Opaque analysis for resource sharing in compositional real-time systems

Martijn M. H. P. van den Heuvel[†], Moris Behnam[‡], Reinder J. Bril[†], Johan J. Lukkien[†] and Thomas Nolte[‡]

[†]Technische Universiteit Eindhoven, Den Dolech 2, 5612 AZ Eindhoven, The Netherlands

[‡]Mälardalen Real-Time Research Centre (MRTC), P.O. Box 883, SE-721 23 Västerås, Sweden

Abstract—In this paper we propose opaque analysis methods to integrate dependent real-time components into hierarchical fixed-priority scheduled systems. To arbitrate mutually exclusive resource access between components, we consider two existing protocols: HSRP - comprising overrun with and without payback - and SIRAP. An opaque analysis allows to postpone the choice of a synchronization protocol until component integration time.

First, we identify the sources of pessimism in the existing analysis techniques and we conclude that both protocols assume different models in their local analysis. In particular, the compositional analysis for overrun with payback (OWP) is not opaque and is pessimistic. The latter makes OWP expensive compared to its counter part without a payback mechanism (ONP). This paper presents an opaque and less pessimistic OWP analysis.

Secondly, SIRAP requires more timing information to perform a task-level schedulability analysis. In many practical situations, however, detailed timing characteristics of tasks are hard to obtain. We introduce an opaque analysis for SIRAP using the analysis of ONP to reduce the required timing information during the local analysis. We show that the analysis for ONP cannot deem systems schedulable which are infeasible with SIRAP. The SIRAP analysis may therefore reduce the required system resources of a component by sacrificing the choice for an arbitrary synchronization protocol at system integration time[1].

I. INTRODUCTION

The increasing complexity of real-time systems demands a decoupling of (*i*) development and analysis of individual components and (*ii*) integration of components on a shared platform, including analysis at the system level. Hierarchical scheduling frameworks (HSFs) have been extensively investigated as a paradigm for facilitating this decoupling [1]. A component that is validated to meet its timing constraints when executing in isolation will continue meeting its timing constraints after integration (or admission) on a shared uni-processor platform. The HSF therefore provides a promising solution for current industrial standards, e.g. the AUTomotive Open System ARchitecture (AUTOSAR) which specifies that an underlying operating system should prevent timing faults in any component to propagate to other components on the same processor. The HSF provides temporal isolation between components by allocating a *budget* to each component.

An HSF without further resource sharing is unrealistic, however, since components may for example use operating system services, memory mapped devices and shared communication devices requiring mutually exclusive access. An HSF with such support makes it possible to share logical resources between arbitrary tasks, which are located in arbitrary components, in a mutually exclusive manner. A resource that is used in more than one component is denoted as a *global shared resource*. A resource that is only shared by tasks within a single component is a *local shared resource*. If a task that accesses a global shared resource is suspended during its execution due to the exhaustion of its budget, excessive blocking periods can occur which may hamper the correct timeliness of other components [2].

To accommodate resource sharing between components, two synchronization protocols [3], [4] have been proposed based on the Stack Resource Policy (SRP) [5] for two-level fixed-priority-scheduled HSFs. Each of these protocols describes a run-time mechanism to handle the depletion of a component's budget during global resource access. In short, two general mechanisms are proposed: (*i*) *self-blocking* when the remaining budget is insufficient to complete a critical section - called SIRAP [4] or (*ii*) *overrun* the budget until the critical section ends - called HSRP [3]. HSRP comes in two flavors: overrun with payback (OWP) and overrun without payback (ONP). The term *without payback* means that the additional amount of budget consumed during an overrun does not have to be paid back during the next budget period.

In practical situations, many critical-section lengths are unknown, e.g. tasks may execute many critical sections which are hard to analyze individually (as with SIRAP). It can be easier to determine the worst-case critical-section length per shared resource, as used in the traditional response-time analysis. Upon component integration, we apply a global schedulability test based on a choice for a global synchronization protocol. We therefore call the local analysis of a component opaque, if

1) it does not include any timing information about global resource arbitration;
2) it allows to post-pone the classification of shared resources into global and local until integration time.

These criteria are important in open multi-vendor environments, since it hides which local analysis method is applied. Hence, an opaque analysis enables an incremental analysis, since it separates concerns of local and global scheduling.

Using the notion of opaque analysis methods, we can apply the existing analysis for ONP to integrate such a component into the HSF. The reason for this is that the local analysis of ONP is compliant to the widely accepted synchronization protocols PCP [6] and SRP [5]. Surprisingly and contrary to ONP, the current OWP analysis does not support opaque

analysis, because it modifies the abstracted processor supply to a component. For the same reason SIRAP has no opaque analysis.

Contributions: The contributions of this paper are five fold. First, we show that ONP can be used as a valid upper bound for SIRAP to support components with an opaque analysis. This means that ONP provides a valid analysis technique for resources arbitrated by a run-time implementation of SIRAP. Secondly, we reduce the pessimism of OWP and show that OWP is in most cases better than ONP. Thirdly, we show that our improved analysis for OWP supports the integration of components with an opaque analysis into the HSF. Fourthly, we compare the abstraction overheads of the different analysis techniques for SIRAP, OWP and ONP. Finally, we derive revised guidelines for selecting a synchronization protocol.

II. RELATED WORK

Deng and Liu [7] proposed a two-level HSF for open systems, where components may be independently developed and validated. The corresponding schedulability analysis have been presented in [8] for fixed-priority preemptive scheduling (FPPS) and in [9] for earliest-deadline-first (EDF) global schedulers. For global resource sharing in HSFs, three protocols have recently been presented to prevent budget depletion during resource access, i.e. HSRP [3], SIRAP [4] and BROE [10]. Unlike HSRP and SIRAP's analysis, however, the global schedulability analysis of BROE is limited to EDF and cannot be generalized to include other scheduling policies.

The overrun mechanism (with payback) was first introduced in the context of aperiodic servers in [2]. This mechanism was later re-used in HSRP in the context of two-level HSFs by Davis and Burns [3] and complemented with a variant *without* payback. Although the analysis presented in [3] does not integrate in HSFs due to the lacking support for independent analysis of components, this limitation is lifted in [11].

