Efficient Utilization of Resources for Hierarchical Scheduling of Real-Time Systems

Mr.P.Vijaykumar M.E. (Asst.Prof)
Department of Electronics and Communication Engineering
SRM University
Chennai. India
vijay_at23@rediffmail.com

Mr.R.Vigneshwar M.Tech
Department of Electronics and Communication Engineering
SRM University
Chennai. India
vigneshwar87@gmail.com

***Abstract*- The Hierarchical Scheduling Framework (HSF) has been introduced as a design-time framework to enable compositional schedulability analysis of embedded software systems with real-time properties. In this paper, a software system consists of a number of semi-independent components called Subsystems. Subsystems are developed and later integrated to form a system. To support this design process, the proposed methods allow non-intrusive configuration and tuning of subsystem timing-behaviour via subsystem interfaces for selective scheduling parameters. Three overrun methods to handle overrun due to resource sharing between subsystems in the HSF. The analysis is generalized to allow for Rate-Monotonic (RM) scheduling. Also, a further contribution to this is the technique of calculating resource-holding times within the framework under different scheduling algorithms; the resource-holding times being an important parameter in global schedulability analysis.**

***Index Terms*- Hierarchical Scheduling, operation system, real time system, scheduling, resource sharing, synchronization.**

I. INTRODUCTION

The Hierarchical Scheduling Framework (HSF) has been introduced to support hierarchical resource sharing among applications under different scheduling services. The hierarchical scheduling framework can be generally represented as a tree of nodes, where each node represents an application with its own scheduler internal workloads (e.g. threads), and resources are allocated from a parent node to its children nodes.

The HSF provides means of decomposing a complex system into well-defined parts. In essence, the HSF provides a mechanism for timing-predictable composition of course-grained components or *subsystem.* In HSF, a subsystem provides an interface that specifies the timing properties of the subsystem precisely [1]. This means that subsystems can be independently developed and tested, and later assembled without introducing unwanted temporal behaviour. Also, the HSF facilitates reusability of subsystem in timing-critical and resource constrained environments, since the well defined interfaces characterize their computational requirements.

Earlier efforts have been made in supporting compositional subsystem integration in the HSFs, preserving the independently analyzed schedulability of individual subsystems. One common assumption shared by earlier studies is that subsystems are independent. This paper relaxes the assumption by addressing the challenge of enabling efficient compositional integration of independently developed semi-independent subsystems interacting through sharing of mutual exclusion access logical resources. Here, semi-independence means that subsystems are allowed to synchronize by the sharing of logical resources.

To enable sharing of logical resources in HSFs, Davis and Burns proposed a synchronization protocol

implementing the overrun mechanism, allowing the subsystem to overrun to complete the execution of a critical section [2]. Two version of overrun mechanisms were presented in [2], called overrun without payback and overrun with payback, and in the remainder of this paper these overrun mechanism are called Basic Overrun (BO), and Basic Overrun with Payback (PO) respectively. The study presented by Davis and Burns provides schedulability analysis for both overrun mechanisms; however, the schedulability analysis does not allow independent analysis of individual subsystems. Hence the presented schedulability analysis does not naturally support composability of subsystems.

In addition to these overrun mechanism, a new overrun mechanism has been presented, called Enhanced Overrun (EO) that potentially increases schedulability within a subsystem by providing CPU allocation more efficiently.

II. RELATED WORK

A. *Hierarchical Scheduling*

The HSF for real-time systems, originating in open systems [4], has been receiving an increasing research attention. Since Deng and Liu [4] introduced a two-level HSF, its schedulability has been analyzed under fixed-priority global scheduling [5] and under Earliest Deadline First based global scheduling[6],[7].

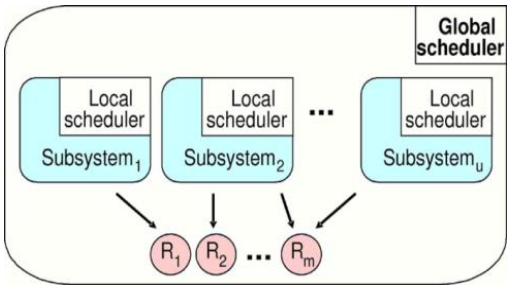

Fig. 1. Two-level HSF with resource sharing.

