# Uniform Interfaces for Resource-Sharing Components in Hierarchically Scheduled Real-Time Systems

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

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/62691

**Abstract**

In literature, several hierarchical scheduling frameworks (HSFs) have been proposed for enabling resource sharing between components on a uni-processor system. Each HSF comes with its own set of composition rules which take into account a specific synchronization protocol for arbitrating access to resources. However, the inventors of these synchronization protocols have also chosen to describe these composition rules with the help of protocol-specific component interfaces. This creates unnecessary framework dependencies on components.

In this chapter, we review existing interfaces and propose uniform interfaces to integrate resource-sharing components into HSFs. For the purpose of computing uniform interfaces, we introduce a local (component-level) timing analysis for components which is independent (or agnostic) of the global synchronization protocol being used for arbitrating access to shared resources. An individual component can therefore be analyzed as if all resources are entirely dedicated to it. Given its interface, the component can then be used with an arbitrary global synchronization protocol. This increases the reusability of a component's timing interface, because the interface can still be fed to protocol-specific composition rules when components are integrated.

**Keywords:** hierarchical scheduling frameworks, resource sharing, component composition, real-time interfaces, synchronization protocol

## 1. Introduction

Hierarchical scheduling frameworks (HSFs) have been developed to enable composition and reusability of real-time components in complex systems, for example, as described by Holenderski et al. [1] for the automotive domain. During the development of such systems, component models have become important in order to separate and structure the development of system parts over engineering teams (or third parties). The increasing system complexity therefore demands a decoupling of (*i*) development and analysis of individual components and (*ii*) integration of components on a shared platform. This decoupling requires component interfaces covering both functional aspects as well as non-functional aspects, such as timing. An HSF supports system composition from a timing perspective because it isolates components by allocating a *processor budget* to each component. 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, e.g., the AUTomotive Open System ARchitecture (AUTOSAR), which more and more specify that an underlying operating system should prevent timing faults in any component to propagate to other components on the same processor.

Independent analysis of components and their integration in HSFs is enabled through a set of composition rules (e.g., as proposed by Shin and Lee [2]). By splitting the timing analysis in complementary parts, one could establish 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 provisioning of the processor budget to a component is defined by a resource-supply model, e.g., the periodic resource model by Shin and Lee [2] or the bounded-delay model by Feng and Mok [3]. These models make it possible to combine the timing constraints of the tasks within a component (typically deadlines) and abstract from the way tasks are locally scheduled. A component can therefore be represented by a single real-time constraint, called *a real-time interface*. Components can be composed into an HSF by combining a set of real-time interfaces, which will treat each component as a single task by itself. This enables reuse of components.

The global scheduling environment (a parent component) can provide more resources to its (child) components than just processor resources. For example, components may 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*. Their local scheduling impact can be easily abstracted by real-time interfaces. A resource that is used in more than one component is a *globally shared resource*.

Any access to a resource (local or global) is assumed to be arbitrated by a synchronization protocol. 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 global usage is determined only upon integration of components into the HSF. Fortunately, the syntax of the primitives for accessing local and global resources can be the same, even though the synchronization protocols are different (e.g., as implemented by van den Heuvel, et al. [4]). 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 in the interface from their use in the implementation. This flexible decoupling of the sharing scope of resources in the application's programming interface is called *opacity* by Martinez et al. [5] and it abstracts whether or not a resource is globally shared in the system.

This chapter presents an extension of this notion of opacity to component analysis and the corresponding derivation of a real-time interface of a component. Opacity requires that the implementation of a component, as well as the way in which interface parameters are derived (the local analysis), are unaware of the global synchronization protocol. In this way, components cannot make use of any knowledge about the constraints and modifications to a component imposed by the global synchronization protocol. By definition of opacity, all computed interface parameters of a component are made independent of a global synchronization protocol.

Based on this observation, we present the following contributions:

- We present a uniform representation of component interfaces and a corresponding opaque analysis to derive these interfaces.
- We survey the existing analyses for components that are assumed to run in HSFs with a particular synchronization protocol. We characterize the opacity compliance of their analyses.

## 2. Related work

In hierarchically scheduled systems, a group of recurring tasks, forming a component, is mapped on a reservation; reservations were originally introduced by Mercer et al. [6] and Rajkumar et al. [7]. We first review existing works on hierarchical scheduling of independent components. Secondly, we lift the assumption on the independence of components, so that tasks may share resources with other tasks, either within the same component or located in other components. This means that resource sharing expands across reservations which calls for specialized synchronization protocols for arbitrating access to resources. Finally, we discuss the extension of real-time interface representations for components requiring access to shared resources through a synchronization protocol.

### 2.1. Timing interfaces of independent real-time components

The increasing complexity of real-time systems led to a growing attention for component-based systems. Deng and Liu [8] therefore proposed a two-level HSF for open systems, where components may be independently developed and validated. The corresponding timing analysis of the HSF has been presented by Kuo and Li [9] for fixed-priority preemptive scheduling (FPPS) and by Lipari and Baruah [10] for earliest-deadline-first (EDF) global schedulers.

Later, the research community identified the challenges of separating the timing analysis of the HSF by means of real-time interfaces for components. A real-time interface separates the component's internals (i.e., its tasks and scheduling policy) from its global resource allocation strategy. Wandeler and Thiele [11] calculate demand and service curves for components using real-time calculus. Shin and Lee [2] proposed the periodic resource model to specify periodic processor allocations to components. The explicit-deadline periodic (EDP) resource model by Easwaran et al. [12] extends the periodic resource model of Shin and Lee [2] by distinguishing the relative deadline for the allocation time of budgets explicitly. The bounded-delay model by Feng and Mok [3] describes linear service curves with a bounded initial service delay.

Many works presented approximated [e.g., 13–15] and exact [e.g., 2,12,14] budget allocations for the bounded-delay and periodic resource models under preemptive scheduling policies. Both Lipari and Bini [14] and Shin and Lee [16] have presented methods to convert the bounded-delay model into a periodic resource model. In our chapter, we extend these models in order to support task synchronization.

### 2.2. Task synchronization in hierarchically scheduled systems

In literature, several alternatives are presented to accommodate resource sharing between tasks in reservation-based systems. de Niz et al. [17] support this in their fixed-priority preemptively scheduled (FPPS) Linux/RK resource kernel based on the immediate priority ceiling protocol (IPCP) by Sha et al. [18]. Steinberg et al. [19] implemented a capacity-reserve donation protocol to solve the problem of priority inversion for tasks scheduled in a fixed-priority reservation-based system. A similar approach is described by Lipari et al. [20] for EDF-based systems and termed bandwidth inheritance (BWI).

