Opaque analysis for resource-sharing components in hierarchical real-time systems : extended version

Opaque analysis for resource-sharing components in hierarchical real-time systems - extended version -

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

A real-time component may be developed under the assumption that it has the entire platform at its disposal. Composing a real-time system from independently developed components may require resource sharing between components. We propose opaque analysis methods to integrate resource-sharing components into hierarchically scheduled systems. Resource sharing imposes blocking times within an individual component and between components. An opaque local analysis ignores global blocking between components and allows to analyse an individual component while assuming that shared resources are exclusively available for a component.

To arbitrate mutually exclusive resource access between components, we consider four existing protocols: SIRAP, BROE and HSRP - comprising overrun with payback (OWP) and overrun without payback (ONP). We classify local analyses for each synchronization protocol based on the notion of opacity and we develop new analysis for those protocols that are non-opaque.

Finally, we compare SIRAP, ONP, OWP and BROE by means of an extensive simulation study. From the results, we derive guidelines for selecting a global synchronization protocol[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 executed in isolation will continue meeting its timing constraints after integration (or admission) on a shared uni-processor platform. The HSF is therefore a promising solution for industrial standards which more often specify that an underlying operating system should prevent timing faults in any component to propagate to other components on the same processor.

An HSF provides temporal isolation between components by allocating a *processor budget* to each component. To analyse a component's budget requirement independently of other components, compositional real-time scheduling frameworks have been developed. Their main goal [1] is establishing global (system level) timing properties by composing independently specified and analyzed local (component level) timing properties. Local timing properties are analyzed by assuming a worst-case supply of processor resources to a component. The way of modeling the processor supply is defined by a resource model, e.g., the periodic resource model [1] or the bounded-delay model [2]. These models make it possible to combine and abstract deadline constraints of all tasks within a component as a single real-time constraint, called *a real-time interface*. Components can be composed by combining a set of real-time interfaces, which will treat each component as a single task by itself.

An HSF without further resource sharing is unrealistic, since components may, for example, use operating system services, memory mapped devices and shared communication devices requiring mutually exclusive access. An HSF with support for resource sharing makes it possible to share serially accessible resources (from now on referred to as resources) between arbitrary tasks, which are located in arbitrary components, in a mutually exclusive manner. A resource that is only shared by tasks within a single component is a *local shared resource*. A resource that is used in more than one component is a *global shared resource*. Any access to a resource is assumed to be arbitrated by a synchronization protocol.

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 [3]. To prevent such budget depletion during global resource access (see Figure 1), four synchronization protocols [4], [5], [6] have been proposed based on the Stack Resource Policy (SRP) [7]. These are based on two general mechanisms:

[1]Part of this work has been presented in the CRTS 2011 workshop (co-located with the RTSS) and is unofficially available as a technical report from the University of York (YCS-2011-469).

(*i*) *self-blocking* when the remaining budget is insufficient to complete a global access - having two flavors called SIRAP [5] and BROE [6] or (*ii*) *overrun* the budget until the resource is released - called HSRP [4]. HSRP has 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 returned in the next budget period.

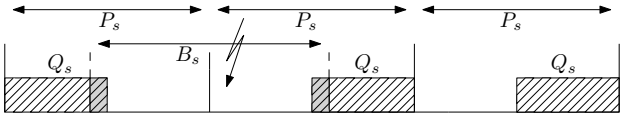

Figure 1. When the budget Q_s of a task depletes while a task executes on a global resource, tasks in other components may experience excessive blocking durations, B_s.

A. Towards opaque component development

In practical situations, a component developer is typically unconcerned about the sharing scope of resources. A component may access resources for which just local usage or (shared) global usage is determined only upon integration. During component development unified primitives may be desirable to access all resources. The actual binding of function calls to scope-dependent synchronization primitives, that arbitrate either global or local resource access, can be done at compile time or when the component is loaded. Dynamic binding of primitives makes it possible to decouple the specification of global resources from their use in the implementation. This decoupling is called *opacity* [8] and it abstracts whether the resource is global in the system.

B. Opaque local analysis

After developing a component and before publishing it to a framework integrator, a component is packaged as a re-usable entity. This includes deriving a timing interface to abstract from internal deadline constraints of tasks. Such an abstraction requires an explicit choice for a resource model, capturing the virtual processor supply to a component. Moreover, those resources that may be globally shared are exposed in the component's interface, i.e, a component specifies what it needs in terms of resources. Whether or not a global resource is actually used by other components is unknown within the context of a component.

If, and only if, a global resource is actually shared between components, it must be arbitrated by a global synchronization protocol. To prevent budget depletion during resource access, processor resources may need to be delivered differently. This, on its turn, may add constraints to the supply of processor resources in order to preserve local deadline constraints. Opacity requires that the implementation of a component does not use any assumptions about these constraints and modifications.

There are several ways to account for local scheduling penalties due to global resource sharing. One might assume that each resource is global and, subsequently, account for the worst-case overhead inside the local analysis (e.g., SIRAP [5], [9] and OWP [4], [10]). Alternatively, one may assume that all resources are local during the local analysis and compensate for sharing between components at integration time (e.g., ONP [10]).

The latter alternative presents the same view as during component development, i.e., a component has the entire platform at its disposal and all resources. Whenever a synchronization protocol for global resources is used that is compliant with a synchronization protocol for local resources, the local analysis of a component can be based on local properties only. We call such a local analysis *opaque*, because it separates local and global resource arbitration.

Definition 1: An opaque analysis provides a sufficient local schedulability condition for an individual component. It considers all resources as exclusively local and, even under global sharing, it excludes global timing information of global resources.

Table I gives an overview of local analyses by indicating their opacity. The local analysis of ONP in [10] satisfies the notion of opacity, because it uses a simple overrun upon integration and nothing else locally. Surprisingly and contrary to ONP, the current local OWP analysis is non-opaque, because it needs to know which resource are globally shared. For the same reason SIRAP has no opaque local analysis. Contrary to the other protocols, BROE only applies to global EDF of components and explicitly assumes the bounded-delay resource model [2]. Since BROE's underlying resource model captures the processor supply to a component sufficiently pessimistic, all resources can be treated as local in the local analysis, i.e., BROE's local analysis is opaque.

Contributions: The main contribution of this paper is leveraging the concept of an opaque analysis to a methodology for deriving and composing resource-efficient component interfaces, e.g., independent of a chosen resource model. Moreover, an opaque analysis makes it possible to defer the choice for a global synchronization protocol until component integration.

First, we reduce the pessimism of OWP. Our new OWP analysis is opaque and, in most cases, it is better than ONP. Secondly, we show that ONP can be used as an upper bound for SIRAP. This means that an opaque analysis for ONP provides also an opaque analysis technique for resources arbitrated by an implementation of SIRAP. Thirdly, we are the first to present an extensive experimental comparison of the different analysis techniques for BROE, SIRAP, OWP and ONP. Global resource sharing eventually causes global scheduling costs. We therefore do not only evaluate the individual protocols, but we also evaluate the effect of using an opaque analysis for them. Finally, we derive new guidelines for selecting a synchronization protocol.

Organization: The remainder of this paper is organized as follows. Section II presents related work. Section III describes our system model. Section IV recapitulates the mechanisms for global sharing of the synchronization protocols considered in this paper (i.e., HSRP, SIRAP and BROE). Section V recapitulates the existing compositional analysis for HSFs in the presence of shared resources. Section VI presents an improved, simplified and opaque analysis for overrun with payback (OWP). Section VII presents a methodology which shows how an opaque analysis allows for an efficient design-space exploration of resource-sharing components. We subsequently show how our methodology applies to SIRAP. Section VIII evaluates the different analyses and the different protocols for global resource sharing by means of a simulation study. We investigate how global resource sharing impacts the schedulability of an individual component and, subsequently, how it impacts the schedulability of an entire system. Finally, Section IX concludes this paper with guidelines for selecting a global synchronization protocol.