B. *Resource Sharing*

In many real systems, tasks are semi-independent, interfacing with each other through mutually exclusive resource sharing. Many protocols have been introduced to address the priority inversion problem for semi-independent tasks, including the Priority Inheritance Protocol (PIP) [9], the Priority Ceiling Protocol (PCP) [10], and Stack Resource Policy (SRP) [11]. There have been studies on extending SRP for HSFs, for sharing of logical resources within a subsystem [5], [12] and across the subsystems [2], [13]. David and Burns [2] proposed the Hierarchical Stack Resource Policy (HSRP) supporting sharing of logical resources on the basis of an overrun mechanism. Behnam *et al.* [13] proposed the Subsystem Integration and Resource Allocation Policy (SIRAP) protocol that supports subsystem integration in the presence of shared logical resources, on the basis of skipping. Lipari *et al.* proposed the Band Width Inheritance protocol (BWI) which extends the resource reservation framework to subsystems where tasks can share resources. The BWI approach based on using the Constant Bandwidth Server (CBS) algorithm together with a technique that is derived from the PIP. Particularly, BWI is suitable for systems where the execution time of a task inside a critical section cannot be evaluated.

III. SYSTEM MODEL AND BACKGROUND

A. *Resource Sharing in the HSF*

The HSF has been introduced to support CPU time sharing among applications under different scheduling policies. Here a two level hierarchical scheduling framework is considered, which works as follow: a global scheduler allocates CPU time to subsystems, and a local scheduler subsequently allocates CPU time to its internal tasks.

Having such a HSF also allows for the sharing of logical resources among tasks in a mutually exclusive manner (see fig.1). Specifically, tasks can share local logical resources within a subsystem as well as global logical resources across subsystems. This work focuses around mechanism for sharing of global logical resources in a HSF, while local logical resources can be supported by traditional synchronization protocols such as SRP.

B. *Virtual Process Models*

The notion of real-time virtual processor model was introduced by Mok *et al.* [8] to characterize the CPU allocations that a parent node provides to a child node in a HSF. The CPU supply of a virtual processor model refers to the amount of CPU allocations that the virtual processor model can provide. The supply bound function of a virtual processor model calculates its minimum possible CPU supply for any given time of length t

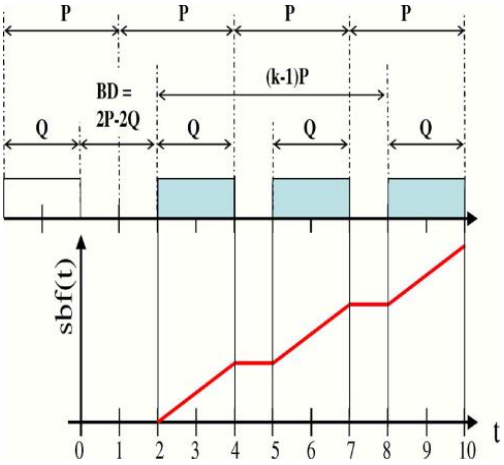

Fig.2. The supply bound function of a periodic virtual processor model

The periodic virtual processor model Γ(P,Q) was proposed by Shin and Lee [1] to characterize periodic resource allocations where P is a period (P > 0) and Q is a periodic allocation time (0 < Q ≤ P).

The supply bound function sbf_{Γ} (t) of the periodic virtual process model Γ (P, Q) was given in [1] to compute the minimum resource supply during an interval of length t

$$sbf(t) = \begin{cases} t - (k+1)(P-Q), & if\ t \in W \\ (k-1)Q & , otherwise \end{cases} \quad (1)$$

Where k=max ([(t-(P-Q))/P], 1) and W denotes an interval [(k+1) P-2Q, (k+1) P-Q]. Note that an interval of length *t* may not begin synchronously with the beginning of period P, as shown in fig.2; the interval of length *t* can start in the middle of the period of a periodic virtual processor model Γ (P, Q). BD represent the longest possible *blackout duration* during which the periodic virtual processor model may provide no resource allocation at all.