BWI regulates resource access between tasks that each have their dedicated budget. It works similar to the priority-inheritance protocol (PIP) by Sha et al. [18], and when a task blocks on a resource, it donates its remaining budget to the task that causes the blocking. BWI does not require a priori knowledge of tasks, i.e., no ceilings need to be precalculated. BWI-like protocols are therefore not very suitable for arbitrating hard real-time tasks in HSFs, because the worst-case interference of all tasks in other components that access global resources needs to be added to a component's budget at integration time in order to guarantee its internal tasks' schedulability also in case budget needs to be donated. This leads to pessimistic budget allocations for hard real-time components. To accommodate resource sharing in HSFs, three synchronization protocols have therefore been proposed based on the stack resource policy (SRP) from Baker [21], i.e., HSRP by Davis and Burns [22], SIRAP by Behnam et al. [23], and BROE by Bertogna et al. [24].

### 2.3. Timing interfaces for resource-sharing components

Global resource sharing in HSFs is often based on the SRP by Baker [21] in order to compute blocking delays in the schedule; these computations follow a similar approach as the SRP, which is then re-used at the various scheduling levels in the HSF. In addition, resource sharing requires scheduling mechanisms which have an impact on the local scheduling of a component. If a task that accesses a globally shared resource is suspended during its execution due to the exhaustion of its (processor) budget, excessive blocking periods can occur which may hamper the correct timeliness of other components (see Ghazalie and Baker [25]). To prevent such budget depletion during global resource access (see **Figure 1**), four synchronization protocols have been proposed. These are based on two general mechanisms to prevent budget depletion during the execution of a task's critical section:

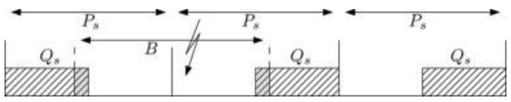

**Figure 1.** When the budget $Q_s$ (allocated every period $P_s$) of a task depletes while a task executes on a global resource, tasks in other components may experience excessive blocking durations, $B$.

1. *self-blocking:* wait with accessing a resource when the remaining budget is insufficient to complete a resource access entirely. Self-blocking comes in two flavors: (*i*) the subsystem integration and resource allocation policy (SIRAP) by Behnam et al. [23] and (*ii*) the bounded-delay resource open environment (BROE) by Bertogna et al. [24]. With SIRAP, a self-blocked task essentially spin locks, i.e., it idles the component's budget away, while the task is waiting for its budget to replenish. Instead, BROE delays the remaining processor's resource supply to a component if there is insufficient budget to complete the entire critical section and if the budget supplied so far is running ahead with respect to the guaranteed processor utilization.

   The idea of self-blocking has also been considered in different contexts, e.g., see [26] for supporting soft real-time tasks and see Holman and Anderson [27] for a zone-based protocol in a pfair scheduling environment. SIRAP by Behnam et al. [23] and BROE by Bertogna et al. [24] use self-blocking for hard real-time tasks in HSFs on a single processor and their associated analysis supports composition. Behnam et al. [28] have significantly improved the original SIRAP analysis by Behnam et al. [23] for arbitrating multiple shared resources; similarly, Biondi et al. [29] have improved the analysis of BROE for arbitrating multiple shared resources. However, these improvements also complicate the analysis and they make the timing analysis more protocol specific.

2. *overrun:* execute over the budget boundary until the resource is released—called the hierarchical stack resource policy (HSRP) by Davis and Burns [22]. 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.

   The overrun mechanism (with payback) was first introduced by Ghazalie and Baker [25] in the context of aperiodic servers. This mechanism was later re-used in HSRP in the context of two-level HSFs by Davis and Burns [22] and complemented with a variant *without* payback. Although the analysis presented by Davis and Burns [22] does not integrate in HSFs due to the lacking support for independent analysis of components, this limitation is lifted by Behnam et al. [30].

Although these four resource-arbitration protocols prevent budget depletion during a task's resource access, in order to do so, 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 of tasks. Protocol developers deal with these constraints in different ways and sometimes these are already taken into account in the local analysis of a component (e.g., see [28–30]). This may therefore result in protocol-specific interfaces of components.

We present a uniform way to model the local constraints on the component's processor supply imposed by resource sharing by extending the periodically constrained model of Feng and Mok [3], as presented for independent components by Shin and Lee [16]. It is therefore important to know which resources a task will access in order to support independent analysis of each of the resource-sharing components. Our local analysis then yields the same timing interface, regardless of the protocol being used for global resource synchronization. During the integration of components, we take those interfaces and we analyze the impact of synchronization penalties with the help of protocol-specific composition rules.

## 3. Real-time scheduling model

A system contains a single processor and a set $\mathcal{R}$ of $M$ resources $R_1, \ldots, R_M$. The processor and (some of) these resources need to be shared by $N$ components, $C_1, \ldots, C_N$, and each component executes its work through a set of (concurrent) tasks (as depicted in **Figure 2**).

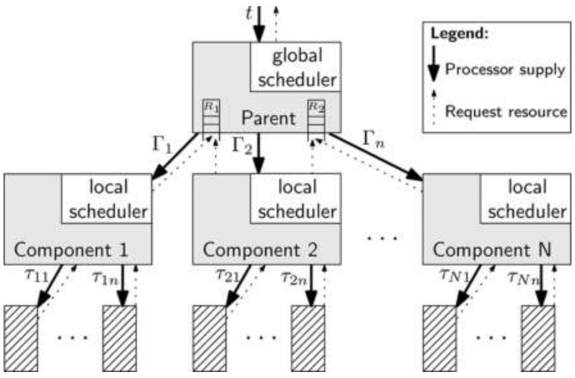

**Figure 2.** Overview of our system model. A parent component implements a global scheduler to allocate a share of the processor and a share of other resources, e.g., $R_1$ and $R_2$, to each of its child components, $C_1 \ldots C_N$. Each child component, $C_s$, contains a set of tasks, $\tau_{s1} \ldots \tau_{sn}$, and a local scheduler. Tasks, located in arbitrary components, may share resources. Tasks receive their share of the resources as specified by their component interface, $\Gamma_s$.

### 3.1. Component and task model

Each component $C_s$ contains a set $\mathsf{T}_s$ of $n_s$ sporadic, deadline-constrained tasks $\tau_{s1}, \ldots, \tau_{sn_s}$. A sporadic task generates an infinite sequence of jobs which are activated at least $T_{si}$ time units separated from each other. Each sporadic job may arrive at an arbitrary moment in time, i.e., it may delay its arrival for an arbitrarily long period. A sporadic task can be seen as a sporadically periodic task which exhibits its worst-case processor demand when subsequent jobs arrive separated minimally in time, i.e., similar to a periodic task under arbitrary phasing (see Liu and Layland [31]).