The idea of self-blocking has also been considered in different contexts, e.g. for supporting soft real-time tasks [12] and for a zone-based protocol in a pfair-scheduling environment [13]. SIRAP [4] uses self-blocking for hard real-time tasks in HSFs on a single processor and its associated analysis supports composability. In [14] the original SIRAP analysis [4] has been significantly improved when arbitrating multiple shared resources. We will show that the strength of SIRAP's analysis comes from its detailed system model, making it difficult to analyze components with opaque timing characteristics.

The original SIRAP [4] and HSRP [11] analyses have been analytically compared with respect to their impact on the system load for various component parameters [15]. The performance of each protocol heavily depends on the chosen system parameters. Moreover, these results suggest that HSRP's overrun mechanism with payback (OWP) is hardly beneficial compared to overrun without payback (ONP). This observation is contradictory with the recommendations from Davis and Burns [3]. Our new analysis methods make the results in [15] obsolete and we will provide new guidelines to select a synchronization protocol in two-level FPPS-based HSFs.

III. REAL-TIME SCHEDULING MODEL

We consider a two-level FPPS-based HSF, following the periodic resource model [1] to guarantee processor allocations to components. However, we believe that our results can be straightforwardly extended to other scheduling policies. We use SRP-based synchronization to arbitrate mutually exclusive access to global shared resources.

A. Component model

A system contains a single processor, a set $\mathcal{C}$ of N components C_1, ..., C_N for which we assume a periodic resource model [1], and a set $\mathcal{R}$ of M global logical resources R_1, ..., R_M. Each component C_s has a dedicated budget which specifies its periodically guaranteed fraction of the processor.

The timing characteristics of a component C_s are specified by means of a triple $\Gamma_s(P_s, Q_s, \mathcal{X}_s)$, where $P_s \in \mathbb{R}^+$ denotes its period, $Q_s \in \mathbb{R}^+$ its budget, and $\mathcal{X}_s$ the set of maximum access times to global resources. The maximum value in $\mathcal{X}_s$ is denoted by X_s, where $0 < X_s \leq P_s$. The set $\mathcal{R}_s$ denotes the subset of global resources accessed by component C_s. The maximum time that a component C_s executes while accessing resource $R_l \in \mathcal{R}_s$ is denoted by X_{sl}, where $X_{sl} \in \mathbb{R}^+ \cup \{0\}$ and $X_{sl} > 0 \Leftrightarrow R_l \in \mathcal{R}_s$.

Processor supply: The *processor supply* refers to the amount of processor allocation that a component C_s can provide to its workload. The supply bound function $\mathtt{sbf}_{\Gamma_s}(t)$ of the periodic resource model $\Gamma_s(P_s, Q_s)$, that computes the minimum supply for any interval of length t, is given by [1]:

$$\mathtt{sbf}_{\Gamma_s}(t) = \max \left\{ \begin{array}{l} 0, \\ t - (k(t) + 1)(P_s - Q_s), \\ (k(t) - 1)Q_s \end{array} \right\}, \quad (1)$$

where $k(t) = \left\lceil \frac{t - (P_s - Q_s)}{P_s} \right\rceil$. The longest interval a component may receive no processor supply is named the *blackout duration*, BD_s, i.e. $BD_s = 2(P_s - Q_s)$.

B. Task model

Each component C_s contains a set $\mathcal{T}_s$ of n_s sporadic tasks τ_1, ..., τ_{n_s}. The timing characteristics of a task $\tau_{si} \in \mathcal{T}_s$ are specified by means of a triple (T_{si}, C_{si}, D_{si}), where $T_{si} \in \mathbb{R}^+$ denotes its minimum inter-arrival time, $C_{si} \in \mathbb{R}^+$ its worst-case computation time, $D_{si} \in \mathbb{R}^+$ its (relative) deadline, where $0 < C_{si} \leq D_{si} \leq T_{si}$. We assume that period P_s of component C_s is selected such that $2P_s \leq T_{si}(\forall \tau_{si} \in \mathcal{T}_s)$, because this efficiently assigns a budget to component C_s [1]. For notational convenience, tasks (and components) are given in priority order, i.e. τ_{s1} has the highest priority and τ_{sn_s} has the lowest priority.

The worst-case computation time of task τ_{si} within a critical section accessing global resource R_l is denoted h_{sil}, where $h_{sil} \in \mathbb{R}^+ \cup \{0\}$, $C_{si} \geq h_{sil}$ and $h_{sil} > 0 \Leftrightarrow R_l \in \mathcal{R}_s$.

C. Synchronization protocol

This paper focuses on arbitrating *global* shared resources using SRP. Traditional protocols such as PCP [6] and SRP [5] can be used for *local* resource sharing in HSFs [16]. To be able to use SRP in an HSF for synchronizing global resources, its associated ceiling terms need to be extended.

1) Resource ceilings: With every global resource R_l two types of resource ceilings are associated; a *global* resource ceiling RC_l for global scheduling and a *local* resource ceiling rc_{sl} for local scheduling. These ceilings are statically calculated and are defined as the highest priority of any component or task sharing resource R_l. According to SRP, these ceilings are:

$$RC_l = \min(N, \min\{s \mid R_l \in \mathcal{R}_s\}), \quad (2)$$
$$rc_{sl} = \min(n_s, \min\{i \mid h_{sil} > 0\}). \quad (3)$$

The outermost $\min$ in (2) and (3) define RC_l and rc_{sl} in those situations where no component or task uses R_l.

The local resource ceiling rc_{sl} can be used to trade-off preemptiveness against *resource holding times* [17], i.e. X_{sil} of a task τ_{si} to a resource R_l. From [11], [15] we know:

Lemma 1: Given $2P_s \le T_{si}(\forall \tau_{si} \in \mathcal{T}_s)$, all tasks τ_{sj} that are allowed to preempt a critical section accessing resource R_l, i.e. $j < rc_{sl}$, can preempt at most once during an access to a global shared resource R_l by task τ_{si}.

This makes it possible to compute the *resource holding time*, X_{sil} of task τ_{si} to resource R_l as follows [15]:

$$X_{sil} = h_{sil} + \sum_{1 \le j < rc_{sl}} C_{sj}, \quad (4)$$

and the maximum resource holding time within a component C_s is computed as $X_{sl} = \max\{X_{sil} \mid 1 \le i \le n_s\}$.