***C. Stack Resource Policy* (*SRP*)**

To be able to use SRP in the HSF, its associated terms are extended as follows:

- *Pre-emption level*. Each task τ_i has a pre-emption level equal to $\pi_i=1/D_i$, where D_i is the relative deadline of the task. Similarly each subsystem S_s has an associated pre-emption level equal to $\Pi_s=1/P_s$, where P_s is the subsystems per-period deadline.
- *Resource Ceiling*. Each globally shared resources R_j is associated with two types of resource ceilings; one internal resource ceiling for local scheduling rc_j = max { π_i | τ_i accesses R_j} and one external resource ceiling for global scheduling.
- *System/Subsystem ceilings* System/Subsystem ceilings are dynamic parameters that change

during runtime. The system/subsystem ceiling is equal to the currently locked highest external/internal resource ceiling in the system/subsystem.

D. *System Model*

In this paper a periodic task model τ_i (T_i, C_i, D_i,{$c_{i,j}$}) is considered, where T_i, C_i, D_i , represent the task's period, worst-case execution time (WCET) and relative deadline, respectively, where $D_i \leq T_i$, and {$c_{i,j}$} is the set of WCETs within critical sections associated with task τ_i .

Looking at a shared resources R_j, the resource holding time $h_{j,i}$ of a task τ_i is defined as the time given by the task's maximum execution time inside a critical section plus the interference of higher priority tasks having pre-emption level greater than the internal ceiling of the locked resource.

A subsystem $S_s \in S$, where S is the whole system of subsystems, is characterized by a task set τ_s that contain n_s task and a set of internal resource ceiling RC_s inherent from internal tasks using the globally shared resources. Each subsystem S_s is assumed to have an RM local scheduler, and the subsystems are scheduled according to RM on a global level. The collective CPU resource requirements by each subsystem S_s is characterized by its interface defined as (P_s, Q_s, H_s), where P_s is the subsystems period, Q_s is its execution requirement budget, and H_s is the subsystem maximum resource holding time i.e., H_s = max {$hj_{,i}$ | $\tau_i \in \tau_s$ accesses R_j }.

IV. SCHEDULABILITY ANALYSIS

This section presents the schedulability analysis of the HSF, starting with local schedulability analysis needed to calculate subsystem interfaces, followed by global schedulability analysis. The analysis presented assumes that SRP is used for synchronization on the local level.

A. *Local Schedulability Analysis*

Let request bound function RBF (i, t) of a task τ_i for RM scheduler is

$$RBF(i, t) = C_i + \Sigma_{\tau k \in HP(i)} [t / P_k].C_k \quad (2)$$

Where HP (i) is the set of tasks with priorities higher than that of τ_i. Note that *t* can be selected within a finite set of scheduling points.

B. *Global Schedulability Analysis*

Under global RM scheduling, the subsystem load bound function is as follows [on the basis of a similar reasoning of [4]]:

$$LBF(t) = RBF(t) + B_s \quad (3)$$

Where

$$RBF(t) = Q_s + \Sigma_{S \in HPS(s)} [t / P_k].Q_k \quad (4)$$

Where HPS(s) is the set of subsystem with priority higher than that of S_s. Let B_s denote the maximum blocking imposed to a subsystem S_s, when it is blocked by lower-priority subsystems

B_s = max {H_j | $S_j \in LPS(S)$ }

Where LPS (S_s) is the set of subsystems with priority lower than that of S_s.

V. OVERRUN MECHANISM

This section explains three overrun mechanism that can be used to handle budget expiry during a critical section in the HSF. Consider a global scheduler that schedules subsystems according to their periodic interfaces (P_s, Q_s, and Hs). The subsystem budget Q_s, is said to expire at the point when one or

more internal tasks have executed a total of $Q_{s,}$ time units within the subsystems period $P_{s.}$ Once the budget is expired, no new task within the same subsystem can initiate its execution until the subsystem's budget is replenished. The replenishment takes place in the beginning of each subsystem period, where the budget is replenished to a value of Qs.

Budget expiration may cause a problem if it happens while a job $J_{i\ of}$ a subsystem $S_{s\ is}$ executing within a critical section of a global shared resource R_j . If another job $J_{i,}$ belonging to another subsystem, is waiting for the same resource $R_{j,}$ this job must still wait until S_s is replenished again so J_i can continue to execute and finally release the lock on resource R_j . This waiting time exposed to J_k can be potentially very long, causing J_k to miss its deadline.