We extend the timing characteristics of a sporadic task, as introduced by Mok [32], with a parameter to capture its resource requirements. The timing characteristics of a task $\tau_{si} \in \mathsf{T}_s$ are therefore specified by means of a quadruple ($T_{si}$, $E_{si}$, $D_{si}$, $\mathcal{H}_{si}$), where $T_{si} \in \mathrm{IR}^+$ denotes its minimum inter-arrival time, $E_{si} \in \mathrm{IR}^+$ its worst-case execution time (WCET), $D_{si} \in \mathrm{IR}^+$ its (relative) deadline (where $0 < E_{si} \leq D_{si} \leq T_{si}$), and $\mathcal{H}_{si}$ denotes the set of its WCETs of critical sections. The WCET of task $\tau_{si}$ within a critical section accessing global resource $R_\ell$ is denoted $h_{si\ell}$ (i.e., a value contained in $\mathcal{H}_{si}$), where $h_{si\ell} \in \mathbb{R}^+ \cup \{0\}$, $E_{si} \geq h_{si\ell}$. For tasks, we also adopt the basic assumptions by Liu and Layland [31], i.e., jobs do not suspend themselves, a job of a task does not start before its previous job is completed, and the overhead of context switching and task scheduling is ignored. For notational convenience, tasks (and components) are given in deadline-monotonic order, i.e., $\tau_{s1}$ has the smallest deadline and $\tau_{sn_s}$ has the largest deadline.

A task set is said to be schedulable if all jobs of the tasks are able to complete their WCET of $E_{si}$ time units within $D_{si}$ time units from their arrival. The tasks of this component have to meet their deadlines with a particular budget on the processor and each resource being used. These budgets specify the periodically guaranteed fraction of the resource that the tasks may use. The timing interface of a component $C_s$ specifies this budget, i.e., the interface is denoted by a triple $\Gamma_s = (P_s, Q_s, \mathsf{X}_s)$, where $P_s \in \mathrm{IR}^+$ denotes the component's period, $Q_s \in \mathrm{IR}^+$ denotes its budget on the processor, and $\mathsf{X}_s$ denotes the set of resource holding times to global resources (which may be seen as budgets on resources). The maximum value in $\mathsf{X}_s$ is denoted by $X_s$ and, just like any budget, the resource holding time must fit in the components budget: $0 \leq X_s \leq P_s$. The period $P_s$ therefore serves as an implicit deadline of the component.

The set $\mathcal{R}_s$ denotes the subset of global resources accessed by component $C_s$, so that $h_{si\ell} > 0 \Leftrightarrow R_\ell \in \mathcal{R}_s$ and the cardinality of $\mathcal{R}_s$ is denoted by $m_s$ (just like the cardinality of $\mathsf{X}_s$). The maximum time that a component $C_s$ executes on the processor while accessing resource $R_\ell \in \mathcal{R}_s$ is called the resource holding time which is denoted by $X_{s\ell}$, where $X_{s\ell} \in \mathrm{IR}^+ \cup \{0\}$ and $X_{s\ell} > 0 \Leftrightarrow R_\ell \in \mathcal{R}_s$. The relation between the WCET of a critical section ($h_{si\ell}$) and the resource holding times ($X_{s\ell}$) of a component is further explained in Section 4.1.

### 3.2. Scheduling model

A unique system-level (global) scheduler selects which component, and when a component, is executed on the shared processor. The component-level (local) scheduler decides which of the tasks of the executing component is allocated the processor. The global scheduler and each of the local schedulers of individual components may apply different scheduling policies. As scheduling policies, we consider EDF, an optimal dynamic uniprocessor scheduling algorithm, and the deadline-monotonic (DM) algorithm, an optimal priority assignment for FPPS of deadline-constrained tasks. The SRP by Baker [21] is used to arbitrate access to shared resources between components at the global level; similarly, the SRP is used at the local level to arbitrate access to shared resources between tasks locally.

### 3.3. Synchronization protocol

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

#### *3.3.1. Preemption levels*

With the SRP, each task $\tau_{si}$ is assigned a static preemption level equal to $\pi_{si} = 1/D_{si}$. Similarly, we assign a component 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.

#### *3.3.2. Resource ceilings*

With every global resource $R_\ell$ two types of resource ceilings are associated; a *global* resource ceiling $RC_\ell$ for global scheduling and a *local* resource ceiling $rc_{s\ell}$ 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 the SRP, these ceilings are defined as:

$$RC_\ell = \max(\Pi_N, \max\{\Pi_s \mid R_\ell \in R_s\}), \tag{1}$$

$$rc_{s\ell} = \max(\pi_{sn_s}, \max\{\pi_{si} \mid h_{si\ell} > 0\}). \tag{2}$$

We use the outermost max in (1) and (2) to define $RC_\ell$ and $rc_{s\ell}$ in those situations where no component or task uses $R_\ell$. The values of the local and global ceilings as defined in (1) and (2) by definition guarantee mutual exclusive access to their corresponding resource $R_\ell$ by the sharing tasks and components and, therefore, the values of these ceilings cannot be further decreased. In some situations—as further investigated by Shin et al. [33] and van den Heuvel et al. [34]—it might be desirable to limit preemptions more than is strictly required for mutual exclusive resource access, which can be established by increasing the value of the local resource ceilings in (2) artificially. On the contrary, increasing the global ceiling, i.e., the value of $RC_\ell$ in (1), never returns schedulability improvements.

### *3.3.3. System and component ceilings*

The system ceiling and the component ceiling 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 the 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.

## 4. Opaque schedulability analysis of a component

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 deadline constraints of tasks. Such an abstraction requires an explicit choice for a resource-supply model, capturing the processor supply to a component. Moreover, a component specifies what it needs in terms of resources and exposes those resources that may be shared globally in its interface. Whether or not a global resource is actually used by other components is unknown within the context of a component.

There are several ways to account for local scheduling penalties due to global resource sharing. One might assume that each resource must be globally shared and, subsequently, account for the worst-case overhead inside the local analysis. Alternatively, we opt for the assumption that all resources are just locally shared during the local analysis and we compensate for global sharing between components at integration time. These assumptions are often made tacitly.

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 treats all resources accessed by the component as local, so that, even under global sharing, properties of the global synchronization protocol do not influence the computed interface parameters.*

The key consequence of an opaque local analysis is the absence of assumptions on the global synchronization protocol. Section 4.1 shows how resource holding times, $X_{s\ell} \in \mathsf{X}_s$, can be computed without making assumptions on the global synchronization protocol. Next, we accomplish the same for budget parameter $Q_s$ in the interface of the component. This means that the values of the resource holding times should be absent in the equations that validate the local tasks' schedulability. **Table 1** gives an overview of local analyses found in literature by indicating their opacity. This section proceeds with an opaque analysis which ultimately results in a uniform representation of component interfaces, $\Gamma(P_s, Q_s, \mathrm{X}_s)$.

| **Analysis of resource-sharing strategies** | **Authors** | **Opacity** | **Impact on local analysis** |
|---|---|---|---|
| BROE | Bertogna et al. [24] | Yes | – |
| Enhanced BROE | Biondi et al. [29] | No | Proc.-supply model uses RHTs |
| HSRP—overrun without payback (ONP) | Davis and Burns [22] | No | Not compositional |
| HSRP—overrun without payback (ONP) | Behnam et al. [30] | Yes | – |
| Enhanced overrun | Behnam et al. [30] | No | Proc.-supply model uses RHTs |
| Improved overrun without payback | Behnam et al. [35] | No | Proc.-supply model uses RHTs |
| HSRP—overrun with payback (OWP) | Davis and Burns [22] | No | Not compositional |
| HSRP—overrun with payback (OWP) | Behnam et al. [30] | No | Proc.-supply model uses RHTs |
| SIRAP | Behnam et al. [23,28] | No | Proc.-supply model uses RHTs |

**Table 1.** Overview of the synchronization protocol's support for integrating resource-sharing components into the HSF with opaque analysis.