II. Related work

Deng and Liu [12] proposed a two-level HSF for open systems, where components may be independently developed and validated. The corresponding schedulability analysis have been presented in [13] for fixed-priority preemptive scheduling (FPPS) and in [14] 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 [4], SIRAP [5] and BROE [6]. 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 [3]. This mechanism was later re-used in HSRP in the context of two-level HSFs by Davis and Burns [4] and complemented with a variant *without* payback. Although the analysis presented in [4] does not integrate in HSFs due to the lacking support for independent analysis of components, this limitation is lifted in [10].

The idea of self-blocking has also been considered in different contexts, e.g. for supporting soft real-time tasks [15] and for a zone-based protocol in a pfair-scheduling environment [16]. SIRAP [5] uses self-blocking for hard real-time tasks in HSFs on a single processor and its associated analysis supports composability. In [9] the original SIRAP analysis [5] 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 opaquely with little timing characteristics.

The original SIRAP [5] and HSRP [10] analyses have been analytically compared with respect to their impact on the system load for various component parameters [17]. The performance of each protocol heavily depends

Table I
Overview of the synchronization protocol's support for integrating components into the HSF with opaque analysis.

Analysis of global resource-sharing strategies	Opacity
BROE [6]	yes
HSRP - overrun without payback (ONP) [4]	no
HSRP - overrun without payback (ONP) [10]	yes
Enhanced overrun [10]	no
Improved overrun without payback (IONP) [11]	no
HSRP - overrun with payback (OWP) [4], [10]	no
SIRAP [5], [9]	no

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 of Davis and Burns [4]. Our new analysis methods make the results in [17] obsolete and we will provide new guidelines, including BROE, to select a synchronization protocol in two-level HSFs.

III. Real-time scheduling model

A. Component and task model

A system contains a single processor, a set $\mathcal{C}$ of N components C_1, ..., C_N and a set $\mathcal{R}$ of M serially accessible global resources R_1, ..., R_M. Each component C_s has a dedicated budget which specifies its periodically guaranteed fraction of the processor. The timing interface of a component C_s is specified by means of a triple $\Gamma_s = (P_s, Q_s, \mathcal{X}_s)$, where $P_s \in \mathbb{R}^+$ denotes its period, $Q_s \in \mathbb{R}^+$ denotes its budget, and $\mathcal{X}_s$ denotes 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 $X_{sl} > 0 \Leftrightarrow R_l \in \mathcal{R}_s$.

Each component C_s contains a set $\mathcal{T}_s$ of n_s sporadic tasks τ_{s1}, ..., τ_{sn_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 deadline-monotonic order, i.e. τ_{s1} has the smallest deadline and τ_{sn_s} has the largest deadline.

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 $h_{sil} > 0 \Leftrightarrow R_l \in \mathcal{R}_s$.

B. Resource models for a virtual processor

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

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

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*, i.e. $BD_s = 2(P_s - Q_s)$. The linear lower bound of the periodic resource with parameters $\Gamma_s = (P_s, Q_s, \mathcal{X}_s)$ is given by [1]:

$$\mathtt{lsbf}_{\Gamma_s}(t) = \frac{Q_s}{P_s} \left(t - 2\left(P_s - Q_s\right)\right), \tag{2}$$

modeling a bounded-delay resource [2] with a virtual processor speed of $\frac{Q_s}{P_s}$ and a longest initial service delay BD_s [18].

C. Synchronization protocol

This paper focuses on arbitrating *global* shared resources using SRP. To be able to use SRP for synchronizing global resources, its associated ceiling terms need to be extended.

1) Preemption levels: Each task τ_{si} has a static preemption level equal to $\pi_{si} = 1/D_{si}$. Similarly, a component has a preemption level equal to $\Pi_s = 1/P_s$, where period P_s serves as a relative deadline. If components (or tasks) have the same calculated preemption level, then the smallest index determines the highest preemption level.

2) 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 values, which are defined as the highest preemption level of any component or task that shares the resource. According to SRP, these ceilings are defined as:

$$RC_l = \max(\Pi_N, \max\{\Pi_s \mid R_l \in \mathcal{R}_s\}), \quad (3)$$

$$rc_{sl} = \max(\pi_{sn_s}, \max\{\pi_{si} \mid h_{sil} > 0\}). \quad (4)$$

We use the outermost $\max$ in (3) and (4) to define RC_l and rc_{sl} in those situations where no component or task uses R_l.

The local resource ceiling rc_{sl} influences the *resource holding times* [19], i.e. X_{sil} of a task τ_{si} to a resource R_l. The resource holding time includes the cumulative processor requests of tasks within the same component C_s that can preempt τ_i while it is holding resource R_l. The way of computing resource holding times under a particular global synchronization protocol may deviate from [19], e.g., see [5], [6] and [10]. However, it can be simplified by assuming that the component's period is smaller than the tasks' periods. The next lemma generalizes [17], [10] for global SRP:

Lemma 1: Given $P_s < T_s^{\min}$ and $T_s^{\min} = \min\{T_{si} | 1 \leq i \leq n_s\}$, all tasks τ_{sj} that are allowed to preempt a critical section accessing a global shared resource R_l, i.e. $\pi_{sj} > rc_{sl}$, can preempt at most once during an access to resource R_l when using any global SRP-compliant protocol and independent if the local scheduler is EDF or FPPS.

Proof: If a task, having a period of at least $T_s^{\min}$, executes two or more times inside a critical section of resource R_l, then the resource is also locked during this period, i.e., $X_{sil} > T_s^{\min}$. Since $P_s < T_s^{\min}$, this would mean that $X_{sil} > P_s$. According to SRP [7], a global resource should be accessed and released by the same instance of a component, i.e., within period P_s. However, $X_{sil} > P_s$ yields a contradiction by requiring a component utilization of $U_s \geq \frac{X_{sil}}{P_s} > 1$, making the component infeasible. ■

Lemma 1 makes it possible to compute the *resource holding time*, X_{sil} of task τ_{si} to resource R_l as follows:

$$X_{sil} = h_{sil} + \sum_{\pi_{sj} > rc_{sl}} C_{sj}, \quad (5)$$

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

3) 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 preemption level is higher than its component ceiling. A similar condition for preemption holds for components.

IV. Global synchronization: prevent excessive blocking

In this section, we give a brief overview of the run-time mechanisms employed by the synchronization protocols considered in this paper. Each of the protocols applies straightforward resource arbitration by SRP at the local level, for both local and global resources. This means that when a task has started its execution and tries to access a resource, irrespective of any other protocol specific actions for global synchronization, the local component ceiling is updated as if resource access is granted.

To prevent budget depletion while a task executes on a shared resource, HSRP [4] allows to overrun the budget until the task releases the resource. Alternatively, SIRAP [5] and BROE [6] each employ a self-blocking mechanism to prevent budget overruns by only granting resource access when there is sufficient budget to complete the resource access.

A. HSRP: Budget overruns

HSRP [4] uses an overrun mechanism [10] when a budget depletes during a critical section. If a task $\tau_{si} \in \mathcal{T}_s$ has locked a global resource when its component's budget Q_s depletes, then component C_s can continue its execution until task τ_{si} releases the resource.

To distinguish this additional amount of required budget from the *normal budget* Q_s, we refer to X_s as an *overrun budget*. HSRP has two flavors: overrun with payback (OWP) and overrun without payback (ONP). The term *without*

payback means that the additional amount of budget consumed[2] during an overrun does not have to be returned in the next budget period.

Budget overruns cannot take place across replenishment boundaries, i.e. for each component C_s the analysis guarantees $Q_s + X_s$ processor time before its relative deadline P_s [4], [10].

B. SIRAP: task-level self-blocking

With SIRAP [5], [9] a task is only allowed to access a global resource when it has sufficient budget to complete the entire critical section. If a resource attempts to access a resource and the remaining budget is insufficient, then the task blocks itself until the budget is replenished. SIRAP guarantees access to a global resource in the replenished budget subsequent to self-blocking. Essentially, a self-blocked task τ_{si} consumes at most X_{sil} amount of idle time from the component's budget while the task is waiting for its budget to replenish.