2) System and component ceilings: These ceilings are dynamic parameters that change during execution. The system ceiling is equal to the highest global resource ceiling of a currently locked resource in the system. Similarly, the component ceiling is equal to the highest local resource ceiling of a currently locked resource within a component. Under SRP a task can only preempt the currently executing task if its priority is higher than its component ceiling. A similar condition for preemption holds for components.

IV. COMPOSITIONAL ANALYSIS FOR FIXED-PRIORITY SCHEDULING

This section first recapitulates the existing analysis for ONP, OWP, and SIRAP. Next, we will show that SIRAP strictly dominates ONP. In Section VI we will make use of this property to define an opaque analysis for all three protocols.

A. Global schedulability analysis

For global FPPS of components the following sufficient schedulability condition holds:

$$\forall 1 \le s \le N : \exists t \in (0, P_s] : \text{RBF}(t, s) \le t, \quad (5)$$

where $\text{RBF}(t, s)$ denotes the worst-case cumulative processor request of C_s for a time interval of length t. The function $\text{RBF}(t, s)$ depends on the chosen global synchronization protocol. We therefore assume that *during component-integration time* the synchronization protocol is known. For SIRAP and ONP, the $\text{RBF}(t, s)$ is defined as follows:

$$\text{RBF}(t, s) = B_s + \sum_{1 \le r \le s} \left\lceil \frac{t}{P_r} \right\rceil (Q_r + O_r). \quad (6)$$

A component C_r using ONP demands more resources in its worst-case scenario [11], i.e. the overrun budget O_r is:

$$O_r = \begin{cases} X_r & \text{for ONP and OWP} \\ 0 & \text{for SIRAP.} \end{cases} \quad (7)$$

For OWP, the $\text{RBF}(t, s)$ is slightly modified (using the same definition for O_r):

$$\text{RBF}(t, s) = B_s + \sum_{1 \le r \le s} \left(\left\lceil \frac{t}{P_r} \right\rceil Q_r \right) + O_r. \quad (8)$$

The blocking term, B_s, is defined according to [5]:

$$B_s = \max(0, \max\{X_{ul} \mid s < u \land R_l \in \mathcal{R}_u \land RC_l \le s\}). \quad (9)$$

We use the outermost $\max$ in (9) to define B_s in those situations where no shared resources are used.

B. Local schedulability analysis

By filling in task characteristics in the demand bound RBF of (5) and replacing the right-hand side by (1), i.e. replace t for $\text{sbf}_{\Gamma_s}(t)$, the same schedulability analysis holds for tasks executing within a component as for components at the global level. For local FPPS of tasks the following sufficient schedulability condition holds:

$$\forall 1 \le i \le n_s : \exists t \in (0, D_{si}] : \text{rbf}_s(t, i) \le \text{sbf}_{\Gamma_s}(t), \quad (10)$$

where $\text{rbf}_s(t, i)$ denotes the worst-case cumulative processor request of τ_{si} for a time interval of length t. For ONP and OWP, the $\text{rbf}_s(t, i)$ is fully compliant to the schedulability analysis for task sets on a dedicated unit-speed processor, i.e.

$$\text{rbf}_s(t, i) = b_{si} + \sum_{1 \le j \le i} \left\lceil \frac{t}{T_{sj}} \right\rceil C_{sj}. \quad (11)$$

The blocking term, b_{si}, is defined according to [5]:

$$b_{si} = \max(0, \max\{h_{sjl} \mid i < j \land h_{sjl} > 0 \land rc_{sl} \le i\}). \quad (12)$$

The outermost $\max$ in (12) defines b_{si} also in those situations where no shared resources are used within a component.

According to the current analysis for OWP, however, it is required to modify the $\text{sbf}(t)$ compared to the definition given in (1), see [11]. In Section V, we eliminate this pessimism.

A component using SIRAP demands more resources in its worst-case scenario [14]. We therefore need to add a term, $I_{si}(t)$, to account for *self-blocking* to the $\text{rbf}_s(t, i)$. The self-blocking term I_{si} of a task τ_{si} is defined in terms of $z(t) = \left\lceil \frac{t}{P_s} \right\rceil$, representing an upper bound to the number of self-blocking occurrences within a time interval of length t, and a multi-set $G_{si}^{\text{sort}}(t)$ which comprises all self-blocking lengths X_{sil} that a task τ_{si} may experience by itself and other tasks τ_{sj} in the same component in a non-decreasing order. We recall that $G_{si}^{\text{sort}}(t)$ stores all values X_{sil} in a non-decreasing order and includes a value for each individual resource access by a job of task τ_{si} to resource R_l. As a supplemental to our evaluation and proofs, we will show how to construct such a multi-set in the Appendix.

C. Sources of pessimism in the existing analysis

For the SIRAP analysis one needs to know how many critical sections each job accesses. Although this information is not required using HSRP, it makes SIRAP superior to ONP.

HSRP accounts for a worst-case overrun in each component period, while an actual overrun does not necessarily happen each period. However, exposing a multi-set of resource-holding times to the global schedulability test (similar to SIRAP) is impossible for HSRP, because this breaks the independent analysis of components due to the dependency of $G_{si}^{\mathtt{sort}}(t)$ on the time values t in the testing set of the tasks in $\mathcal{T}_s$.

Initially, SIRAP accounts for one self-blocking too much, because of the ceiling operator in the definition of $z(t)$ as part of the self-blocking term. Since each element in the set $G_{si}^{\mathtt{sort}}(t)$ is at most of length X_s, the only reason for HSRP to become less pessimistic is when a self-blocking of approximately X_s is deducted in each component period. In this case the ONP analysis is more efficient.

We can conclude that SIRAP is always superior to ONP. In those cases where ONP yields better results, then the ONP analysis can be safely used to implement a SIRAP system.

Theorem 1: If a task set $\mathcal{T}_s$ is deemed schedulable on a periodic resource $\Gamma_s(P_s, Q_s, \mathcal{X}_s)$ using the ONP analysis, then it is also feasible on a periodic resource $\Gamma'_s(P_s, Q_s + X_s, \mathcal{X}_s)$ using a SIRAP implementation.