### 4.1. Computing resource holding times

The resource holding times were introduced by Bertogna et al. [36] in order to represent the cumulative processor time consumed by the tasks within the same component $C_s$ that can preempt a task $\tau_i$ while it is holding a resource $R_\ell$. The way of computing resource holding times of tasks therefore depends on the local scheduling policy, because the scheduling policy determines possible preemptions. Besides the scheduling policy, preemptions may be limited by a resource ceiling, i.e., the value of the local resource ceiling $rc_{s\ell}$ also influences the *resource holding times* (see **Figure 3**). In an HSF, the resource holding time represents the longest critical-section length as experienced by blocked tasks in other components.

In literature, various system assumptions in the description of a particular global synchronization protocol have shown to affect the way of computing resource holding times [e.g., see 23, 24, 30]. However, all these methods can be simplified and unified (independent of the local scheduling policy and the global synchronization protocol) by assuming that the component's period $P_s$ is smaller than the tasks' periods.

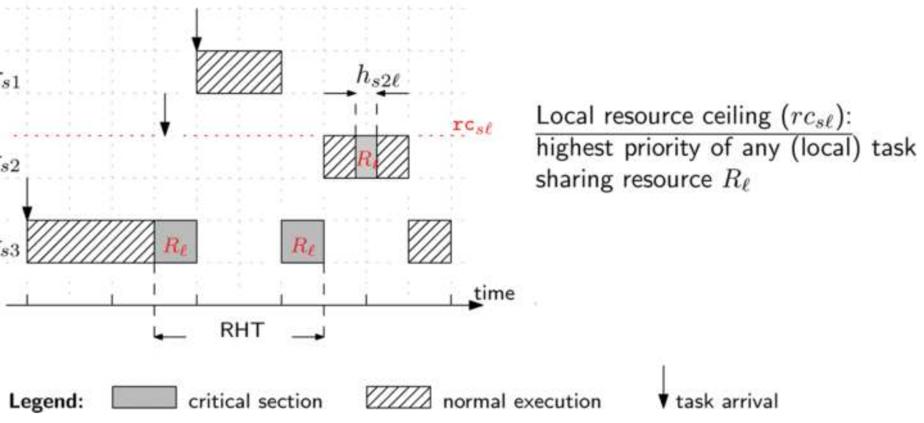

**Figure 3.** The resource holding time (RHT) represents the cumulatively consumed processor time by any task of a component while one task holds a resource. In order to guarantee mutual-exclusive access to resource $R_\ell$, the associated resource ceiling ($rc_{s\ell}$) is at least equal to the highest preemption level of any (local) task sharing resource $R_\ell$. One may consider to further limit preemptions during critical sections by increasing the resource ceiling. On the one hand, this may lead to longer blocking delays to tasks with a higher preemption level (in this case: $\tau_{s1}$). On the other hand, this decreases the tasks' RHTs (in this case: $\tau_{s2}$ or $\tau_{s3}$).

The main observation leading to this simplification is that an access to a global resource must be followed by a release of the resource in the same component period, for example, established by the self-blocking mechanisms or the overrun mechanisms considered in real-time literature. If a resource must be accessed and released in the same component period which is smaller than the task periods, then we can limit the number of preemptions within a critical section and this, on its turn, will lead to a smaller resource holding time. The resource holding time of a task $\tau_i$ accessing a resource $R_\ell$ is captured by a value $X_{si\ell}$, which represents the amount of processor time supplied to component $C_s$ from the access until the release of task $\tau_{si}$ to resource $R_\ell$. We now present a lemma that captures the possible preemptions of task $\tau_i$, regardless of other system assumptions.

**Lemma 1** (Taken from van den Heuvel et al. [34]). *Given* $P_s < T_s^{\min}$ *and* $T_s^{\min} = \min\{T_{si} \mid 1 \le i \le n_s\}$, *all tasks* $\tau_{sj}$ *that are allowed to preempt a critical section accessing a global shared resource* $R_\ell$, *i.e.,* $\pi_{sj} > rc_{s\ell}$, *can preempt at most once during an access to resource* $R_\ell$ *when using any global SRP-compliant protocol, independent if the local scheduler is EDF or FPPS.*

Lemma 1 makes it possible to compute the *resource holding time*, $X_{si\ell}$ of task $\tau_{si}$ to resource $R_\ell$ as follows:

$$X_{si\ell} = h_{si\ell} + \sum_{\pi_{sj} > rc_{s\ell}} E_{sj}, \tag{3}$$

and the maximum resource holding time within a component $C_s$ is computed as

$$X_{s\ell} = \max\{X_{si\ell} \mid 1 \le i \le n_s\}. \tag{4}$$

The computed values of $X_{s\ell}$ are included in the set $X_s$ which is part of the component's interface, $\Gamma_s$. We recall that opacity requires that the way of computing the interface parameters $Q_s$ and $\mathrm{X}_s$ of a component is independent of the global synchronization protocol; Lemma 1 establishes this requirement for the set of resource holding times, $\mathrm{X}_s$, of a component.

## 4.2. Computing a processor budget

The traditional schedulability analysis of tasks fills in task characteristics in a demand-bound function or a request bound function and compares the tasks' requirements with the supplied processor resources. The same schedulability analysis holds for tasks executing within a component, although the processor supply is modeled in a more complicated way.

The *processor supply* refers to the amount of processor resources that a component $C_s$ can provide to its tasks in order to satisfy deadline constraints. The linear lower bound of the processor resources supplied to a component with a periodically assigned processor (specified by an interface $\Gamma_s = (P_s, Q_s, \mathrm{X}_s)$) is given by [2]:

$$\mathtt{lsbf}_{\Gamma_s}(t) = \frac{Q_s}{P_s}\big(t - 2(P_s - Q_s)\big). \tag{5}$$

The longest interval a component may receive no processor supply on a periodic resource $\Gamma_s = (P_s, Q_s, \mathrm{X}_s)$ is named the *blackout duration*, i.e.,

$$BD_s = \max\left\{t \mid \mathtt{lsbf}_{\Gamma_s}(t) = 0\right\} = 2(P_s - Q_s). \tag{6}$$

The $\mathtt{lsbf}_{\Gamma_s}(t)$ in (5) is not only a linear approximation of the supplied processor resources in an interval of length $t$, it also models a bounded-delay resource supply as defined by [3] with a continuous, fractional provisioning of $\frac{Q_s}{P_s}$ of the shared processor (also referred to as the virtual processor speed) and a longest initial service delay of $BD_s$ time units.