In this paper, an overrun mechanism is considered as follows; when the budget of subsystem S_s expires and S_s has a job J_i that is still locking a globally shared resource, job J_i continues its execution until it release the locked resource. The extra time that J_i needs to execute after the budget of S_s expires is denoted as *overrun time* Θ. The maximum Θ occurs when J_i locks a resource that gives the longest resource holding time just before the budget of S_s expires.

Here two versions of overrun mechanism [2] are considered;

1) The overrun mechanism with payback, introduced as PO and later EO: whenever overrun happens, the subsystem S_s payback Θ in its next execution instant, i.e., the subsystem budget Q_s will be decreased by Θ for the subsystem's execution instant following the overrun.
2) The overrun mechanism without payback, introduced as BO: in this version of the overrun mechanism, no further actions will be taken after the event of an overrun.

A. *Basic Overrun Mechanism*

a) PO-Basic Overrun with Payback: First, the request bound function of a subsystem with the basic overrun mechanism with payback is extended. Looking at the PO mechanism in a subsystem S_s, the maximum contribution on RBF (t) for RM scheduling is H_s. When S_s overrun with its maximum, which is H_s, the subsystem's resource demand within the subsystem period P_s will be increased to $Q_s + H_s$. Following this budget the next period will be decreased to $Q_s - H_s$ due to the payback mechanism. Then, suppose that the subsystem overruns again. Now during the next subsystem period, the subsystem's resource demand will be $Q_s - H_s + H_s = Q_s$. Here it is easy to observe that the subsystem's resource demand will be at most $kQ_s + H_s$ during k subsystem periods. When using a global RM scheduler, the request bound function $RBF^o(t)$ is

$$RBF^o(t) = (Q^o{}_s + H_s) + \Sigma_{Sk \in HPS(S)} ([t/P_k]\ (Q^o{}_k) + H_k) \quad (4)$$

b) *BO-Basic overrun without payback*: This version of overrun does not payback the budget after overrun happens. This means that the subsystem resource demands within period of P_s can be up to $Q_s + H_s$ for all periods considering that the maximum overrun will happen every period, which is the worst case scenario. Then, for global RM scheduler, the request bound function $RBF^{\#}(t)$ is

$$RBF^{\#}(t) = (Q^{\#}{}_s + H_s) + \Sigma_{Sk \in HPS(S)}([t/P_k]\ (Q^{\#}{}_k + H_k) \quad (5)$$

B. *Enhanced Overrun Mechanism*

The PO mechanism works with a modified request bound function $RBF^{o}(t)$ that is less efficient in terms of CPU resource usage compared with the original RBF (t). While for the BO mechanism, the request bound function will be increased by $Q_s + H_s$ in all periods which may require more resources as well. In the following, an enhanced overrun mechanism (EO) is proposed. This new overrun mechanism makes it possible to improve $SBF^{o}(t)$ and at the same time the request bound function will be $Q_s + H_s$ for the first instance and then only Q_s for the following periods when applying global schedulability analysis.

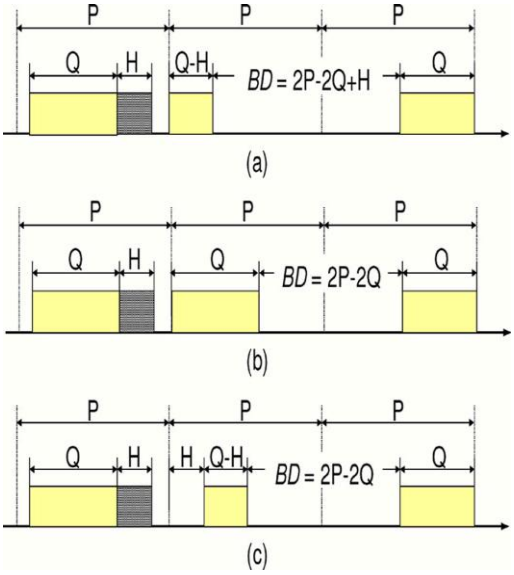

Fig.3. Basic and enhanced overrun mechanisms.
(a) Basic overrun mechanism with payback (PO)
(b)Basic overrun mechanism without payback (BO)
(c)Enhanced overrun mechanism (EO)

The EO mechanism is based on imposing an offset (delaying the budget replenishment of subsystem) equal to the amount of an overrun ($\Theta_s = H_s$) to the execution instant that follows a subsystem overrun. As shown in Fig.3(c), the execution of the subsystem will be delayed by Θ_s after a new period followed by overrun even if that subsystem has the highest priority at that time. By this the maximum BD will be decreased by 2(P-Q) compared with PO.