After self-blocking has caused budget-depletion, tasks with an higher priority than the local resource ceiling ($\pi_{sj} > rc_{sl}$) may arrive. Those jobs will be pushed through to the next budget period, but those are not accounted for in the resource holding time. To avoid the additional complexity of analysing this extra budget requirement and to avoid tasks from executing twice within one budget Q_s, similarly to [5], [17], [9] we assume[3] $2P_s \leq T_s^{\texttt{min}}$.

Example 1: Consider a component C_2 with a local fixed-priority scheduler and with two tasks τ_{21} and τ_{22}. Task τ_{22} accesses a global shared resource R_l and τ_{21} is independent, so that the local resource ceiling $rc_{sl} = \pi_{22}$. Now the following scenario can happen:

1) task τ_{22} starts its execution and upon its attempt to access resource R_l, it encounters insufficient remaining budget to fit a processor request of X_{22l} time units. Task τ_{22} therefore initiates self-blocking.
2) a high priority task τ_{21} arrives just after budget depletion; Hence, it starts executing as soon as the component's budget is replenished and becomes available.
3) After τ_{21} has finished its execution, the remaining budget is again insufficient to fit X_{22l} time units.

The scenario is illustrated in Figure 2 and the condition $2P_s \leq T_s^{\texttt{min}}$ prevents this scenario.

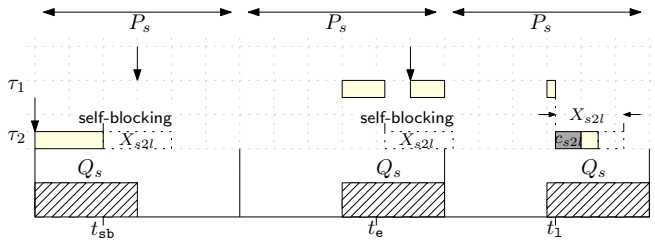

Figure 2. SIRAP disallows that task τ_2 self-blocks two times to prevent budget overruns before access to resource R_l is granted.

Alternatively to constraining the budget period P_s, additional budget could be allocated to service these pushed-through jobs together with the jobs accounted for in the resource holding time [20]. In [20], it has been shown that one can trade-off the amount of compensating budget versus the amount of worst-case local self-blocking by refining SIRAP with an additional local resource ceiling to regulate preemptions during self-blocking.

Note that it is analytically unattractive for SIRAP to lift the requirement of executing a critical section upon replenishment immediately after self-blocking. In that case, we would potentially have to account for multiple subsequent self-blocking occurrences for a single resource access. Contrary to SIRAP, BROE - which also uses a self-blocking mechanism - does not require resource access in the next budget replenishment after the first self-blocking occurrence.

C. BROE: component-level self-blocking

Bertogna et al. [6] have proposed an alternative method of self-blocking compared to SIRAP - called BROE - which uses EDF scheduling of tasks and components. An analysis for BROE under task-level FPPS is presented

[2]The actually consumed amount of processor time is per definition smaller than or equal to the worst-case resource holding time X_{sil}.

[3]It has been shown in [1] that this assumption allocates an efficient budget for a periodic resource model. Moreover, for relatively small budget periods compared to task periods, the bounded-delay approximation of the periodic resource model is tighter [18].

in [21]. Although the given analyses are opaque, BROE is restricted to global EDF scheduling of components and the bounded-delay model [2].

Contrary to the other protocols, BROE's resource-sharing overhead is left implicit in its local analysis, because the bounded-delay resource model models the processor supply to a component sufficiently pessimistic. It is therefore unnecessary to account for the largest overrun of each task, as with HSRP, and BROE refrains from idling the processor to prevent budget overruns, as with SIRAP. A comparison of different synchronization protocols is therefore biased by the underlying resource model.

BROE uses a *hard constant bandwidth server* (H-CBS) [22] to provide its allocated processor bandwidth to a component C_s. Apart from period P_s and maximum budget Q_s, defining its utilization $U_s = \frac{Q_s}{P_s}$, at each time t a H-CBS is characterized by an absolute server deadline $d_s(t)$ and a remaining budget $Q_s^{\texttt{rem}}(t)$. Like with other servers, all pending jobs are contending for processor resources at the server's deadline $d_s(t)$ and whenever a job executes, the budget $Q_s^{\texttt{rem}}(t)$ is decreased by the received execution time of that job. The rules (1-5) of a BROE server, with respect to the current time t, are as follows [6]:

1) Initially, $Q_s^{\texttt{rem}}(0) = 0$ and $d_s(0) = 0$.

2) When a new job of a task τ_i arrives at time t, if the server is idle and if $Q_s^{\texttt{rem}}(t) \geq (d_s(t) - t)U_s$, then the server budget is replenished to the maximum value Q_s and the server deadline is set to $d_s(t) \leftarrow t + P_s$.

3) Let $t_r = d_s(t) - \frac{1}{U_s} Q_s^{\texttt{rem}}(t)$. When a new job of a task τ_i arrives at time t, if the server is idle and if $t < t_r$, then the server budget is suspended until time t_r. At time t_r the server budget is replenished to the maximum value Q_s and the server deadline is set to $d_s(t) \leftarrow t_r + P_s$.

4) When $Q_s^{\texttt{rem}}(t) = 0$, the server is suspended until time $d_s(t)$, so that pending jobs cannot contend for processor resources during time interval $[t,\ d_s(t)]$. At time $d_s(t)$, the server budget is replenished to the maximum value Q_s and the server deadline is set to $d_s(d_s(t)) \leftarrow d_s(t) + P_s$.

5) Whenever a pending task wishes to access a global resource R_l at a time t, it must perform a budget check. I.e., if the remaining budget $Q_s^{\texttt{rem}}(t) \geq X_{sl}$, then there is enough budget to complete the resource access prior to server deadline $d_s(t)$. Then, the task is granted access to resource R_l. Otherwise, the server will replenish its budget no later than time $t_r \leftarrow d_s(t) - \frac{1}{U_s} Q_s^{\texttt{rem}}(t)$. If $t_r \leq t$, this results in an immediate replenishment of the server budget to the maximum value Q_s and the server deadline is set to $d_s(t) \leftarrow t + P_s$. If $t_r > t$, the server is suspended until time t_r. Next, at time t_r the server budget is replenished to the maximum value Q_s and the server deadline is set to $d_s(t) \leftarrow t_r + P_s$.

Rule 1, 2 and 4 describe a H-CBS, see [22] and [23]. BROE adds Rule 3 to the H-CBS to guarantee a fully replenished budget when the server continues after a duration of idle time. Rule 2 and Rule 3 are mutually exclusive. Rule 2 applies when the amount of remaining budget $Q_s^{\texttt{rem}}(t)$ until the current deadline $d_s(t)$ would require to supply more processor resources in the interval until deadline $d_s(t)$ than the the server utilization U_s. Otherwise, Rule 3 applies, i.e., the supply by the server is still running ahead with respect to its guaranteed processor utilization. Rule 5 adds resource arbitration to the modified H-CBS. For any continuously backlogged interval of length t, i.e., the BROE server has pending jobs, BROE behaves as a conventional H-CBS extended with resource arbitration (Rule 5). A request to access a global shared resource only causes a server self-suspension if there is insufficient budget to complete the critical section and if - similar to Rule 3 - the supplied budget by the server is running ahead with respect to its guaranteed processor utilization.

Although it has been shown by Kumar et al. [23] that a conventional H-CBS complies to the periodic resource model, Example 2 shows BROE's pessimism compared to a conventional H-CBS at both the local and global scheduling level. Firstly, BROE cannot guarantee at least $\texttt{sbf}_{\Gamma_s}(t)$ processor resources to its task set in any interval of length t within a backlogged period. Secondly, there are many possible server deadlines. An important difference of BROE compared to other protocols is that the worst-case processor supply to a component changes dependent on both (*i*) the size of the statically computed resource holding times X_{sl} and (*ii*) the actual time at which a task attempts to access resource R_l. With the other protocols the processor supply merely changes dependent on the size of the statically computed X_{sl} values. The latter indicates an infinite amount of possibilities for absolute deadlines within a backlogged server period. The lack of a finite set of server deadlines complicates an integration of BROE servers into the HSF by using Baruah's [24] enhanced demand-bound test for EDF. Bertogna et al. [6] have therefore proven a sufficient utilization-based integration test.