### *4.2.1. Testing interfaces with earliest-deadline-first scheduling of tasks*

Assume we are given a component $C_s$ and its tasks have to execute on a periodic budget $Q_s$ every period $P_s$. If this processor budget is allocated to the tasks according to an EDF scheduling policy, then the following sufficient schedulability condition holds (as described by Shin and Lee [16]):

$$\forall t : t \in S_s : dbf_s(t) \leq lsbf_{\Gamma_s}(t), \tag{7}$$

where $\mathtt{dbf}_s(t)$ denotes the cumulative processor demand of all tasks of component $C_s$ for a time interval of length $t$ and the set $\mathsf{S}_s$ denotes a non-empty finite set of time-interval lengths (see Baruah [37]), i.e.,

$$\mathrm{S}_s \overset{\text{def}}{=} \left\{ t = b \cdot T_{si} + D_{si} \mid 1 \leq i \leq n_s; b \in \mathrm{N}^+; t \in \left(0, lcm\left\{T_{s1}, \ldots, T_{sn_s}\right\}\right]\right\}. \tag{8}$$

The $\mathtt{dbf}_s(t)$ is fully compliant to the schedulability analysis for task sets on a dedicated unit-speed processor, i.e.,

$$\mathtt{dbf}_s(t) = b_s(t) + \sum_{1 \leq i \leq n_s} \left\lfloor \frac{t - D_{si} + T_{si}}{T_{si}} \right\rfloor E_{si}. \tag{9}$$

The blocking term, $b_s(t)$, is defined according to the SRP, as described by Baruah [37]:

$$b_s(t) = \max\{h_{sj\ell} \mid \exists k : h_{sk\ell} > 0 \wedge D_{sk} \leq t < D_{sj}\}. \tag{10}$$

The algorithmic complexity of verifying the scheduling condition in (7) is pseudo-polynomial in the number of tasks.

### 4.2.2. *Testing interfaces with fixed-priority preemptive scheduling of tasks*

Assume we are given a component $C_s$ and its tasks have to execute on a periodic budget $Q_s$ every period $P_s$. If this processor budget is allocated to the tasks according to a FPPS policy, then the following sufficient schedulability condition holds (as described by Shin and Lee [16]):

$$\forall 1 \leq i \leq n_s : \exists t \in \mathrm{S}_{si} : \mathtt{rbf}_s(t,i) \leq \mathtt{lsbf}_{\Gamma_s}(t), \tag{11}$$

where $\mathtt{rbf}_s(t, i)$ denotes the worst-case cumulative processor request of $\tau_{si}$ for a time interval of length $t$ and the set $\mathsf{S}_{si}$ denotes a non-empty finite set of time-interval lengths (see Lehoczky et al. [38]), i.e.,

$$\mathrm{S}_{si} \overset{\text{def}}{=} \left\{ t = b \cdot T_{sa} \mid a < i; b \in \mathbf{N}^+; t \in (0, D_{si}]\right\} \cup \{D_{si}\}. \tag{12}$$

The $\texttt{rbf}_s(t,\, i)$ is fully compliant to the schedulability analysis for task sets on a dedicated unit-speed processor, i.e.,

$$\texttt{rbf}_s(t,i) = b_{si} + \sum_{1 \le j \le i} \left\lceil \frac{t}{T_{sj}} \right\rceil E_{sj}. \tag{13}$$

The blocking term, $b_{si}$, is defined according to the SRP, as described by Baker [21]:

$$b_{si} = \max\{h_{sj\ell} \mid \pi_{sj} < \pi_{si} \le rc_{s\ell}\}. \tag{14}$$

The algorithmic complexity of verifying the scheduling condition in (11) is pseudo-polynomial in the number of tasks.

*4.2.3. Deriving the processor budget from the scheduling test*

Computing a processor budget for a component $C_s$ involves a function that takes a fixed period $P_s$, a task set and a local scheduling policy as input and returns the smallest component budget $Q_s$. The function should satisfy, dependent on the local scheduling policy, the condition in (7) or (11).

One may approximate the size of the smallest required processor budget by means of a binary search in the range $Q_s \in (0, P_s)$. As amongst others suggested by Shin and Lee [2], the smallest value of budget $Q_s$ can be found by means of taking an intersection between the left-hand sides and the right-hand sides of the inequalities. This intersection concerns solving a simple quadratic equation (e.g., see Lipari and Bini [14]).

## 5. Integration of components and global schedulability

In this section, we present the composition rules of components in HSFs in the presence of global shared resources. A global integration test implements the admission control for components based on the resource requirements specified in their interfaces. The global analysis explicitly takes into account the corresponding penalties for global resource sharing which depend on the synchronization protocol applied at the top-level scheduler. These penalties include (*i*) blocking between components and (*ii*) protocol-specific penalties (in our case, either BROE, ONP, OWP, or SIRAP). Dependent on the chosen global synchronization protocol, the latter influences the processor requests by a component or it influences the processor supplies to a component. To analyze these scheduling penalties appropriately, it is reasonable to assume that during component-integration in the HSF, the synchronization protocol of the HSF is known.

Looking at resource sharing between components, the effectively used processor bandwidth of a component therefore depends on two parts (see Section 3.1): the processor budget (denoted by $Q_s$ in interface $\Gamma_s$) and the set of budgets on global resources (which are the resource holding times denoted by $X_s$ in interface $\Gamma_s$). The budget $Q_s$ should be sufficient to meet the deadline constraints of the tasks and no other constraints should influence the size of $Q_s$ (e.g., constraints related to global synchronization should be avoided). The resource holding times define the amount of execution time that a component receives for an accessing a global resource. In other words, if component $C_s$ is granted access to resource $R_\ell$, it receives $X_{s\ell}$ time units of execution time on resource $R_\ell$ prior to the implicit deadline $P_s$. The global synchronization protocol defines how this is established and the run-time rules of the protocol may or may not lead to an overlap of the processor allocation times to a component as contained in $Q_s$ and $X_s$. In the remainder of this section, we show the integration of components for two scheduling policies (EDF and FPPS) applied to the allocated bandwidth of the components.

### 5.1. Earliest-deadline-first scheduling of components

In the processor supply model, we assumed that the component's period also serves as a deadline for the provisioning of its processor budget. The following utilization-based schedulability condition, as defined by Baruah [37], can therefore be applied to the top-level EDF-scheduler:

$$\forall w : 1 \le w \le N : \frac{B(P_w)}{P_w} + \sum_{1 \le s \le w} \frac{Q_s + O_s(P_s)}{P_s} \le 1. \tag{15}$$

The blocking term, $B(t)$, presents the resource holding time of a potentially interfering, resource-sharing component with a deadline beyond the considered component, $C_w$; it is defined by Baruah [37]:

$$B(t) = \max\{X_{u\ell} \mid \exists s : R_\ell \in \mathrm{R}_u \cap \mathrm{R}_s \wedge P_s \le t < P_u\}. \tag{16}$$

The term $O_s(t)$ defines the additional amount of budget that a component $C_s$ requires under a certain global synchronization protocol in order to prevent excessive blocking durations for other components in the system.