Proof: The sufficient schedulability condition for a task set $\mathcal{T}_s$ on a periodic resource $\Gamma_s(P_s, Q_s, \mathcal{X}_s)$ is given by [14]:

$$\forall \tau_i \in \mathcal{T}_s : \exists t \in (0, D_{si}] : \mathtt{rbf}_s(t,i) + I_{si}(t) \leq \mathtt{sbf}_{\Gamma_s}(t), \quad (13)$$

where $\mathtt{rbf}_s(t,i)$ is defined in (11), $\mathtt{sbf}_{\Gamma_s}(t)$ is defined in (1) and the exact construction of $I_{si}(t)$ is given in the Appendix. By definition it holds that $\forall e \in G_{si}^{\mathtt{sort}}(t) : e \leq X_s$. Hence, the schedulability condition in (13) is implied by:

$$\forall \tau_i \in \mathcal{T}_s : \exists t : \quad \mathtt{rbf}_s(t,i) + \left\lceil \frac{t}{P_s} \right\rceil X_s \leq \mathtt{sbf}_{\Gamma_s}(t). \quad (14)$$

Since within one budget period a self-blocking occurrence can only happen at the end of a supply due to insufficient budget to complete a critical section, we can remove the dependency on t provided that we add X_s extra budget in each component period. In other words, a conservative budget Q' is:

$$X_s + (\min Q_s : (\forall \tau_i \in \mathcal{T}_s : \exists t : \mathtt{rbf}_s(t,i) \leq \mathtt{sbf}_{\Gamma_s}(t))). \quad (15)$$

The right-hand term of (15) is the same as schedulability condition for ONP, see (10), which concludes our proof. ∎

Given Theorem 1, we make it possible to integrate a component validated by a standard analysis for SRP+FPPS into the HSF, while using SIRAP for global resource arbitration. Next, we derive an opaque analysis for OWP.

According to [15], [11], overrun with payback (OWP) has additional pessimism at the local schedulability compared to overrun without payback (ONP). Firstly, due to payback a component may supply less resource within a component period. Secondly, the payback increases the blackout duration of a component. Should overrun with payback therefore be considered obsolete based on these observations, or not?

V. SRP WITH BUDGET OVERRUNS: TO PAYBACK OR NOT?

We reconsider the problem of resource sharing across budgets. Ghazalie and Baker [2] recognized that when tasks access resources across their budget with the SRP, their budget may deplete during resource access so that other components may experience an excessive blocking duration. As a solution, they proposed to overrun the budget Q_s until the critical section completes and they subsequently deduct the amount of overrun from the next budget replenishment of the corresponding component. Their (global) analysis corresponds to the analysis in [3], [11] in the sense that we need to account for additional interference to all other components due to an worst-case over-provisioning of X_s budget which facilitates the overrun. This results in the sufficient schedulability condition under global FPPS of components, where the $\mathtt{RBF}(t,s)$ is defined in (8).
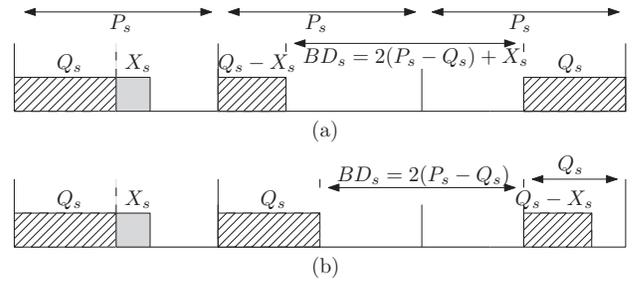

Figure 1. Worst-case characterization of the periodic processor supply for SRP with mechanisms for overrun and payback, as presented in [11].

We now need to characterize the worst-case resource supply to the tasks serviced by component C_s. Behnam et al. [11] distinguish two cases to represent the worst-case processor supply, see Figure 1. The worst-case scenario happens after the first budget supply of Q_s has overrun with an amount of X_s. This leads to a payback in one of the subsequent component periods. A payback in the second period, as shown in Figure 1(a), means that (*i*) the amount of overrun X_s is deducted from the next replenishment of Q_s; and (*ii*) the next replenishment of Q_s is serviced as late as possible before the deadline P_s. The longest blackout of the processor supply is $BD_s = 2(P_s - Q_s) + X_s$.

Alternatively, the component may overrun its budget again in the second period, see Figure 1(b), so that a payback happens in the third period. The budget in the third period is again supplied as late as possible, taking into account that there must be enough time until the deadline to accommodate for another overrun. Although this case has a smaller worst-case processor blackout of $BD_s = 2(P_s - Q_s)$, this is still pessimistic.

Since component deadlines are assumed to be equal to their period P_s, it is sufficient to consider the response time of the first activation of each component, see (8). Furthermore, the schedulability test in (5) guarantees that an amount of $Q_s + X_s$ budget can be provisioned within a period P_s. As a consequence, the latest start time of that budget provisioning is $P_s - (Q_s + X_s)$. This is independent of whether or not an overrun has taken place, as shown in Figure 2.
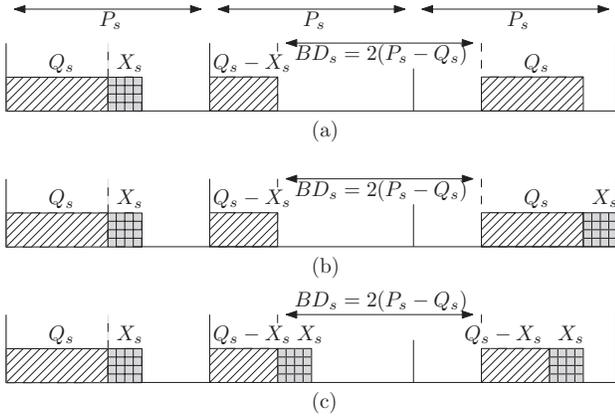
We can now derive the following lemma:

Figure 2. The latest starting time of the processor supply in each period is independent of whether or not an overrun takes place in that period.

Lemma 2: A component C_s following the periodic resource model $\Gamma_s(P_s, Q_s, \mathcal{X}_s)$, arbitrating global shared resource using the OWP mechanism, cannot experience more than the regular blackout duration of $BD_s = 2(P_s - Q_s)$.