We conclude that a BROE server is non-compliant with the periodic resource supplies of Shin and Lee [1] and Kumar et al [23]. Inherent to the rules of BROE [6], however, the server has a period P_s. Given a period

constraint P_s, BROE's bounded-delay resource model always gives a linear lower bound $\texttt{lsbf}_{\Gamma_s}(t)$ of the actually supplied resources $\texttt{sbf}_{\Gamma_s}(t)$ by a periodic resource Γ_s with the same period and budget parameters [18]. BROE's pessimism, inherited from the resource model, heavily depends on the timing characteristics of tasks and the interface parameters of the comprising component.

V. Compositional 2-level analysis

This section recapitulates the existing compositional analysis for BROE, ONP, OWP, and SIRAP. As scheduling algorithms we consider EDF, an optimal dynamic scheduling algorithm, and the deadline-monotonic (DM) algorithm, an optimal FPPS algorithm.

A. Global schedulability analysis

To integrate a set of components on a shared processor, we must characterize the worst-case processor requests by each component. This depends on the chosen global synchronization protocol. We therefore assume that *during component-integration time* the synchronization protocol is known.

The following sufficient schedulability condition holds for global EDF-scheduled systems [24]:

$$\forall t > 0 \quad : \quad B(t) + \texttt{DBF}(t) \leq t. \tag{6}$$

The blocking term, $B(t)$, is defined as [24]:

$$B(t) = \max\{X_{ul} \mid \exists s : R_l \in \mathcal{R}_u \cap \mathcal{R}_s \wedge P_s \leq t \wedge P_u > t\}. \tag{7}$$

The demand bound function $\texttt{DBF}(t)$ computes the total processor demand of all components in the system for every time interval of length t, i.e.,

$$\texttt{DBF}(t) \quad = \quad \sum_{C_s \in \mathcal{C}} \left\lfloor \frac{t}{P_s} \right\rfloor (Q_s + O_s). \tag{8}$$

A component C_s, using ONP for global resource sharing, demands $O_s = X_s$ more resources in its worst-case scenario [10]; for SIRAP $O_s = 0$. For OWP, the $\texttt{DBF}(t)$ is slightly modified:

$$\texttt{DBF}(t) \quad = \quad \sum_{C_s \in \mathcal{C}} \left(O_s(t) + \left\lfloor \frac{t}{P_s} \right\rfloor Q_s \right), \tag{9}$$

where the extra demand of $O_s(t)$ is [10]:

$$O_s(t) = \begin{cases} X_s & \text{if } t \geq P_s \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

A global admission of EDF-scheduled components with the analysis in [24] is inapplicable to BROE. Consequently, BROE has modified [24] and applies a sufficient utilization-based test [6].

For global FPPS of components - by definition disallowing BROE - the following sufficient schedulability condition holds:

$$\forall 1 \leq s \leq N : \quad \exists t \in (0, P_s] : \quad \texttt{RBF}(t, s) \leq t, \tag{11}$$

where $\texttt{RBF}(t, s)$ denotes the worst-case cumulative processor request of C_s and all higher priority components for a time interval of length t. For SIRAP and ONP, the $\texttt{RBF}(t, s)$ is defined as follows:

$$\texttt{RBF}(t, s) = \quad B_s + \sum_{1 \leq r \leq s} \left\lceil \frac{t}{P_r} \right\rceil (Q_r + O_r). \tag{12}$$

For OWP, the $\texttt{RBF}(t, s)$ is slightly modified:

$$\texttt{RBF}(t, s) = \quad B_s + \sum_{1 \leq r \leq s} \left(O_r + \left\lceil \frac{t}{P_r} \right\rceil Q_r \right). \tag{13}$$

In (12) and (13), again $O_r = X_r$ for ONP and OWP and $O_r = 0$ for SIRAP. The blocking term, B_s, is defined according to [7]:

$$B_s = \max\{X_{ul} \mid \Pi_u < \Pi_s \leq RC_l\}. \tag{14}$$

B. Local schedulability analysis

This section distinguishes opaque and non-opaque local schedulability analyses under various global synchronization protocols.

1) Opaque local analysis: Traditional protocols such as PCP [25] and SRP [7] can be used for *local* resource sharing in HSFs [26]. With an opaque local analysis, we can re-use the same local analysis in the presence of global shared resources. By filling in task characteristics in the demand bound DBF of (6) or the request bound RBF of (11) and replacing their right-hand sides by (1), i.e. replace t by $\mathtt{sbf}_{\Gamma_s}(t)$, the same schedulability analysis holds for tasks executing within a component as for components at the global level. Due to space constraints, we focus on local FPPS of tasks for which the following sufficient schedulability condition holds:

$$\forall 1 \leq i \leq n_s : \exists t \in (0, D_{si}] : \quad \mathtt{rbf}_s(t,i) \leq \mathtt{sbf}_{\Gamma_s}(t), \tag{15}$$

where $\mathtt{rbf}_s(t,i)$ denotes the worst-case cumulative processor request of τ_{si} for a time interval of length t. In Section VII we shall show that BROE's analysis requires to replace the $\mathtt{sbf}_{\Gamma_s}(t)$ with $\mathtt{lsbf}_{\Gamma_s}(t)$. For BROE, ONP and our new OWP analysis, the $\mathtt{rbf}_s(t,i)$ is fully compliant to the schedulability analysis for task sets on a dedicated unit-speed processor, i.e.,

$$\mathtt{rbf}_s(t,i) \quad = \quad b_{si} + \sum_{1 \leq j \leq i} \left\lceil \frac{t}{T_{sj}} \right\rceil C_{sj}. \tag{16}$$

The blocking term, b_{si}, is defined according to [7]:

$$b_{si} = \max\{h_{sjl} \mid \pi_{sj} < \pi_{si} \leq rc_{sl}\}. \tag{17}$$

2) Non-opaque local analysis: The local analysis of a component under resource arbitration by OWP or SIRAP does not regard global resources as local.

A component using SIRAP demands more resources in its worst-case scenario [9]. We therefore need to add a term, $I_{si}(t)$, to account for *self-blocking* to the $\mathtt{rbf}_s(t,i)$. 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 $G_{si}^{\mathtt{sort}}(t)$ 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 $G_{si}^{\mathtt{sort}}(t)$ 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. Supplemental to our evaluation and proofs, the Appendix shows how to construct such a multi-set.

According to [17] and [10], OWP has additional pessimism at the local scheduling level compared to overrun without payback (ONP). They have therefore modified the $\mathtt{sbf}(t)$ compared to the definition given in (1), see [10]. 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?

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

We reconsider the problem of resource sharing across budgets. Ghazalie and Baker [3] 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 [4], [10] 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 EDF and FPPS of components as defined in (6) and (11).

We now need to characterize the worst-case resource supply to the tasks serviced by component C_s. Behnam et al. [10] distinguish two cases to represent the worst-case processor supply, see Figure 3. 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 3(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 $BD_s = 2(P_s - Q_s) + X_s$.

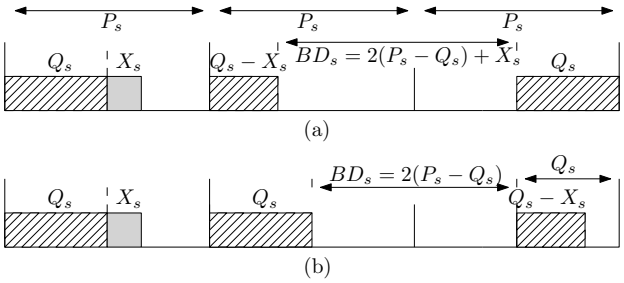

Figure 3. Worst-case characterization of the periodic processor supply for SRP with mechanisms for overrun and payback, as presented in [10].

Alternatively, the component may overrun its budget again in the second period, see Figure 3(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. This scenario has a smaller worst-case processor blackout of $BD_s = 2(P_s - Q_s)$.