With ONP, a component can request for an additional amount of $X_s$ time units of processor time each period $P_s$. Similarly, with SIRAP, a task of a component may idle away at most $X_s$ time units of processor time each period $P_s$. Hence, the synchronization penalties of both SIRAP and ONP can be modeled by allocating $X_s$ time units of processor time in addition to the regular processor budget $Q_s$ in each period $P_s$, so that the term $O_s(t)$ is defined by Behnam et al. [30] for ONP and van den Heuvel et al. [39] for SIRAP:

$$O_s(t) = \left\lfloor \frac{t}{P_s} \right\rfloor X_s. \tag{17}$$

With OWP, a component can only request an additional amount of $X_s$ time units of processor time once. Hence, the term $O_s(t)$ is defined by [30]:

$$O_s(t) = \begin{cases} X_s & \text{if } t \geq P_s \\ 0 & \text{otherwise.} \end{cases} \tag{18}$$

For both SIRAP and BROE, it is required that $X_s \leq Q_s$ in order to be able to complete an entire critical section within a single budget of size $Q_s$. For SIRAP, we establish this condition by allocating $Q_s + X_s$ time units of processor budget every period $P_s$. For BROE, however, we increase $Q_s$ with $O_s(t)$ time units if it is too small to fit $X_s$ time units contiguously, where $O_s(t)$ is defined as follows:

$$O_s(t) = \left\lfloor \frac{t}{P_s} \right\rfloor \max\left(0, X_s - Q_s\right). \tag{19}$$

### 5.2. Fixed-priority preemptive scheduling of components

For global FPPS of components—by definition disallowing BROE—the following sufficient scheduling condition can be applied (as defined by Lehoczky et al. [38]):

$$\forall 1 \leq s \leq N : \exists t \in (0, P_s] : B_s + \sum_{1 \leq r \leq s} \left( O_r(t) + \left\lceil \frac{t}{P_r} \right\rceil Q_r \right) \leq t. \tag{20}$$

The blocking term, $B_s$, is defined by the resource holding time of a lower-priority, resource-sharing component (in line with Baker [21]):

$$B_s = \max\{X_{u\ell} \mid \Pi_u < \Pi_s \leq RC_\ell\} \tag{21}$$

and the term $O_r(t)$ defines the additional amount of budget that a component $C_s$ requires under a certain global synchronization protocol in order to prevent excessive blocking durations for other components in the system.

Similar to EDF, under global FPPS the term $O_r(t)$ is defined by Behnam et al. [30] for ONP and by van den Heuvel et al. [39] for SIRAP:

$$O_r(t) = \left\lceil \frac{t}{P_r} \right\rceil X_r. \tag{22}$$

Also under FPPS, a component arbitrated by OWP can only request an additional amount of $X_s$ time units of processor time once. Hence, the term $O_r(t)$ becomes time independent and it is defined by [30]:

$$O_r(t) = X_r. \tag{23}$$

Just like with tasks, a finite set of time-interval lengths $t$ can be tested in order to determine the schedulability of a set of components, i.e., the set can be specified as in (12) using component period $P_s$ as the deadline for the execution of budget (WCET) $Q_s$. The algorithmic complexity of verifying the scheduling condition in (20) is then pseudo-polynomial in the number of components.

## 6. On the importance of opacity and its properties

Traditional protocols such as the PCP by Sha et al. [18] and the SRP by Baker [21] can be used for *local* resource sharing in HSFs, as observed by Almeida and Peidreiras [13]. With an opaque local analysis, we can re-use the same local analysis of components in the presence of global shared resources. The local analysis for HSFs with the ONP protocol, as presented by [30], already satisfied the notion of opacity because it uses a simple overrun upon integration and nothing else locally. In the previous sections, we also unified the local analysis of HSFs with other resource-sharing protocols (OWP, SIRAP, and BROE). This means that the interface of a component is independent of the resource-arbitration protocol. In this section, we briefly review non-opaque analysis and we highlight some important properties of opacity.

### 6.1. Monotonicity of the analysis

In Sections 4 and 5, we have summarized the compositional timing analysis of an HSF: the global analysis verifies the admission of a set of components into the HSF and the local analysis verifies the deadline constraints of tasks of each component in isolation on a periodically allocated budget, $Q_s$. The local scheduling conditions in (7) and (13) determine the smallest size of $Q_s$. For analyzing these conditions, we observe that increasing a local resource ceiling $rc_{s\ell}$ cannot lead to less blocking of local tasks (term $b_s(t)$ or $b_{si}$) and, thus, it cannot lead to a smaller budget $Q_s$. As a result, we have the following property.

**Property 1***In an analysis satisfying* (7) *for EDF or* (13) *for FPPS, the total requested resources of a component reflected in the allocated budget $Q_s$ is monotonically non-decreasing with an increase of a local resource ceilings $rc_{s\ell}$, $\forall\ R_\ell \in \mathcal{R}_s$.*

Although not mentioned explicitly, this property is tacitly assumed in some analysis (e.g., by Shin et al. [33], Behnam et al. [40] and Behnam et al. [30]) and our analysis supports it as well. It holds for all global synchronization protocols except for some non-opaque analysis (see **Table 1**).

### *6.1.1. SIRAP and its opacity*

The analysis as traditionally presented for SIRAP is non-opaque. However, SIRAP is an important and widely used protocol, so we side-step this problem by applying the same local and global analysis of ONP also to SIRAP, i.e., inserting $X_s$ units of idle time every period $P_s$. The intuition behind this idea is that SIRAP never idles away more processor time in one component period $P_s$ than ONP requires for overrun (see van den Heuvel et al. [39] for the details). This adjusted analysis of SIRAP satisfies Property 1.

### *6.1.2. How Property 1 could be violated*

Since global resources may need to be shared with tasks in other components, the ideas underlying most of the non-opaque analyses (like the non-opaque ones in **Table 1**) is to use the resource holding times of local tasks to tighten the analysis of wasted resources. Often this means that the tasks of a component are penalized by changes in the processor supply due to arbitrated accesses to global resources. The properties of a synchronization protocol are then reflected on the computed value of budget $Q_s$ of a component. For example, this could work based on the following observations:

- With OWP, see Behnam et al. [30], the resource holding time can be used to account for the processor time that is being exchanged between two consecutive component periods due to overruns and paybacks.
- With ONP, see Behnam et al. [35], the resource holding times can be used to tighten the delivery of budget $Q_s$ in component period $P_s$, because an overrun must fit in each component period as well.
- With SIRAP, see Behnam et al. [23], the resource holding times can be used to determine the amount of resources that can be idled away by the tasks in each component period.
- With BROE, see Biondi et al. [29], the resource holding times can be used to bound an additional delay due to resource sharing experienced by tasks compared to the regular delay of their resource supply ($BD_s$).

Intuitively, resource sharing comes with penalties to the tasks involved. However, sometimes the local tasks may also benefit from resource sharing, i.e., the tasks may require less processor resources. Section 6.1.3 presents an example of such a scenario using a non-opaque analysis of ONP. This clearly shows that a non-opaque analysis may violate our monotonicity property (Property 1).