Under RM scheduler, the offset imposed by the EO mechanism for each subsystem S_s can be modelled as a release jitter J_s with the range of $[0, H_s]$ so $J_s = H_s$. The upper bound function of $RBF^{*}(t)$ is

$$RBF^{*}(t) = (Q^{*}_s + H_s) + \Sigma_{Sk \in HPS(S)}([t + J_k/P_k] (Q^{*}_k)+ H_k) \quad (6)$$

B. System Level Comparison

In doing a comparison among the three approaches, *System load* is defined as a quantitative measure to represent the minimum amount of CPU allocations necessary to guarantee the schedulability of the system *S*. Then, the impact of each overrun mechanism on the system load can be investigated respectively. When using RM as a local scheduler, the system load is computed as follows:

$$Load_{sys} = \max (RBF(t)) / t \quad (7)$$

VI. COMPARISON OF OVERRUN MECHANISMS

The comparison between the three overrun mechanisms in terms of request bound function is shown below:

1) **PO versus BO**: comparing (4) and (5), it is show that $RBF^{o}(t) > RBF^{\#}(t)$ for $0 \leq t \leq P_s$. The reason is that the interface from other higher priority tasks is always $Q_k + H_k$ for both cases from $Q^{o}_s > Q^{\#}_s$. If $t > P_s$ then the mechanism that requires a lower request bound function is different depending on the subsystem parameters. It can be concluded

that if the subsystem periods of all subsystem periods of all subsystems are equal, then the BO mechanism will require less system load than using the PO mechanism.

2) **BO versus EO**: comparing (5) and (6), it is show that $RBF^{*}(t) \geq RBF^{\#}(t)$ for $0 \leq t \leq P_s$. If $t > P_s$ then finding the best mechanism that requires the least system load depends on the system parameters.

3) **PO versus EO**: comparing (4) and (6), it can be concluded that $RBF^{o}(t) > RBF^{*}(t)$ when t is in the range $kP_s - H_s \geq t < kP_s$ and $RBF^{o}(t) \geq RBF^{*}(t)$ when t is in $(k-1)P_s \geq t < (k)P_s - H_s$, where k is an integer value greater and $k>0$.

The following example show some of the cases discussed above:

Example 1: Suppose that a system S consists of three subsystems with parameters as shown below:

Subsystem	P_s	Q^{o}_s	$Q^{\#}_s$	Q^{*}_s	H_s
S1	0	4.7	4	4	2
S2	5	1.5	1.2	1.3	4
S3	20	3.8	3.4	3.6	4

The global scheduler is Rate Monotonic. Using the PO mechanism load $_{sys}$ = 0.79 at t= 20, using the BO mechanism load $_{sys}$= 0.81 at t = 20, and for the EO mechanism load$_{sys}$ = 0.69 at t = 15.

Example 2: Suppose that a system S consists of three subsystems with parameters as shown in the next table:

The global scheduler is Rate Monotonic. Using the PO mechanism load $_{sys}$ = 0.56 at t= 15, using the BO mechanism load $_{sys}$= 0.6 at t=20, and for the EO mechanism load $_{sys}$ = 0.65 at t=13.

Subsystem	P_s	Q^{o}_s	$Q^{\#}_s$	Q^{*}_s	H_s
S1	12	2	1.5	2	1
S2	15	3.1	2.1	3	2
S3	20	5.1	4.8	5	3

VII. COMPUTING RESOURCE HOLDING TIME

This section explain how to compute the resource holding time $h_{j,i}$, a very important parameter in the global analysis. The resource holding time is the time given by the tasks maximum execution time inside a critical section plus the interference of higher priority tasks having pre-emption level greater than the internal ceiling of the locked resource. This means that the internal resource ceiling rc_j is one of the parameters that can have great effect on resource holding times of the globally shared resources. The resource holding time $h_{j,i}$, of a shared resource R_j accessed by τ_i is the smallest positive time t^{*} such that $w_j(t^{*}) = t^{*}$, with w_j computed as follows:

$$W_j(t) = CX_j + \Sigma_{\tau k \in U} [i/T_k].\ C_k \qquad (8)$$

Where $CX_{j,i} = \max\{c_{i,j}\}$ is the maximum execution time of task τ_i inside critical section of the resource and U is the set of tasks.

