theta_k, \tag{11}$$

where $\mathtt{HP}(s)$ is the set of subsystems with priority higher than that of S_s, and $\mathtt{LP}(s)$ denotes a set of subsystems with priority lower than that of S_s.

Now consider TDMA scheduling as global scheduling. Here, the schedulability of subsystems can be ckecked by simulating TDMA scheduling up to the least common multiplier of all subsystem periods. For a special case where all subsystem periods are harmonic, the schedulability condition is $\sum U_{\Gamma_s} \leq 1$.

7.3 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 [2] 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

$$I_L(i) = \max(\max(2 \cdot X_k), B_i), \quad \text{where } k = i+1, \ldots, n. \tag{12}$$

and 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.

7.4 Supporting multiple shared resource accesses of a single task

Up until now, it has been an assumption that a single task accesses only a single global shared resource. Here this assumption is relaxed. We now consider an extended periodic task model $\tau_i^*(T_i, C_i, X_i^*)$, where X_i^* is a set of WCET's within critical sections, i.e., $X_i^* = \{X_{i,1}, \ldots, X_{i,k}\}$, where τ_i^* has accesses to k different global shared resources. The following discusses how this extended task model can be supported by SIRAP.

Firstly $I_H^*(i,t)$ is defined as a method that computes the maximum possible interference imposed by higher-priority tasks by replacing X_k in Eq. (6) with $\sum_{x_j \in X_k^*}(x_j)$, i.e.,

$$I_H^*(i,t) = \sum_{k=1}^{i-1} \left\lceil \frac{t}{T_k} \right\rceil (C_k + \sum_{x_j \in X_k^*} x_j). \tag{13}$$

Secondly, $I_L^*(i,t)$ replaces Eq. (7) by substituting X_k with $\max_{x_j \in X_k^*}(x_j))$ as follows:

$$I_L^*(i) = \max(2 \cdot \max_{x_j \in X_k^*}(x_j)), \quad \text{where } k = i+1, \ldots, n. \tag{14}$$

Now, the processor demand bound function is given by

$$\mathtt{dbf}^*_{\mathtt{FP}}(i,t) = C_i + \sum_{x_k \in X_i^*} x_k + I_H^*(i,t) + I_L^*(i). \tag{15}$$

Finally, Theorem 12 holds by replacing $\mathtt{dbf}_{\mathtt{FP}}(i,t)$ with $\mathtt{dbf}^*_{\mathtt{FP}}(i,t)$.

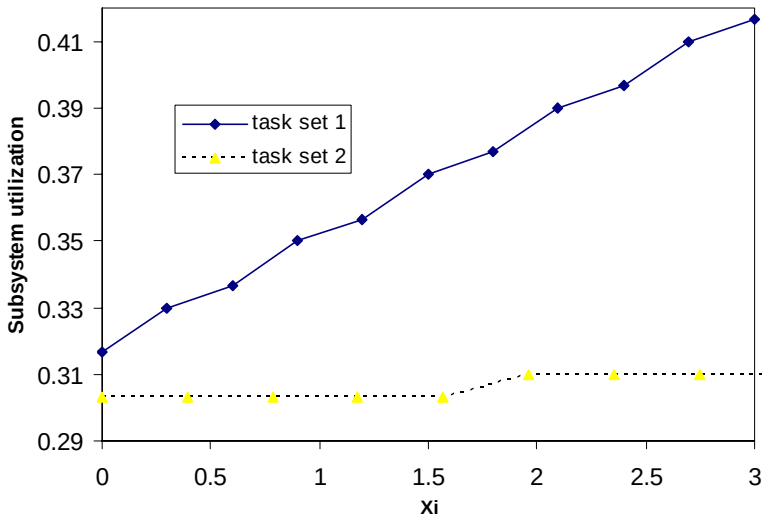

Figure 4. U_Γ **as a function of** X_i **for two task sets where only the lowest priority task share a resource.**

8 Protocol evaluation

In this section, the cost of using SIRAP is investigated in terms of extra CPU utilization (U_Γ) required for subsystem schedulability guarantee. Allowing for shared logical resources, the subsystem's required U_Γ is expected to increase. This increment in U_Γ depends on many factors such as the maximum WCET within a critical section X_i, the priority of the task sharing a global resource, and the subsystem period Π_s.