### *6.1.3. Motivating example*

We now demonstrate the effect of using properties of the global synchronization protocol for optimizing the parameters of the component's interface in the local analysis. We consider a simple non-opaque analysis for ONP. Behnam et al. [35] improved ONP by observing that the normal budget $Q_s$ of a component $C_s$ has to be served at least before $P_s - X_s$ instead of the regular relative deadline $P_s$ (as we assumed for our analysis). The reason is that an overrun of at most length $X_s$ must also fit in each period $P_s$ after budget $Q_s$ has been depleted. This means that the blackout duration of the processor supply becomes shorter, so that tasks have to wait shorter until they get selected for execution by the local scheduler. Behnam et al. [35] model their idea with the help of the explicit deadline periodic resource model by Easwaran et al. [12]. The explicit deadline $P_s - X_s$ improves the required budget of the tasks in a non-opaque way because it uses resource holding times to tighten the deadline.

**Example 1***Consider a component $C_1$with a period $P_1 = 10$ and a single task $\tau_{11} = (27, 5, 27, \{0.5\})$ which specifies an access to a global resource $R_\ell$for a duration of $h_{11\ell} = X_{11\ell} = X_1 = 0.5$ time units. We use ONP for arbitrating access to global resources.*

*According to the improved ONP analysis of Behnam et al.* [35] *where the resource holding time of* 0.5 *time units is exploited to tighten the deadline for budget $Q_1$, it is sufficient to allocate $Q_1 = 2.5$ time units every period of 10 time units. This budget allocation can be captured by interface $\Gamma_1 = (P_1,\ Q_1,\ X_1) = (10,\ 2.5,\ \{0.5\})$ with explicit deadline $P_1 - X_1$. This interface is derived based on the assumption that an additional amount of 0.5 time units may need to be supplied within one component period to complete resource access by means of a budget overrun.*

*If resource $R_\ell$turns out to be local to component $C_1$, i.e., component $C_1$is independent of other components in the system, then budget overruns are unnecessary for accessing resource $R_\ell$. An independent component $C_1$would have required a periodic budget of$Q_1 = \frac{8}{3}$time units every period of 10 time units. We recall, however, the 2.5 time units must be supplied within 9.5 time units from the budget's release, leading to a density of processor allocations of$\frac{2.5}{9.5}$. This density is higher than the one without resource sharing, i.e.,$\frac{8/3}{10} < \frac{2.5}{9.5}$.*

*Once the HSF is composed, one may admit a component into the system requiring$\frac{8}{3}$time units every 10 time units while one may need to reject a component requiring 2.5 time units before relative deadline 9.5 every 10 time units. This depends on the resources requirements of other components in the HSF. Hence, a non-opaque analysis may give the illusion of resource efficiency by artificially creating resource dependencies.*

By making assumptions in the local analysis on how ONP changes the processor supply to a component, a manufacturer may give an untruthful impression on the efficiency of the component. Such impressions cannot be given through an opaque analysis due to its monotonicity property. For some protocol-specific local analysis, monotonicity of the local analysis is hard to prove or disprove. Nevertheless, the above example clearly shows the importance of monotonicity for multi-vendor environments, for example, obtained through an opaque local analysis.

### 6.2. Detecting and accounting for shared resources

An additional problem with a non-opaque analysis (e.g., see the analyses in **Table 1**) is that it impacts the budget of a component with synchronization penalties, no matter whether or not an accessed resource is shared with other components. As a result, the synchronization penalties are incorporated in the component's timing interface and they cannot be taken out any longer. Hence, it disallows us to account for the synchronization penalties corresponding to the resources that really need to be shared between components as detected at integration time.

Since a component is unaware of other components, it is also unknown which resources actually need to be shared. Instead of directly deriving the interface of a component, one may therefore perform an intermediate step. One may specify partial interfaces for components for each of the resources the component requires. Upon integration of components, these partial interfaces can then be combined into a true interface by selecting just the interfaces corresponding to the resources that are globally shared with other components.

This procedure works intuitively with an opaque analysis and works as follows. Given a component $C_s$, we assume that $P_s$ is given by the system designer and is fixed during the whole process of merging partial interfaces into a single interface. A partial interface $\Gamma_{s\ell}$ considers one global resource $R_\ell$ in isolation, i.e., $R_\ell$ can be globally shared or it can be local to the component.

**Definition 2 (Partial interface candidate)***(Taken from van den Heuvel et al.* [34]*) A partial interface candidate $\Gamma_{s\ell} = (P_s, Q_{s\ell}, \{X_{s\ell}\})$ of component $C_s$ accessing resource $R_\ell$ is a valid interface $\Gamma_s$ for component $C_s$—i.e., an interface that satisfies the local scheduling condition in (7) for EDF or (11) for FPPS—under the assumption that only resource $R_\ell$ may be globally shared with other components.*

A partial interface $\Gamma_{s\ell}$ is a valid interface for the restrictive case that resource $R_\ell$ is the only resource being exposed globally. It specifies the budget and the resource holding time to resource $R_\ell$ of the component, but other resources accessed by the same component are excluded from the interface. The remaining problem is to derive one interface for the case a component accesses more than one globally shared resource. Lemma 2 shows how to merge partial interfaces into a single interface.

**Lemma 2** (Taken from van den Heuvel et al. [34]). *Assume a component $C_s$ accesses $m_s$ resources. Let a selection of partial interfaces of component $C_s$ be a series of $\Gamma_{s\ell} = (P_s, Q_{s\ell}, \{X_{s\ell}\})$, i.e., one partial interface $\Gamma_{s\ell}$ is selected for each resource $R_\ell$. The local tasks' deadlines of component $C_s$ are met by interface $\Gamma_s = (P_s, Q_s, \{X_{s\ell} \mid R_\ell \in \mathcal{R}_s\})$, where $Q_s = \max\{Q_{s\ell} \mid R_\ell \in \mathcal{R}_s\}$.*

The intuition behind this lemma is as follows. By virtue of the SRP [21], a task $\tau_{si}$ can be blocked by just one (outermost) critical section of task $\tau_{sj}$ with a lower preemption level (where $\pi_{si} \geq \pi_{sj}$) before $\tau_{si}$ can start its execution. Hence, it is sufficient to add the largest difference in budget to the value of $Q_{s\ell}$ in order to accommodate for one local blocking occurrence.

Lemma 2 presents an important result for open environments in which components may be loaded or removed from the platform after deployment. It enables us to incrementally analyze the resource dependencies of the components in the HSF. Prior to the integration of components, it is still unclear which of the global resources need to be shared with other components and which resources can be treated as local. Upon integration of components, the sets $\mathcal{R}_s$ of accessed resources by component $C_s$ with the resources that truly need to be shared globally are known. We can then make an appropriate selection of partial interfaces and combine them into a single interface for each component. **Figure 4** illustrates this procedure as defined by Lemma 2. The result is that we account for the synchronization penalties of just the globally shared resources. If components enter or leave the HSF, one may use the partial interfaces to detect the updated resource dependencies between the components in the HSF.