Now for hierarchical scheduling framework that uses the overrun mechanism, the following equation shows how to evaluate resource holding time for a task τ_i that access the resource R_j

$$h_{j,i} = cx_{j,i} + \Sigma_{\tau k \in U} ck \qquad (9)$$

the difference between (7) and (8) is all the tasks that can pre-empt inside the critical section are assumed to be executed only once. The reason for why it is safe to assume only one execution of each pre-empting task inside the critical section is given in the following lemma, showing that if a task executes more than one

time inside the critical section, the subsystem will become unschedulable.

Lemma A: For a subsystem that uses an overrun mechanism to arbitrate access to a global shared resource under the periodic virtual processor model, each task that is allowed to pre-empt the execution of another task currently inside the critical section of a globally shared resource can, in the worst case, only execute once independent if the local scheduler is either fixed priority scheduler or dynamic priority scheduler.

Proof: This lemma will be proved by contradiction to show that if more than one job of a task pre-empts a critical section then the system utilization will be greater than one. The following cases are considered in the proof:

1) $P_s < T_m$ (where $T_m = min\ (T_i)$ for all i = 1.....n), if the task having period T_m executes 2 or more times inside the critical section, this means that the resource will be locker during this period, i.e., $h_{j,i} > T_m$ then $h_{j,i} > P_s$ which in turn means that the CPU utilization required by the subsystem S_s will be $U_s = (Q_s + h_{j,i})/ P_s > 1$.
2) If $P_s \geq T_m$, sbf (t) should provide at least C_m at time $t = T_m$ to ensure the schedulability test for the RM scheduler. Note that sbf (t)=0 during t [0, $2P_s - 2Q_s$] so, $2P_s - 2Q_s + C_m \leq T_m$ which means $Q_s^{min} \geq P_s - T_m / 2 + C_m / 2$. Then, the minimum subsystem budget is

$$Q_s^{min} = P_s - T_m / 2 + C_m / 2 \qquad (10)$$

Let us define G_s as the maximum time in which a subsystem may not get any budget within the subsystem period P_s because of pre-emptions from other higher priority subsystems, then $G_s = P_s - Q_s$ (see fig.) and substituting Q_s by the minimum subsystem budget in (10)

$$G_s = (T_m - C_m) / 2 \qquad (11)$$

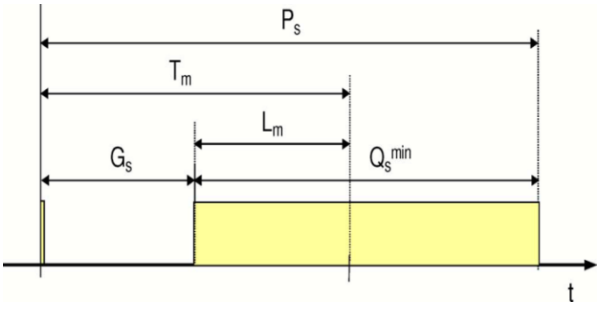

Fig.4. Resource holding times for the case

The maximum number of activation of the τ_m within P_s while a lower priority task accessing a global shared resource, will happen when τ_m is release at the beginning of the subsystem period just after the lower priority task has locked global shared resource. Now, let's assume that τ_m will execute two times while the global shared resource is locked, then the subsystem budget given to the subsystem within the first period of τ_m should be low enough such that the shared resource will not be released before the second activation of τ_m. Let us define L_m as the minimum subsystem budget that will be supplied to the subsystem within the first period of T_m, $L_m = T_m - G_s$ and from (11)

$$L_m = (T_m + G_s)/ 2 \qquad (12)$$

From Lemma A, it can be concluded that all tasks that can pre-empt the execution of a critical section should do so maximum one time in order to keep the utilization of a subsystem less than one. If a task pre-empts the execution of critical section more than one time then it will be seen from (9). This proves the correctness of (9) which is based on the assumption that all task can interface only once as a worst case while a task is in the critical section of the resource R_j.