Proof: Following the periodic resource model [1], shown in Figure 2, the latest time that a budget of at least $Q_s - X_s$ will be provisioned is at time $P_s - (Q_s + X_s)$, because there must be sufficient time between the finishing time of the normal budget Q_s and the period boundary P_s to accommodate for an overrun situation. Hence, the $P_s - X_s$ is an implicit deadline[2] for the normal budget Q_s, so that the blackout for two consecutive budget supplies is at most $BD_s = 2(P_s - Q_s)$. ■

The result of this lemma is the same as the analysis derived by Davis and Burns [3], although they do not support a compositional analysis. Behnam et al. [11] came up with an improved overrun method - called *enhanced overrun* - to improve the blackout duration assumed by their initial analysis, see Figure 3. They improve their analysis by postponing the next replenishment of a component, i.e. contrary to Lemma 2 they postpone the start time of the budget provisioning. However, their alternative (*i*) requires modifications in the implementation of the overrun mechanism, since it alters the periodicity of budget releases and (*ii*) still assumes a pessimistic budget supply of at most $Q_s - X_s$ in an interval of length $2P_s$.
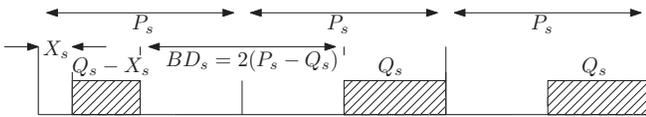

Figure 3. In [11] the extra blackout due to payback is reduced by introducing a flexible release off set for budget $Q_s - X_s$, i.e. the initial delay of X_s.

The latter source of pessimism is inherited from the analysis by Davis and Burns [3], which considers the effect of *push-through blocking* due to an overrun with payback. This effect is shown in Figure 2(c), where a task arrives just after depletion

[2]Contrary to ONP, we cannot make this implicit deadline explicit for OWP by applying the EDP model [18], because this would further reduce the blackout duration to $BD_s = 2(P_s - Q_s) - X_s$; this is obviously optimistic.

of budget $Q_s - X_s$. Although the task is *pushed through* to the next budget replenishment, the blackout duration of the processor supply remains $BD = 2(P_s - Q_s)$. Using the periodic resource model [1], however, we already assume an initial delay of BD_s followed by a periodic supply of a budget of size Q_s.

We already observed that the overrun budget X_s is merely for global reasons, because the task set does not need an extra budget of X_s, i.e. it is already feasible with a budget of Q_s every period P_s. The remaining question is: given that a fixed-priority-scheduled task set using a plain SRP-based resource arbitration is schedulable on a periodic resource $\Gamma_s(P_s, Q_s, \mathcal{X}_s)$, is there any task that may experience insufficient budget after a payback of at most X_s budget?

The analysis by Behnam et al. [11] is based on the point of view that the minimum resource supply in an interval of length P_s must be assumed to be equal to $Q_s - X_s$, as suggested by Figure 1. We will show that the model in [11] is indeed overly complex and pessimistic. The main reasoning behind this claim is that the task set as a whole actually receives Q_s budget in an interval of length P_s, but the *individual resource supply to a task activation* has changed. An overrun advances exactly the amount of budget of at most X_s to complete the critical section. The task activations that have consumed this overrun cannot claim again processor time in the next budget supply, so that a potential subsequent overrun cannot be caused by them. The overrun budget in Figure 2 is grid-marked to indicate its partial availability.

Lemma 3: Given that a fixed-priority-scheduled task set $\mathcal{T}_s$ under SRP-based resource arbitration is schedulable on a periodic resource $\Gamma_s(P_s, Q_s, \mathcal{X}_s)$, a task $\tau_{si} \in \mathcal{T}_s$ cannot miss its deadline when adding an overrun with payback mechanism.

Proof: We only need to consider the case where an overrun situation has taken place subsequently causing a payback at the next budget replenishment. In other situations the resource supply is unchanged compared to the $\mathtt{sbf}_{\Gamma_s}$ for independent components, see (1).

We observe that an overrun situation can only be caused by a resource lock by any of the tasks $\tau_{si} \in \mathcal{T}_s$. Assume that task τ_{si} locks resource R_l, so that the component ceiling is at least equal to the resource ceiling rc_{sl}. Furthermore, budget Q_s depletes during resource access. This means that component C_s may overrun its normal budget Q_s for at most an amount of X_{sl} processor time, which allows to complete the critical section initiated by task τ_{si}.

We proof by contradiction that no task $\tau_{sj} \in \mathcal{T}_s$ will miss a deadline due to the payback of X_{sl} budget at the next replenishment of the normal budget Q_s, i.e. assume that there exists a task $\tau_{sj} \in \mathcal{T}_s$ that will miss a deadline after an overrun.

We tackle this proof obligation by distinguishing four cases: tasks that may preempt the critical section ($j < rc_{sl}$), tasks that are blocked during the critical section ($rc_{sl} \leq j < i$), the resource-locking task τ_{si} itself ($i = j$) and tasks that have a lower priority than the resource-locking task ($i < j$).

1) $j < rc_{sl}$: these tasks may preempt the critical section. Moreover, these tasks contribute to the length of X_{sl} for at most a single preemption (Lemma 1). This means that if the task

arrives after depletion of Q_s and an overrun takes place, then it will execute in the overrun budget. Contrary to the assumptions in [11], these task *will actually consume* the overrun budget when it is available. An activation of task τ_{sj} which consumes C_{sj} of overrun budget cannot request the same amount of budget in the next budget period P_s, because it has already finished its execution during the overrun. And vice versa: if an activation of task τ_{sj} requests for C_{sj} of normal budget, then it did not execute during a possible overrun in the previous budget period. An overrun in the previous period could therefore have at most a length of $X_{sl} - C_{sj}$. If C_{sj} of the overrun has not been consumed, then the next budget supply will also not be reduced with this amount of payback. Thus, the resources requested by the current activation of task τ_{sj}, i.e. C_{sj}, will be available before task τ_{sj} will miss a deadline. Hence, no higher priority task τ_{sj} where $j < rc_{sl}$ will miss a deadline due to a payback.