8.1 WCET within critical section

One of the factors that affect the cost to be paid using SIRAP is the value of X_i. It is clear from Eqs. (6), (7) and (9) that whenever X_i increase, $\texttt{dbf}_{\texttt{FP}}$ will increase as well, potentially causing U_Γ to increase in order to satisfy the condition in Eq. (8). 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, 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, 6, 7 and 8, 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 two subsystem periods, $\Pi_s = 15$ or $\Pi_s = 30$. For each subsystem period, we use 11 different values of X_i ranging from 10% to 50% of the subsystem period.

8.2 Task priority

From Eqs. (6), (7) and (9), looking how tasks sharing global logical resources affect the calculations of $\texttt{dbf}_{\texttt{FP}}$, it is clear that task priority for these tasks is of importance. The contribution of low priority tasks on $\texttt{dbf}_{\texttt{FP}}$ is fixed to a specific value of X_i (see Eq. (7)), while the increase in $\texttt{dbf}_{\texttt{FP}}$ 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 $\texttt{sbf}$ 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.

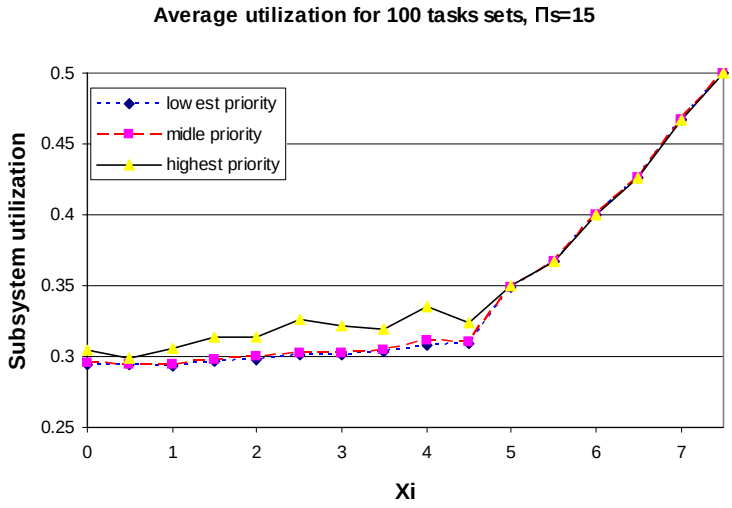

Figure 5. U_Γ **as a function of** X_i**, when low, medium and high priority task share a resource respectively,** $\Pi_s = 15$**.**

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 according to assumption A3 in Section 5. 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 $X_i/15$.

8.3 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. (8)).

Figure 6 compares average subsystem utilization for different values of subsystem period, i.e., for $\Pi_s = 15$ and $\Pi_s = 30$. 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, $\Pi_s = 15$, results in a lower subsystem utilization when X_i is small, i.e., $X_i \leq 4$, and then results in a more subsystem utilization when X_i gets larger from $X_i = 5$. 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 that subsystem utilization of $\Pi_s = 30$ keeps increasing rapidly from $X_i = 10$, which is similarly shown for $\Pi_s = 15$ in Figure 5.

Comparing Figure 7 and Figure 8, in average the highest priority tasks that share global resources need more utilization than the lower priority tasks. However, in many points, the lower priority tasks need more utilization than the higher priority tasks. This is due to that, in these points, the highest priority task period is close to the subsystem period Π_s which, in turn, makes these tasks very sensitive to a small change in $\texttt{dbf}_{\texttt{FP}}$ (as also shown in Figure 4).

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.