VIII. CONCLUSION

This paper presents three different overrun mechanisms that all can handle the problem of sharing of logical resources in a hierarchical scheduling framework while at the same time supporting independent subsystem development (open environments). Compared to previous work [3], here results have been generalized by also allowing for the RM scheduling algorithm for both local and global schedulers, which is suitable for usage in open environments. In addition, a third overrun mechanism, basic overrun without payback (BO), is included in the comparison between the overrun mechanisms. The results from this comparison show that it is not trivial to evaluate, in the general case, which overrun method that is better than the other, as their impact on the CPU utilization is highly dependent on global system parameters such as subsystem periods and budgets. Finally, the calculation of re-source holding times when using the periodic virtual processor model with RM scheduling algorithms is presented, as the resource holding time is a very important parameter in the global schedulability analysis.

Future work includes comparing the enhanced overrun mechanism (EO) with other synchronization mechanisms such as BWI, the BROE server and SIRAP. In addition the three overrun mechanism and comparing the implementation overhead of each mechanism is important. Finally, as the global schedulability analysis gives an upper bound for EO, it will be interesting to find an exact or less pessimistic schedulability analysis.

REFERENCES

[1] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Proc. 24th IEEE Int. Real-Time Syst. Symp. (RTSS'03)*, Dec. 2003, pp. 2–13.

[2] R. I. Davis and A. Burns, "Resource sharing in hierarchical fixed priority pre-emptive systems," in *Proc. 27th IEEE Int. Real-Time Syst.Symp. (RTSS'06)*, Dec. 2006, pp. 389–398.

[3] M. Behnam, I. Shin, T. Nolte, and M. Nolin, "Scheduling of semi-independent real-time components: Overrun methods and resource holding times," in *Proc. 13th IEEE Int. Conf. Emerging Technol. Factory Autom. (ETFA'08)*, Sep. 2008, pp. 575–582.

[4] Z. Deng and J.-S. Liu, "Scheduling real-time applications in an open environment," in *Proc. 18th IEEE Int. Real-Time Syst. Symp. (RTSS'97)*, Dec. 1997, pp. 308–319.

[5] T.-W.Kuo and C.-H. Li, "A fixed-priority-driven open environment for real-time applications," in *Proc. 20th IEEE Int. Real-Time Syst. Symp. (RTSS'99)*, Dec. 1999, pp. 256–267.

[6] G. Lipari and S. K. Baruah, "Efficient scheduling of real-time multitask applications in dynamic systems," in *Proc. 6th IEEE Real-Time Technol. Appl. Symp. (RTAS'00)*, May–Jun. 2000, pp. 166–175.

[7] G. Lipari, J. Carpenter, and S. Baruah, "A framework for achieving inter-application isolation in multiprogrammed hard-real-time environments," in *Proc. 21th IEEE Int. Real-Time Syst. Symp. (RTSS'00)*, Dec, 2000, pp. 217–226.

[8] A. Mok, X. Feng, and D. Chen, "Resource partition for real-time systems," in *Proc. IEEE Real-Time Technol. Appl. Symp. (RTAS'01)*, May 2001, pp. 75–84.

[9] L. Sha, J. P. Lehoczky, and R. Rajkumar, "Task scheduling in distributed real-time systems," in *Proc. Int. Conf. Ind. Electron., Control, Instrum. (IECON'87)*, Nov. 1987, pp. 909–916.

[10] R. Rajkumar, L. Sha, and J. P. Lehoczky, "Real-time synchronization protocols for multiprocessors," in *Proc. 9th IEEE Int. Real-Time Syst. Symp. (RTSS'88)*, Dec. 1988, pp. 259–269.

[11] T. P. Baker, "Stack-based scheduling of realtime processes," *Real-Time Syst.*, vol. 3, no. 1, pp. 67–99, Mar. 1991.

[12] L. Almeida and P. Pedreiras, "Scheduling within temporal partitions: Response-time analysis and server design," in *Proc. 4th ACMInt. Conf. Embedded Softw. (EMSOFT '04)*, Sep. 2004, pp. 95–103.

[13] M. Behnam, I. Shin, T. Nolte, and M. Nolin, "SIRAP: A synchronization protocol for hierarchical resource sharing in real-time open systems," in *Proc. 7th ACM and IEEE Int. Conf. Embedded Softw. (EMSOFT'07)*, Oct. 2007, pp. 279–288.