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 (13). Furthermore, the schedulability test in (11) 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 4.

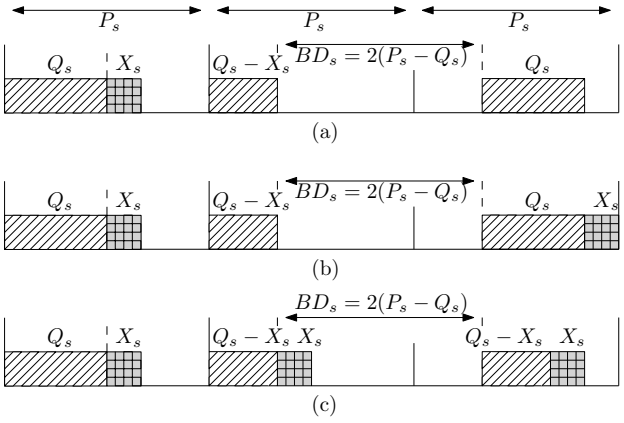

Figure 4. The latest starting time of the processor supply in each period is independent of whether or not an overrun takes place in that period.

We can now derive the following lemma:

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 4, 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 for the normal budget Q_s, so that the blackout for two consecutive budget supplies is at most $BD_s = 2(P_s - Q_s)$. ■

Contrary to ONP, we cannot make the implicit deadline $P_s - X_s$ of budget Q_s explicit for OWP by applying the EDP model [27], because this would further reduce the blackout duration to $BD_s = 2(P_s - Q_s) - X_s$, see Figure 5. Although this is obviously optimistic for OWP, this explicit deadline improves the local analysis of ONP [11]. This improved ONP (IONP) analysis is non-opaque, because it uses resource holding times to tighten the local analysis.

The result of Lemma 2 is the same as the analysis derived by Davis and Burns [4], although they do not support a compositional analysis. Behnam et al. [10] came up with an improved OWP method - called *enhanced overrun* - to improve the blackout duration assumed by their initial OWP analysis, see Figure 6. They improve their analysis

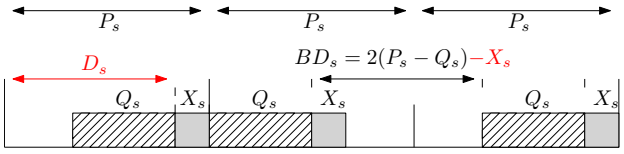

Figure 5. Since budget Q_s must be provisioned prior to deadline $P_s - X_s$, the EDP resource model [27] enables a tighter, non-opaque ONP analysis.

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 $2P_s$.

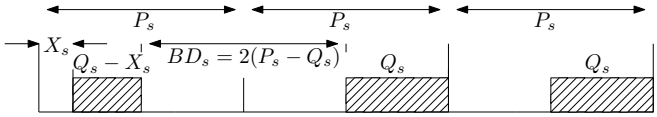

Figure 6. In [10] the extra blackout due to payback is reduced by introducing a flexible release off set for budget $Q_s - X_s$, i.e. the initial delay of X_s.

The latter source of pessimism is inherited from the analysis by Davis and Burns [4], which considers the effect of *push-through blocking* due to an overrun with payback. This effect is shown in Figure 4(c), where a task arrives just after depletion 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 also recall 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. [10] 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 3. We will show that the model in [10] 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 4 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. Otherwise, the resource supply is unchanged compared to the $\texttt{sbf}_{\Gamma_s}$ 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 ($\pi_{sj} > rc_{sl}$), tasks that are blocked during the critical section ($rc_{sl} \geq \pi_{sj} > \pi_{si}$), the resource-locking task τ_{si} itself ($\pi_{si} = \pi_{sj}$) and tasks that have a lower priority than the resource-locking task ($\pi_{si} > \pi_{sj}$).

1) $\pi_{sj} > rc_{sl}$: 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 [10], 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 $\pi_{sj} > rc_{sl}$ will miss a deadline due to a payback.

2) $rc_{sl} \geq \pi_{sj} > \pi_{si}$: 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 is that the blocking duration of at most X_{sl} is already accounted in the $\texttt{rbf}_s(t, j)$ 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) $\pi_{si} = \pi_{sj}$: 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) $\pi_{si} > \pi_{sj}$: these tasks have a lower priority than the resource-locking task and have already accounted X_{sl} as interference in their $\texttt{rbf}_s(t, j)$. 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 OWP mechanism does therefore not cause a deadline miss to any task τ_{sj} where $\pi_{si} > \pi_{sj}$.

By contradiction we have proven 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 1: The local schedulability analysis in (15) 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 (15).

Finally, we answer the main question of this section: to payback or not to payback? 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 an opacity perspective.

VII. A DESIGN METHODOLOGY

In this section, we propose a two-step approach for constructing component interfaces using opaque local analysis. Firstly, one must select a resource model, which may significantly impact the allocated budget to a component - even if a component shares no global resources. In the presence of global shared resources, a synchronization protocol may limit the choice of a resource model.

Secondly, a local resource ceiling must be selected for each shared resource. Contrary to local shared resources, for global resources an artificial increase of the local resource ceiling compared to (4) may improve schedulability. For example, Davis and Burns [4] disable all local preemptions during global resource access. On the one hand, an explicit assumption on local resource ceilings affects the local analysis non-opaquely, because a component

gives up its local view on resource sharing. On the other hand, since opacity allows to defer the choice of a global synchronization protocol until system integration, binding of synchronization primitives may come with globally selected local ceilings.

A. Choosing a resource model

Each of the global synchronization protocols considered in this paper has a period constraint P_s. For SRP with an overrun mechanism, the period P_s serves as a relative deadline for completion of a resource access. For SIRAP and BROE, the period P_s also bounds the waiting time of a task that wishes to access a global resource.

Given a period constraint P_s, the bounded-delay resource model gives a linear lower bound $\texttt{lsbf}_{\Gamma_s}(t)$ of the actually supplied resources by a periodic resource $\texttt{sbf}_{\Gamma_s}(t)$ with the same period parameter [18]. For this reason, the schedulability analysis for OWP, ONP and SIRAP using the bounded-delay model is sufficient but pessimistic. BROE is non-compliant with the periodic resource model [1], which is its weakness compared to the other protocols.

Example 2: Consider component C_1 with a period $P_s = 10$ and two tasks: $\tau_{11} = (1000, 2, 29)$ and $\tau_{12} = (1000, 1, 1000)$. Task τ_{11} accesses a global resource R_1 for a duration of $h_{111} = X_{11} = 0.5$ time units; task τ_{12} is independent. The smallest budget satisfying the local schedulability condition in (15) is $Q_s = 1$, yielding an interface $\Gamma_1^{(\texttt{PRM})} = (10, 1, \{0.5\})$. Without any global resource sharing, the required processor bandwidth of component C_1 is therefore 0.1, see Figure 7(a). When arbitrating resource R_1 with BROE, however, a budget of $Q_1 = 1$ is insufficient, see Figure 7(b). According to BROE's bounded-delay criteria, i.e., using (2), component C_1 requires a budget of $Q_s = 1.63$. The corresponding bounded-delay interface $\Gamma_1^{(\texttt{BDM})} = (10, 1.63, \{0.5\})$ yields a bandwidth of 0.163.

To compare: arbitrating resource R_l with ONP or OWP would allocate an overrun budget of 0.5 time units, so that the allocated processor bandwidth for C_1 becomes 0.15. In this example, BROE requires more processor bandwidth than ONP, OWP or - by virtue of Theorem 2 - SIRAP.

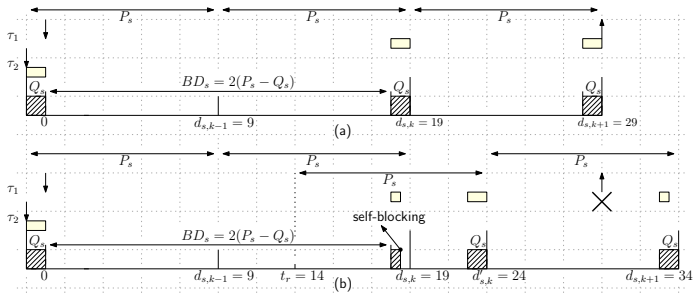

Figure 7. The periodic resource model is inapplicable to BROE.