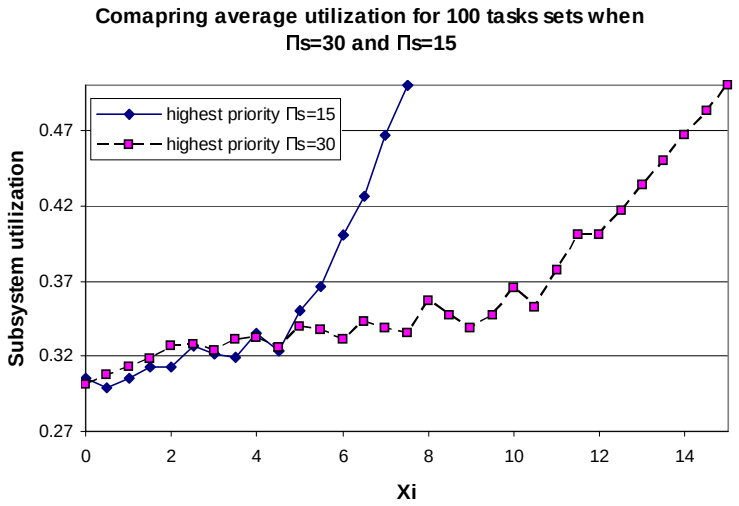

Figure 6. Average utilization between the highest priority tasks having $\Pi_s = 15$ and having $\Pi_s = 30$.

9 Conculsions

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.

SIRAP alleviates many of the problems that stem from temporal side-effects when integrating many subsystems onto a single processor. Subsystems that work perfectly fine when unit-tested (which typically means that they have the CPU to them self) often start to exhibit failures when integrated with other subsystems. A large portion of these failures comes from the changed temporal behaviour and from unpredictable (or at least unanticipated) delays when accessing shared resources.

We have formally proven key features of SIRAP such as bounds on delays for accessing shared resources and the absence of deadlocks. 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 for sensible configurations SIRAP typically gives an overhead of less than 20%. However, the study also shows that if parameters of the subsystem (i.e. period and budget) are chosen poorly with respect to parameters of the tasks in the subsystem the overhead of SIRAP can be well over 100%. The main conclusion we draw from the study is 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 X_i in the subsystem to prevent having high subsystem utilization.

References

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proc. of RTSS'98*, Spain, Dec. 1998.

[2] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *Proc. of EMSOFT'04*, Sep. 2004.

[3] 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, *Proc. of OSPERT'05*, pages 13–19, Spain, Jul. 2005.

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

[5] M. Caccamo and L. Sha. Aperiodic servers with resource constraints. In *Proc. of RTSS'01*, pages 161 – 170, England, Dec. 2001.

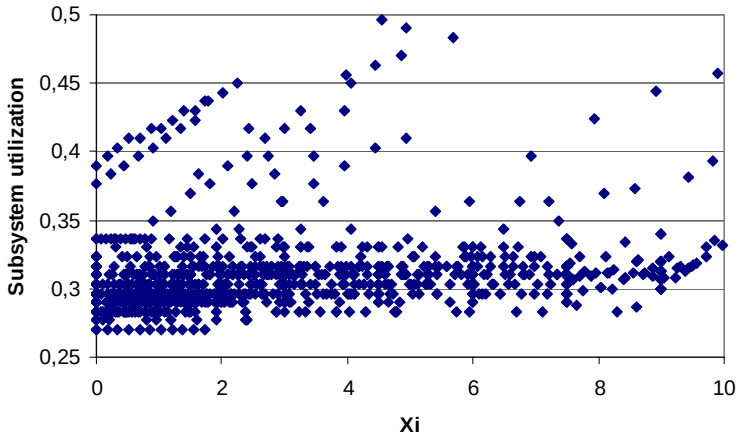

Figure 7. U_Γ as a function of X_i, when only the lowest priority task shares a resource, $\Pi_s = 30$.

[6] R. I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. In *Proc. of RTSS'05*, pages 389–398, USA, Dec. 2005.

[7] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *Proc. of RTSS'06*, pages 257–267, Brazil, Dec. 2005.

[8] Z. Deng and J. W.-S. Liu. Scheduling real-time applications in an open environment. In *Proc.of RTSS'97*, pages 308–319, Dec. 1997.