2) $rc_{sl} \leq j < i$: these tasks are blocked during the critical section by the resource ceiling. When we do not advance the overrun budget X_{sl} compared to plain SRP-based resource arbitration, these tasks are schedulable. The reason for this is that the blocking duration of at most X_{sl} is already accounted in the $\mathrm{rbf}_s(t, j)$ of task τ_{sj}. A new periodic supply cannot start with local blocking, because blocking should already start in the previous provisioning and use the overrun (if needed). Hence, OWP does not cause a deadline miss for any of the tasks τ_{sj} that are blocked by the resource-accessing task τ_{si}.

3) $i = j$: for the resource locking task τ_{si} itself the same reasoning holds as for the first case: it either consumes an amount of h_{sil} of the overrun budget in the previous budget period or it consumes h_{sil} from the normal budget Q_s in the current budget period. Both cases are mutually exclusive and cannot cause a deadline miss.

4) $i < j$: these tasks have a lower priority than the resource-locking task and have already accounted X_{sl} as interference in their $\mathrm{rbf}_s(t, j)$. Hence, similarly to case 3, these tasks cannot assume that any budget would be immediately available after replenishment of Q_s in case of plain SRP-arbitration. The overrun with payback mechanism does therefore not cause a deadline miss to any of the tasks τ_{sj} where $i < j$.

By contradiction we have shown that advancing the resource supply of X_s due to overrun with payback does not hamper the schedulability of task set $\mathcal{T}_s$ compared to plain SRP-based resource arbitration. ∎

From both Lemma 2 and Lemma 3 we directly obtain the following result:

Theorem 2: The local schedulability analysis for a task-set $\mathcal{T}_s$ on an SRP+fixed-priority-scheduled periodic resource $\Gamma_s(P_s, Q_s, \mathcal{X}_s)$ can be applied when arbitrating global shared resources using overrun with payback (OWP).

We believe this theorem yields an interesting result, because it shows that the local schedulablity analysis of overrun with and without payback are exactly the same. In particular, we can reuse the sufficient schedulability condition for ONP as presented in (10).

Finally, we answer the main question of this section:

to payback or not? The global schedulability analysis for components arbitrated by overrun with payback is unchanged and was already considerably better than the global analysis of overrun without payback. In addition, we have improved the local schedulability analysis, such that there is no difference between ONP and OWP. Hence, there is no reason to deploy overrun without a payback mechanism from a compositional schedulability perspective.

VI. OPAQUE ANALYSIS: EFFICIENCY VS ABSTRACTION

In this section we define the notion of an *opaque analysis*.

Definition 1: An opaque analysis provides a sufficient schedulability condition for a task set $\mathcal{T}_s$. Even under global resource sharing, it uses an unmodified processor supply abstraction $\mathrm{sbf}_{\Gamma_s}(t)$ as in (1) and an unmodified processor request bound for all tasks in $\mathcal{T}_s$ compared to their analysis on a dedicated processor, i.e. $\mathrm{rbf}_s(t, i)$ as in (11).

This definition makes it possible to compare the different synchronization protocols at the same abstraction level. Moreover, given an opaque analysis for SIRAP, OWP and ONP, we can defer the selection of a global synchronization protocol until component-integration time. We first investigate what analytical opacity means. Secondly, we investigate what opacity means for a component programmer.

A. Opacity from an analytical perspective

For the ONP and OWP, it is sufficient to know the maximum resource holding time, X_{sl}, of an unspecified access to resource R_l. Contrary to SIRAP, this information is independent of the time values inspected for the local schedulability analysis and the number of accesses to a resource by a job.

The ONP analysis has been improved in [19] by introducing an explicit deadline using the EDP model for the budget supply. However, since this deadline modifies the processor supply sbf based on the value of X_s, this analysis violates our definition of opacity. The same holds for the enhanced overrun in [11], which we consider obsolete based on our new OWP analysis. Table I gives an overview of analysis methods for global resource sharing in HSFs based on their opacity compliance.

Table I
OVERVIEW OF THE SYNCHRONIZATION PROTOCOL'S SUPPORT FOR INTEGRATING COMPONENTS INTO THE HSF WITH OPAQUE ANALYSIS.

Analysis	Opacity
BROE [10]	yes
HSRP - overrun without payback (ONP) [3]	no
HSRP - overrun without payback (ONP) [11]	yes
Enhanced overrun [11]	no
Improved overrun without payback (IONP) [19]	no
HSRP - overrun with payback (OWP) [3], [11]	no
SIRAP [4], [14]	no

We showed that ONP provides an opaque analysis for SIRAP and we proposed an opaque OWP analysis.

B. Opacity from a programmer's perspective

For a programmer, opacity serves two purposes. Firstly, a component may use logical resources for which exclusive local usage or global usage is determined upon component

integration [20]. From an *opacity* perspective [20], i.e. neither the environment nor other components can modify a component's code, unified primitives may be desirable to access local and global resources. The actual binding of function calls to the synchronization primitives that arbitrate either global or local resource access can be done at compile time or when the component is loaded into the framework. This dynamic binding of primitives makes it possible to decouple the specification of global resources from their use in the implementation.

Secondly, a component may *require* external resources in the component's code. Opacity abstracts that the resource is global to the component during development. Since accessing global resources may block tasks in other components, a programmer must specify any required global resource. The system integrator cannot perform the global schedulability analysis without a valid upper bound on the resource holding times X_{sl} of each component C_s that shares resource R_l.

Only during integration time, however, one actually knows whether or not the global resources specified in the timing interface are globally shared, i.e. if resource R_l is not shared by any other component, then resource R_l can be considered as a local shared resource. If we have chosen the SIRAP analysis to analyze global resource arbitration, then we need to redo the local schedulability analysis at integration time to make use of the fact that R_l is a local resource. With the ONP analysis, which provides an upper bound for SIRAP, it is unnecessary to redo the local analysis, because the local analysis presented in (11) is compliant with local SRP. An opaque analysis therefore facilitates an abstraction for global resources sharing during component development.

VII. EVALUATION

This section evaluates the budgets allocated by the periodic resource model under different protocols for global synchronization. We look for the percentage of schedulable task sets. Contrary to [15], [11], [14] where a protocol is chosen at the system level, we investigate for which task-characteristics a particular analysis method is better, i.e. at the component level. From these results, we may later derive which protocol matches the best with given system characteristics.