One could construct an interface for a component using the periodic resource model and convert it to a bounded-delay interface when BROE is elected for global resource arbitration. An interface $\Gamma_s = (P_s, Q_s, \mathcal{X}_s)$, computed according to (15), represents a virtual task $\tau' = (P_s, Q_s, P_s)$. By applying the bounded-delay abstraction using the $\texttt{lsbf}_{\Gamma'_s}(t)$ on the virtual task τ', one can derive a conservative budget Q'_s which $\forall t \geq 0$ upper bounds the periodic supply $\texttt{sbf}_{\Gamma_s}(t)$. According to the method in [18], Q'_s is found by:

$$Q'_s = \frac{-(Y - 2P_s) + \sqrt{(Y - 2P_s)^2 + 8P_sQ_s}}{4}, \tag{18}$$

where $Y = 2P_s - Q_s$.

Reconsidering Example 2: applying the bounded-delay criteria on the periodic resource $\Gamma_1^{(\texttt{PRM})} = (10, 1, \{0.5\})$ gives a conservative budget of $Q'_s = 2.5$ time units. Although this method of converting interfaces allows a component to be be analysed with an arbitrary resource model, the derived interface suffers abstraction overheads of two resource models. It is therefore unattractive to convert a resource model at the interface level.

A more processor-efficient solution is to delay the choice of a resource model by deriving two interfaces for each component, $\Gamma_s^{(\texttt{PRM})}$ and $\Gamma_s^{(\texttt{BDM})}$, i.e., one interface for each resource model. Upon component integration, we select an interface based on the global synchronization protocol. With both solutions, the notion of an opaque local analysis for a component is independent of a resource model. In this paper we implicitly apply the latter approach.

B. Choosing local resource ceilings

Global resource ceilings are optimally configured according to SRP, see (3). This is irrespective of whether the global scheduling policy is FPPS or EDF, because higher resource ceilings incur more blocking and lower resource ceilings violate mutual exclusive access to a shared resource. Within the hierarchy of an HSF, however, the local resource ceilings in (4) require the smallest budget, but may introduce large resource holding times X_{sl} and, hence, large blocking terms for other components in the system.

Each component exposes the maximum resource holding times, $X_{sl} \in \mathcal{X}_s$, of an unspecified access to resource R_l in its interface specification. The local resource ceiling rc_{sl} of resource R_l can have at most n_s possible values, leading to different values for resource holding time X_{sil} and its derivative X_{sl}. In general, each of the m^{n_s} combinations yields a possible interface $(P_s, Q_s, \mathcal{X}_s)$ - called an *interface candidate*. It is therefore unattractive to explore every combination of interface candidates of composed components.

Only during integration time, however, one can actually compute the global blocking. For example, if resource R_l is not shared by any other component, then resource R_l is a local shared resource. Selecting an optimal interface for a component, i.e., leading to the least amount of required processor resources by the entire system, therefore also depends on the interfaces of other components.

A nice property of opacity is that it enables to compute a set of interface candidates which lead to a polynomial procedure for Pareto-optimal interface selection [28]. On the one hand, the lowest local resource ceiling rc_{sl} imposes the least local blocking b_{si} and therefore minimizes the required budget of a component. On the other hand, a higher local resource ceiling rc_{sl} decreases the resource holding time, see (5), and therefore imposes less global blocking B_s to other components.

Using this Pareto trade-off between the values of the local resource ceilings and the resource holding times, Shin et al. [28] have shown that with ONP there are only n_s non-redundant interfaces per component which can be generated in $\mathcal{O}(n_s)$ iterations. At integration time, Shin et al. [28] trade the global blocking for more budget for the component that induces most blocking. Under global EDF, a similar selection method has been presented for BROE [21]. The interface selection procedures take $\mathcal{O}(N \times n_s^{\mathtt{max}})$ steps, where $n_s^{\mathtt{max}} = \max\{n_s \mid 1 \leq s \leq N\}$.

With a non-opaque analysis, the methods in [28] and [21] may be unable to optimally synthesize a system with respect to required processor resources. SIRAP's analysis, for example, may allocate a smaller budget when resource ceilings are higher, so that the resource holding times are smaller. This exponentially increases the search space for optimal interfaces.

Using an opaque analysis, one may efficiently select an interface that minimizes the processor requirements of a system. Although the selected interface may be non-optimal beyond the scope of the applied analysis method, one may further tighten the system's analysis by applying non-opaque local analysis onto the selected local configuration. We recognized that the methods in [28] and [21] for selecting optimal interface candidates can be applied to any opaque local analysis, independent of a global synchronization protocol.

C. Applying opaque local analysis to SIRAP

SIRAP periodically bounds the wasted processor resources due to global resource sharing. Its analysis benefits therefore more from the periodic resource model than from the bounded-delay model. But, we still face the problem of selecting local resource ceilings.

With SIRAP's analysis [9] one must know the amount of accesses to any global resource by each individual job. Although this is unnecessary for HSRP and BROE, it makes SIRAP superior to ONP in case each of those resources are actually shared with at least one other component.

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 $\mathcal{T}_s$.

Since each element in the set $G_{si}^{\mathtt{sort}}(t)$ is at most of length X_s, ONP only performs equally well when a self-blocking of approximately X_s is deducted in each component period. SIRAP is therefore always superior to ONP, so that the ONP analysis can be safely used to implement a SIRAP system.

Theorem 2: 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 [9]:

$$\forall \tau_i \in \mathcal{T}_s : \exists t \in (0, D_{si}] : \mathtt{rbf}_s(t,i) + I_{si}(t) \leq \mathtt{sbf}_{\Gamma_s}(t), \tag{19}$$

where $\mathtt{rbf}_s(t,i)$ is defined in (16), $\mathtt{sbf}_{\Gamma_s}(t)$ 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 (19) is implied by:

$$\forall \tau_i \in \mathcal{T}_s : \exists t : \quad \mathtt{rbf}_s(t,i) + \left\lceil \tfrac{t}{P_s} \right\rceil X_s \leq \mathtt{sbf}_{\Gamma_s}(t). \tag{20}$$

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:

$$X_s + \left(\min Q_s : \left(\forall \tau_i \in \mathcal{T}_s : \exists t : \mathtt{rbf}_s(t,i) \leq \mathtt{sbf}_{\Gamma_s}(t)\right)\right). \tag{21}$$

The right-hand term of (21) is the same as schedulability condition for ONP, see (15), which concludes our proof. ■

Given Theorem 2, we make it possible to integrate a component validated by an opaque analysis for SRP+FPPS into the HSF, while using SIRAP for global resource arbitration. This allows to re-use the methods in [28] and [21] to select local resource ceilings efficiently. When only a subset of the resources in the component's interface are identified as globally shared, one may recompute the value of X_s and re-allocate a potentially tighter budget $Q_s + X'_s$ without re-analysing the component; one would have to re-do a non-opaque local analysis.

VIII. Evaluation

This section evaluates analysis methods for global resource sharing. From the results, we derive which method matches the best with given system characteristics.

In our experiments, we choose a system utilization U and we generate individual component utilizations $U(\mathcal{T})$ using UUnifast [29]. The period of a component is uniformly drawn from the interval $[40, 70]$. We assume global EDF scheduling of components and a single non-preemptively shared global resource by all components.

Given a cumulative component utilization $U(\mathcal{T})$, we generate $n_s = 8$ tasks for each component. The task periods T_{si} are uniformly drawn from the interval $[140, 1000]$. We initially 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 [29]. 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 experimental setting a new set of 10,000 systems is generated.

A. Feasibility of task sets in the presence of global resources

We first investigate for which task-characteristics a particular analysis method is better, i.e. at the component level. We look at the percentage of schedulable task sets, generated according to the description in Section VIII.

In each simulation study a new set of 10,000 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 8.
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 9.