[9] X. Feng and A. Mok. A model of hierarchical real-time virtual resources. In *Proc. of RTSS'02*, pages 26–35, USA, Dec. 2002.

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

[11] K. H. Kim and M. Naghibzadeh. Prevention of task overruns in real-time non-preemptive multiprogramming systems. In *Proc. of PERFORMANCE '80*, pages 267–276, USA, 1980.

[12] 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.

[13] T.-W. Kuo and C. Li. A fixed-priority-driven open environment for real-time applications. In *Proc. of RTSS'99*, Dec. 1999.

[14] G. Lipari and S. Baruah. Efficient scheduling of real-time multi-task applications in dynamic systems. In *Proc. of RTAS'00*, pages 166–175, Taiwan, May 2000.

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

[16] G. Lipari, J. Carpenter, and S. Baruah. A framework for achieving inter-application isolation in multiprogrammed hard-real-time environments. In *Proc. of RTSS'00*, USA, Dec. 2000.

[17] G. Lipari, P. Gai, M. Trimarchi, G. Guidi, and P. Ancilotti. A hierarchical framework for component-based real-time systems. In *Proc. of CBSE'05*, volume LNCS-3054/2004, pages 209–216. Springer Berlin / Heidelberg, May 2005.

[18] G. Lipari, G. Lamastra, and L. Abeni. Task synchronization' in reservation-based real-time systems. *IEEE Transactions on Computers*, 53(12):1591–1601, Dec. 2004.

[19] S. Matic and T. A. Henzinger. Trading end-to-end latency for composability. In *Proc. of RTSS'05*, pages 99–110, USA, Dec. 2005.

[20] A. Mok, X. Feng, and D. Chen. Resource partition for real-time systems. In *Proc. of RTAS'01*, pages 75–84, May 2001.

[21] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proc. of RTSS'88*, pages 259–269, USA, Dec. 1988.

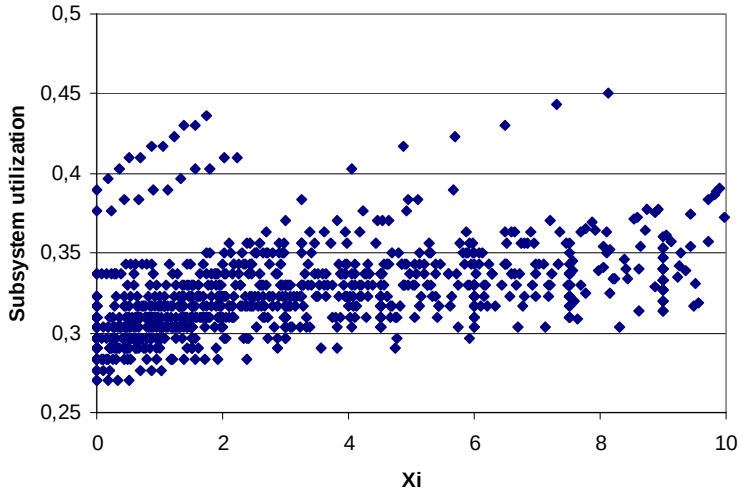

Figure 8. U_Γ **as a function of** X_i**, when only the highest priority task shares a resource,** $\Pi_s = 30$**.**

[22] S. Saewong, R. Rajkumar, J. Lehoczky, and M. Klein. Analysis of hierarchical fixed-priority scheduling. In *Proc. of ECRTS'02*, Austria, Jun. 2002.

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

[24] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proc. of RTSS'03*, Mexico, Dec. 2003.

[25] I. Shin and I. Lee. Compositional real-time scheduling framework. In *Proc. of RTSS'04*, Portugal, Dec. 2004.

[26] M. Spuri. Preemptive and non-preemtive real-time uniprocessor scheduling. Technical Report: RR-2966, INRIA, 1996.

[27] M. Spuri and G. C. Buttazzo. Efficient aperiodic service under earliest deadline scheduling. In *Proc. of RTSS'94*, pages 2–11, Puerto Rico, Dec. 1994.