For each components the task periods T_{si} are uniformly drawn from the interval $[140, 1000]$. We assume deadlines equal to periods, i.e. $T_{si} = D_{si}$ and we assign deadline monotonic priorities to tasks. The individual task utilizations u_{si} are generated using the UUnifast algorithm [21]. Using the task's utilization u_{si} and the randomly generated period T_{si}, we can derive the worst-case execution time C_{si} of a task τ_{si}, i.e. $C_{si} = u_{si} \times T_{si}$. All tasks access a single global resource for a random duration between $0.1 \times C_{si}$ and $0.25 \times C_{si}$. In each simulation study a new set of 1000 systems is generated and the following settings are changed:

1) *Component utilization:* The utilization of a component $U(\mathcal{T})$ is varied within a range of $[0.05, 1.0]$ using incremental steps of 0.05, see Figure 4.
2) *Component periods:* The period of the periodic resource P_s is varied within a range of $[5, 70]$ with incremental steps of 5, see Figure 5.

For comparison purposes we included the results for the improved local analysis of ONP [19], i.e. IONP. Both experiments show that the different overrun methods have little impact on the local schedulability of a task set on a periodic resource. The main reason for this is the constraint that the calculated budget Q_s and the overrun budget X_s have to fit within period P_s, i.e. we applied the constraint $Q_s + X_s \leq P_s$. For SIRAP, we require that $X_s \leq Q_s$. Due to this constraint, SIRAP's performance is suppressed for small resource periods. However, both figures show the cost of an opaque analysis in the context of two-level FPPS-based HSFs.
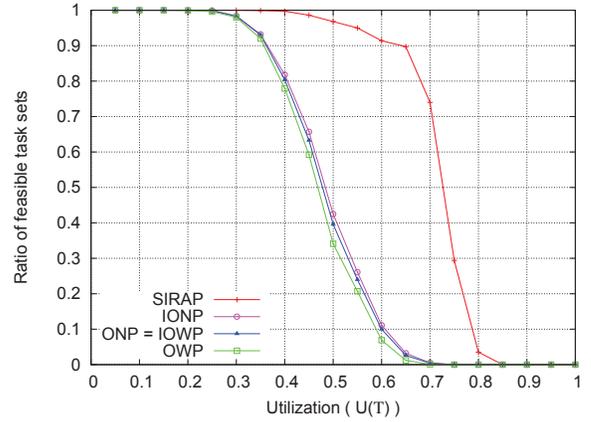

Figure 4. Ratio of schedulable task sets versus the utilization, where the component period is $P_s = 40$ and the number of tasks is $n_s = 8$.
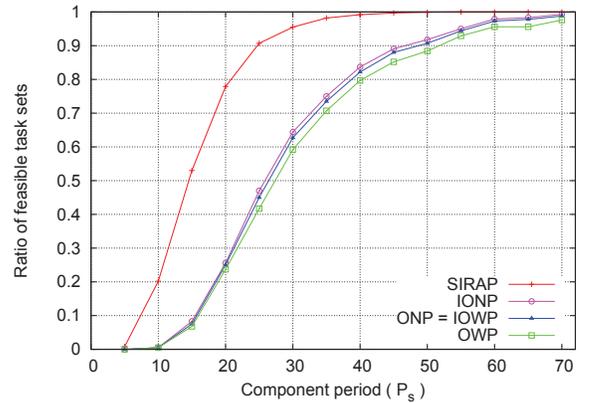

Figure 5. Ratio of feasible task sets as a function of the component period, where the number of tasks is $n_s = 8$ and the utilization $U(\mathcal{T}) = 0.4$.

The constraint, $Q_s + X_s \leq P_s$, is the main weakness for all overrun variants, determined by the ratio $\frac{X_s}{P_s}$. This ratio can be increased by increasing the utilization (Figure 4), choosing smaller resource periods (Figure 5), decreasing the number of tasks (n_s) or by increasing the range of the task periods. When keeping the utilization $U(\mathcal{T})$ constant, the last two alternatives result in larger computation times and resource access times. Since X_s is computed from a fixed fraction of the tasks' computation times, this increases the $\frac{X_s}{P_s}$ ratio. We

leave the remaining experimental results out of this paper due to space constraints. Since OWP performs equally well as ONP at the local level, and the global schedulability is superior for OWP compared to ONP, OWP is prefered above ONP.

Note that the non-opaque IONP analysis in [19] may slightly improve on IOWP and ONP. However, the global analysis for OWP is always better than or equal to the global analysis of ONP. This gives an advantage to ONP when both integration tests in (6) and (8) yield the same result, i.e. when all *component periods* are chosen approximately the same, so that also OWP accounts for an overrun in each component period.

VIII. RECOMMENDATIONS AND CONCLUSIONS

This paper introduced the notion of *opaque* analysis for resource sharing components that need to be integrated on a single processor platform. An opaque analysis makes it possible to defer the choice for a resource-sharing protocol until component integration time. Although SIRAP's analysis is not opaque, we can use overrun without payback (ONP) as a conservative and opaque alternative. We can obtain a tighter schedulability analysis using SIRAP's analysis, if we are provided a task-set's detailed timing information. Finally, we also provided an opaque analysis for overrun with payback (OWP), which dominates the opaque ONP. Only when all component periods are almost the same, a non-opaque ONP may take advantage over OWP.