For comparison purposes we included the results for the improved local analysis of ONP [11], 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 and BROE, we require that $X_s \leq Q_s$. Due to this constraint, both SIRAP's and BROE's performance are suppressed for small resource periods. BROE may require a larger budget for a component, because it must use the bounded-delay model. In terms of the schedulability ratio, however, BROE clearly outperforms the other protocols (see Figure 8). In addition, both figures show the cost of an opaque analysis in the context of two-level FPPS-based HSFs, which excludes BROE.

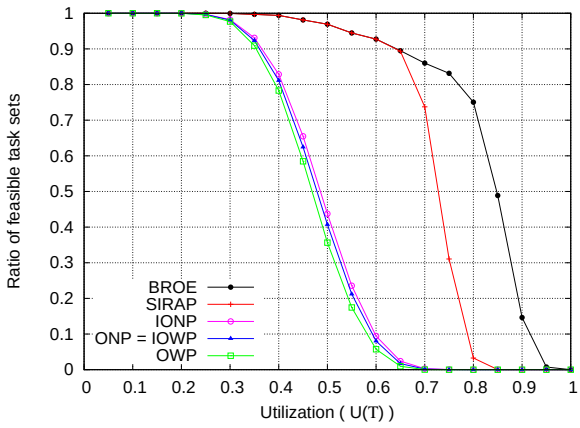

Figure 8. Ratio of schedulable task sets versus the utilization, where the component period is $P_s = 40$ and the number of tasks is $n_s = 8$.

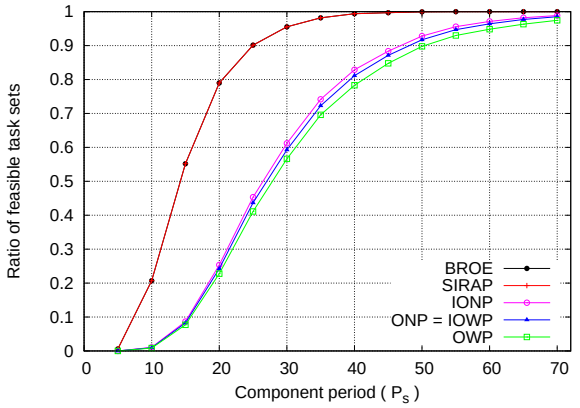

Figure 9. Ratio of feasible task sets as a function of the component period, where the number of tasks is $n_s = 8$ and the utilization $U(\mathcal{T}) = 0.4$.

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 8), choosing smaller resource periods (Figure 9), 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 [11] 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 (12) and (13) 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.

B. Global scheduling penalties for global synchronization

In this section, we compare the analyses at the compositional level, because at the local level - especially with opaque analysis - the resource model may hide scheduling penalties.

We observed that BROE is superior in terms of the number of task sets that can be accommodated, because BROE does not need additional overrun budget and it does not insert idle time. However, these results ignore the required processor bandwidth by a single component. The bounded-delay model, exclusively applied to BROE, performs relatively bad compared to the periodic resource model when the utilization of a component $U(\mathcal{T})$ is small. A solution would be to reduce the period P_s of a component. However, the period size cannot be decreased arbitrarily, because an entire critical section must fit within one period.

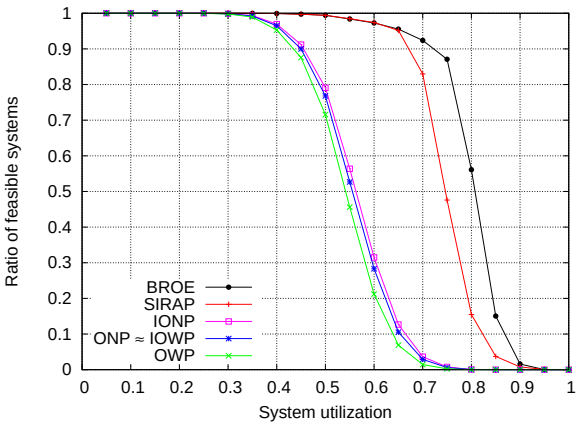

Figure 10. Ratio of schedulable systems versus the system utilization, where the number of components is $N = 2$ and all tasks have $D_{si} = T_{si}$.

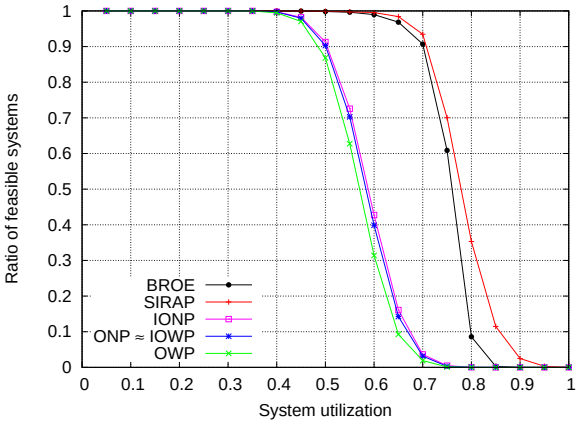

Figure 11. Ratio of schedulable systems versus the system utilization, where the number of components is $N = 5$ and all tasks have $D_{si} = T_{si}$.

In the first experiment, we investigate how composing multiple resource-sharing components affects the number of schedulable systems. Figure 10 shows the results for $N = 2$ components and Figure 11 for $N = 5$ components. When the utilization of a single component is relatively large, i.e., $U(\mathcal{T}) \gtrsim 0.1$, BROE clearly outperforms all other protocols. For smaller utilizations, SIRAP becomes advantageous. 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[4].

In the second experiment, we repeated the same experiment for $N = 5$ components and we randomly generated tasks with deadlines $D_{si} \leq T_{si}$, uniformly drawn from the range $[C_{si} + 0.5(T_{si} - C_{si});\ T_{si}]$. Figure 12 reports the results. Compared to the first experiment, the bounded-delay model further reduces the performance of BROE. Intuitively, postponing budget supply to a task set, being subject to tight deadline constraints, deflates BROE's performance compared to the non-opaque analysis of SIRAP. However, BROE's performance is considerably better than any overrun variant.

In the third experiment, the system utilization $U = 0.5$ and the range of task deadlines $D_{si} \leq T_{si}$ are fixed. The number of components, N, is varied within a range of $[1, 14]$, see Figure 13. Composing a system of many resource-sharing components, e.g., the operating system itself can be a single point of synchronization, may significantly decrease the number of schedulable systems. It is interesting to see that BROE covers the entire performance spectrum compared SIRAP, ONP and OWP: from a superior performance for components with large individual utilizations, to an inferior performance for small component utilizations.

[4]Moreover, the shorter the component period is, the higher context switching overhead will be. The implementation overhead of the synchronization protocols is not considered in this evaluation, however, and it is different for each protocol.

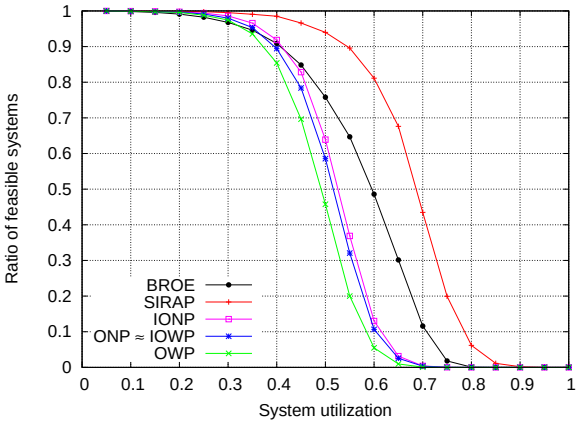

Figure 12. Ratio of schedulable systems versus the system utilization, where the number of components is $N = 5$ and tasks may have $D_{si} \leq T_{si}$.

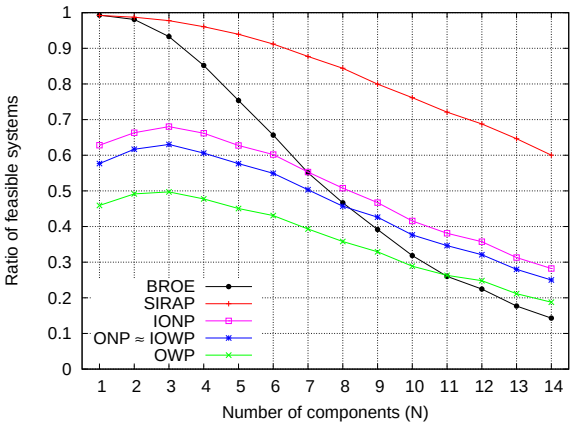

Figure 13. Ratio of schedulable systems versus the number of components, where the system utilization is $U = 0.5$ and tasks may have $D_{si} \leq T_{si}$.

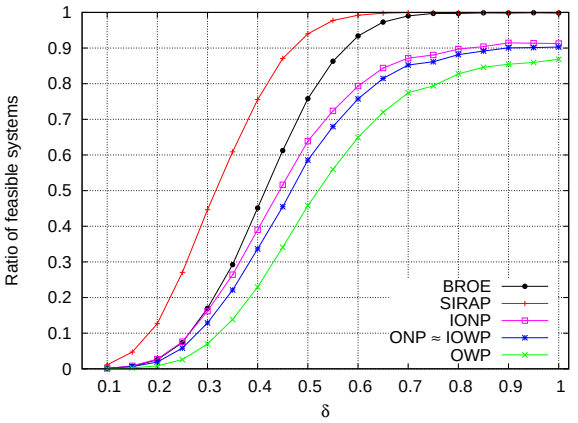

Figure 14. Ratio of schedulable systems versus the deadline distribution of tasks, where the number of components is $N = 5$ and the system utilization is $U = 0.5$.

In the fourth experiment, we keep a system utilization of $U = 0.5$ for $N = 5$ components. We vary the range of the task deadlines using parameter δ. We generated task deadlines uniformly drawn from the range $[C_i + \delta(T_i - C_i);\ T_i]$. A low value of δ allows tasks to have short deadlines relative to their computation time and $\delta = 1$ means that deadlines are equal to periods. Figure 14 shows the results. This experiment confirms the previous experiments: SIRAP's non-opaque analysis is beneficial for deadline-constrained tasks, while HSRP's overrun and BROE perform equally worse under tight deadline constraints.

Finally, both SIRAP and BROE have shown to be more resilient than HSRP's overrun variants for relatively large critical section lengths compared to a component's budget. For SIRAP, the analysis for a single shared resource by each task performs relatively bad, because the more individual resource accesses are considered, the better its analysis. BROE and overrun have opaque analysis, i.e., only based on local SRP. Sharing more global resources would therefore not affect much the performance of opaque analysis, but it might further improve the benefits of non-opaque analysis.

IX. Recommendations and remarks

This paper introduced the notion of *opaque* analysis for resource-sharing components that need to be integrated on a uni-processor platform. An opaque analysis makes it possible to abstract from global resource sharing until component integration and it enables an efficient exploration of a system's design space. Although SIRAP's original analysis is non-opaque, we can use the analysis of overrun without payback (ONP) as a conservative and opaque alternative. We can obtain a tighter schedulability analysis with SIRAP's analysis, if we are provided a task-set's global resource-sharing information. We also presented 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. Finally, we showed that BROE's analysis is opaque and, in many situations, is competitive with SIRAP's non-opaque analysis. If a system is composed of components with tight deadlines or if it is composed of many components having small utilizations, a non-opaque analysis may significantly improve schedulability.

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] X. Feng and A. Mok, "A model of hierarchical real-time virtual resources," in *Real-Time Syst. Symp.*, Dec. 2002, pp. 26–35.

[3] 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.

[4] R. Davis and A. Burns, "Resource sharing in hierarchical fixed priority pre-emptive systems," in *Real-Time Systems Symp.*, 2006, pp. 257–267.

[5] 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.

[6] 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.

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

[8] 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.

[9] M. Behnam, T. Nolte, and R. J. Bril, "Bounding the number of self-blocking occurrences of SIRAP," in *Real-Time Systems Symp.*, Dec. 2010, pp. 61–72.

[10] 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.

[11] 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 Syst.*, April 2011.

[12] Z. Deng and J.-S. Liu, "Scheduling real-time applications in open environment," in *Real-Time Systems Symp.*, Dec. 1997, pp. 308–319.

[13] 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.

[14] G. Lipari and S. Baruah, "Efficient scheduling of real-time multi-task applications in dynamic systems," in *Real-Time Technology and Applications Symp.*, May 2000, pp. 166–175.

[15] M. Caccamo and L. Sha, "Aperiodic servers with resource constraints," in *Real-Time Syst. Symp.*, 2001, pp. 161–170.

[16] P. Holman and J. Anderson, "Locking in pfair-scheduled multiprocessor systems." in *Real-Time Systems Symp.*, Dec. 2002, pp. 149–158.

[17] 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.

[18] G. Lipari and E. Bini, "A methodology for designing hierarchical scheduling systems," *Journal of Embedded Computing*, vol. 1, no. 2, pp. 257–269, 2005.

[19] M. Bertogna, N. Fisher, and S. Baruah, "Static-priority scheduling and resource hold times," in *Parallel and Distributed Processing Symp.*, 2007.

[20] M. Behnam, T. Nolte, and R. J. Bril, "Refining SIRAP with a dedicated resource ceiling for self-blocking," in *Conf. on Embedded Software*, 2009, pp. 157–166.
[21] M. Behnam, T. Nolte, and N. Fisher, "On optimal real-time subsystem-interface generation in the presence of shared resources," in *Conf. on Emerging Technologies and Factory Automation*, Sept. 2010.
[22] L. Abeni, L. Palopoli, C. Scordino, and G. Lipari, "Resource reservations for general purpose applications," *IEEE Trans. on Industrial Informatics*, vol. 5, no. 1, pp. 12–21, Feb. 2009.
[23] P. Kumar, J.-J. Chen, L. Thiele, A. Schranzhofer, and G. Buttazzo, "Real-time analysis of servers for general job arrivals," in *Conf. on Embedded Real-Time Computing Systems and Applications*, Aug. 2011, pp. 251–258.
[24] S. K. Baruah, "Resource sharing in EDF-scheduled systems: A closer look," in *Real-Time Syst. Symp.*, 2006, pp. 379–387.
[25] 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.
[26] 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.
[27] A. Easwaran, M. Anand, and I. Lee, "Compositional analysis framework using EDP resource models," in *Real-Time Systems Symp.*, 2007, pp. 129–138.
[28] I. Shin, M. Behnam, T. Nolte, and M. Nolin, "Synthesis of optimal interfaces for hierarchical scheduling with resources," in *Real-Time Systems Symp.*, Dec. 2008, pp. 209–220.
[29] 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

The SIRAP analysis [9] constructs a multi-set $G_{si}^{\texttt{sort}}(t)$ of self-blocking durations that a task τ_{si} may experience in a time interval of length t. The self-blocking term $I_{si}(t)$ of a task τ_{si} is defined as:

$$I_{si}(t) = \sum_{1 \leq l \leq z(t)} G_{si}^{\texttt{sort}}(t)[l], \tag{22}$$

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 $G_{si}^{\texttt{sort}}(t)$ 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:

$$I_{si}^{low} = \max\{X_{sjl} \mid \pi_{si} > \pi_{sj} \wedge rc_{sl} \geq \pi_{si}\}. \tag{23}$$

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 [9]: $G_{si}(t) = \{I_{si}^{low}\} \cup$

$$\left(\bigcup_{(1 \leq j \leq i)} \bigcup_{\left(1 \leq k \leq \left\lceil \frac{t}{T_{sj}} \right\rceil\right)} \bigcup_{(R_l \in \mathcal{R}_s)} \bigcup_{(1 \leq a \leq m_{sjl})} \{X_{sjl}\} \right). \tag{24}$$

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 $G_{si}^{\texttt{sort}}(t)$. Equation (22) 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 $G_{si}^{\texttt{sort}}(t)$.