REFERENCES

[1] I. Shin and I. Lee, "Compositional real-time scheduling framework with periodic model," *ACM Trans. on Embedded Computing Systems*, vol. 7, no. 3, pp. 1–39, 2008.
[2] T. M. Ghazalie and T. P. Baker, "Aperiodic servers in a deadline scheduling environment," *Real-time Syst.*, vol. 9, no. 1, pp. 31–67, 1995.
[3] R. Davis and A. Burns, "Resource sharing in hierarchical fixed priority pre-emptive systems," in *Real-Time Systems Symp.*, 2006, pp. 257–267.
[4] M. Behnam, I. Shin, T. Nolte, and M. Nolin, "SIRAP: A synchronization protocol for hierarchical resource sharing in real-time open systems," in *Conf. on Embedded Software*, Oct. 2007, pp. 279–288.
[5] T. Baker, "Stack-based scheduling of realtime processes," *Real-Time Syst.*, vol. 3, no. 1, pp. 67–99, March 1991.
[6] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority inheritance protocols: an approach to real-time synchronisation," *IEEE Trans. on Computers*, vol. 39, no. 9, pp. 1175–1185, Sept. 1990.
[7] Z. Deng and J.-S. Liu, "Scheduling real-time applications in open environment," in *Real-Time Systems Symp.*, Dec. 1997, pp. 308–319.
[8] T.-W. Kuo and C.-H. Li, "A fixed-priority-driven open environment for real-time applications," in *Real-Time Systems Symp.*, 1999, pp. 256–267.
[9] G. Lipari and S. Baruah, "Efficient scheduling of real-time multitask applications in dynamic systems," in *Real-Time Technology and Applications Symp.*, May 2000, pp. 166–175.
[10] M. Bertogna, N. Fisher, and S. Baruah, "Resource-sharing servers for open environments," *IEEE Trans. on Industrial Informatics*, vol. 5, no. 3, pp. 202–219, Aug. 2009.
[11] M. Behnam, T. Nolte, M. Sjodin, and I. Shin, "Overrun methods and resource holding times for hierarchical scheduling of semi-independent real-time systems," *IEEE Trans. on Industrial Informatics*, vol. 6, no. 1, pp. 93–104, Feb. 2010.
[12] M. Caccamo and L. Sha, "Aperiodic servers with resource constraints," in *Real-Time Systems Symp.*, Dec. 2001, pp. 161–170.
[13] P. Holman and J. Anderson, "Locking in pfair-scheduled multiprocessor systems." in *Proc. Real-Time Systems Symp.*, Dec. 2002, pp. 149–158.
[14] M. Behnam, T. Nolte, and R. J. Bril, "Bounding the number of self-blocking occurrences of sirap," in *Real-Time Systems Symp.*, Dec. 2010.
[15] M. Behnam, T. Nolte, M. Åsberg, and R. J. Bril, "Overrun and skipping in hierarchically scheduled real-time systems," in *Conf. on Embedded Real-Time Computing Systems and Applications*, Aug. 2009, pp. 519–526.
[16] L. Almeida and P. Peidreiras, "Scheduling with temporal partitions: response-time analysis and server design," in *Conf. on Embedded Software*, Sept. 2004, pp. 95–103.
[17] M. Bertogna, N. Fisher, and S. Baruah, "Static-priority scheduling and resource hold times," in *Parallel and Distributed Processing Symp.*, 2007.
[18] A. Easwaran, M. Anand, and I. Lee, "Compositional analysis framework using EDP resource models," in *Real-Time Sys. Symp.*, 2007, pp. 129–138.
[19] M. Behnam, T. Nolte, and R. J. Bril, "Tighter schedulability analysis of synchronization protocols based on overrun without payback for hierarchical scheduling frameworks," in *Conf. on Engineering of Complex Computer Systems*, April 2011.
[20] P. López Martinez, L. Barros, and J. Drake, "Scheduling configuration of real-time component-based applications," in *Reliable Softw. Technology - Ada-Europe*, ser. LNCS. Springer, 2010, vol. 6106, pp. 181–195.
[21] E. Bini and G. Buttazzo, "Biasing effects in schedulability measures," in *Euromicro Conf. on Real-Time Systems*, July 2004, pp. 196–203.

APPENDIX
CONSTRUCTING SELF-BLOCKING SETS

For the SIRAP analysis [14] we need to construct a multiset $G_{si}^{\mathtt{sort}}(t)$ of self-blocking durations that a task τ_{si} may experience in a time interval of length t. The self-blocking term I_{si} of a task τ_{si} is defined as follows:

$$I_{si}(t) = \sum_{1 \leq l \leq z(t)} G_{si}^{\mathtt{sort}}(t)[l], \tag{16}$$

where $z(t) = \left\lceil \frac{t}{P_s} \right\rceil$ defines an upper bound to the number of self-blocking occurrences within a time interval of length t and $G_{si}^{\mathtt{sort}}(t)$ defines an multi-set (i.e. a set including duplicates of values X_{sil}) of self-blocking lengths that a task τ_{si} may experience by itself and other tasks τ_{sj} in the same component.

This multi-set contains the extra blocking that a task may suffer due to self-blocking by lower priority tasks:

$$I_{si}^{low} = \max(0, \max\{X_{sjl} \mid i < j \wedge X_{sjl} > 0 \wedge rc_l \leq i\}). \tag{17}$$

In addition, the multi-set contains the self-blocking durations of task τ_{si} itself and the interference caused by self-blocking of higher priority tasks, so that we can define the multi-set $G_{si}(t)$ as follows [14]:

$$G_{si}(t) = \{I_{si}^{low}\} \cup$$

$$\left(\bigcup_{(1 \leq j \leq i)} \bigcup_{\left(1 \leq k \leq \left\lceil \frac{t}{T_{sj}} \right\rceil\right)} \bigcup_{(R_l \in \mathcal{R}_s)} \bigcup_{(1 \leq a \leq m_{sjl})} \{X_{sjl}\} \right). \tag{18}$$

The term $\bigcup_{(j \leq i)}$ iterates over all tasks τ_{sj} with an higher priority than task τ_{si} and includes the self-blocking by task τ_{si} itself when $i = j$; the term $\bigcup_{\left(1 \leq k \leq \left\lceil \frac{t}{T_{sj}} \right\rceil\right)}$ considers all activations of task τ_{sj} in an interval of length t; the term $\bigcup_{(R_l \in \mathcal{R}_s)}$ considers all resources R_l accessed by task τ_{sj} and, finally, the term $\bigcup_{(1 \leq a \leq m_{sjl})}$ iterates over *the number of resource accesses* to resource R_l by task τ_{sj}. In other words: during each job-activation a task τ_{sj} may accesses a shared resource R_l for m_{sjl} times and it can self-block at any of these attempts. Finally, we sort the values in the multi-set $G_{si}(t)$ in non-increasing order, resulting in the multi-set $G_{si}^{\mathtt{sort}}(t)$. Equation (16) contributes a number of $z(t)$ largest self-blocking occurrences that a task τ_{si} may experience in an interval of length t, i.e. the first $z(t)$ elements of $G_{si}^{\mathtt{sort}}(t)$.