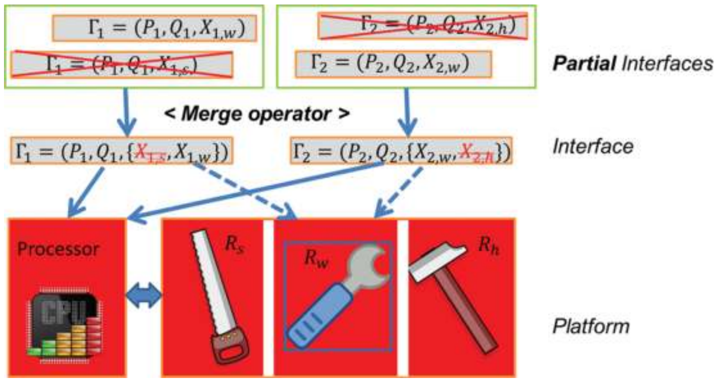
$\Gamma_1 = (P_1, Q_1, X_{1,w})$
~~$\Gamma_1 = (P_1, Q_1, X_{1,s})$~~
~~$\Gamma_2 = (P_2, Q_2, X_{2,h})$~~
$\Gamma_2 = (P_2, Q_2, X_{2,w})$
**Partial** Interfaces
< Merge operator >
$\Gamma_1 = (P_1, Q_1, \{\cancel{X_{1,s}}, X_{1,w}\})$
$\Gamma_2 = (P_2, Q_2, \{X_{2,w}, \cancel{X_{2,h}}\})$
Interface
Processor
CPU
$R_s$
$R_w$
$R_h$
Platform

**Figure 4.** Partial interfaces define the resource requirements of a component on each accessed resource separately. They can be combined into a single interface which captures all resource requirements of a component. The resources that do not need to be shared between components can be ignored (in this example, $R_s$ and $R_h$ can be ignored), so that resource arbitration and the corresponding penalties can be avoided for those resources.

## 7. Conclusion

This chapter introduced the notion of uniform interfaces for resource-sharing components that need to be integrated on a uni-processor platform. The interface of a component abstracts from global resource sharing until component integration. The local timing analysis of a component that returns such an interface is called *opaque*. Sufficient conditions for opacity are

- component periods are smaller than the local tasks' periods, so that resource holding times of a component are defined independently of the global synchronization protocol;
- resource holding times must disappear from the local schedulability test, so that the budget parameter of a component can be solely computed with the purpose of meeting deadline constraints of tasks (and independently of the global synchronization protocol).

As a result of both conditions, when the SRP arbitrates access to shared resources between periodic components, the necessary condition of opacity is satisfied: all interface parameters of a component are computed independently of a global synchronization protocol. Moreover, component dependencies on shared resources and the corresponding synchronization penalties can be optimized at component integration.

We applied opacity to four existing global synchronization protocols: SIRAP, ONP, OWP, and BROE. For some systems that deploy such a protocol, a non-opaque analysis has shown to significantly improve schedulability (e.g., as demonstrated by Behnam et al. [28], Biondi et al. [29]). However, this requires that components are delivered to system integrators with an interface that includes worst-case synchronization penalties, which in practice may never occur. We therefore believe that the simplicity of opaque analysis and its opportunities to analyze systems incrementally may be beneficial for complex systems in which component development, test, analysis, and integration is spread over different research and development teams.

## 8. Glossary

This section gives an overview of the abbreviations and the symbols being used throughout this chapter.

| **Abbreviation** | **Description** |
|---|---|
| AUTOSAR | AUTomotive Open System Architecture |
| BROE | Bounded-delay resource open environment |
| BD | Blackout duration |
| BWI | Bandwidth inheritance |
| DM | Deadline monotonic |
| EDF | Earliest-deadline-first |
| EDP | Explicit-deadline periodic |
| FPPS | Fixed-priority preemptive scheduling |
| HSFs | Hierarchical scheduling frameworks |
| HSRP | Hierarchical SRP |
| IPCP | Immediate PCP |
| LCM | Least common multiple |
| ONP | Overrun and no payback |
| OWP | Overrun with payback |
| PCP | Priority ceiling protocol |
| RHT | Resource holding time |
| LSBF | Linear supply-bound function |
| SIRAP | Subsystem integration and resource allocation policy |

| Abbreviation | Description |
|---|---|
| SRP | Stack resource policy |
| WCET | Worst-case execution time |

| Symbol | Description |
|---|---|
| $N$ | Number of components in the system |
| $M$ | Number of global resources |
| $\mathcal{R}$ | Set of global resources |
| $R_\ell$ | $\ell$-th global resource |
| $RC_\ell$ | Global resource ceiling of $R_\ell$ |
| $C_s$ | $s$-th component |
| $\Pi_s$ | Preemption level of $C_s$ |
| $P_s$ | Period of the resource allocations to component $C_s$ |
| $D_s$ | Relative deadline for the resource allocations to component $C_s$ |
| $Q_s$ | Periodically allocated processor time for $C_s$ |
| $\mathcal{R}_s$ | Set of global resources accessed by $C_s$ |
| $\mathtt{X}_s$ | Set of holding times to global resources accessed by $C_s$ |
| $X_{s\ell}$ | the resource holding time of $C_s$ for $R_\ell$ |
| $X_s$ | Maximum of the resource holding times of $C_s$ |
| $O_s$ | Processor time for $C_s$ merely dedicated to prevent excessive blocking |
| $\Gamma_s$ | Interface of $C_s$ defining periodic resource demands of $C_s$ |
| $\Gamma_{s\ell}$ | Partial interface defining $C_s$' demands for a given resource $R_\ell$ |
| $BD_s$ | Longest duration for $C_s$ without any processor supply |
| $\mathtt{dbf}_s(t)$ | Demand-bound function of the tasks of $C_s$ in an interval $t$ |
| $\mathtt{rbf}_s(t, i)$ | Request-bound function of task $\tau_{si}$ and its higher priorities in an interval $t$ |
| $\mathtt{lsbf}(t)$ | Linear lower bound of the processor supply in any sliding window of length $t$ |
| $\mathtt{T}_s$ | Task set of a component |
| $n_s$ | Number of tasks composing component $C_s$ |
| $\tau_{si}$ | $i$-th task of component $C_s$ |
| $T_{si}$ | Minimal inter-arrival time of task $\tau_{si}$ |
| $E_{si}$ | WCET of $\tau_{si}$ |
| $D_{si}$ | (Relative) deadline of $\tau_{si}$ |
| $S_{si}$ | Set of time instances that to determine schedulability of a task $\tau_{si}$ |
| $\mathcal{H}_{si}$ | Set of WCETs of task $\tau_{si}$ on resources |

| Symbol | Description |
|---|---|
| $rc_{s\ell}$ | Local resource ceiling of resource $R_\ell$ |
| $\pi_{si}$ | Preemption level of task $\tau_{si}$ |
| $h_{si\ell}$ | WCET of $\tau_{si}$'s largest critical section to $R_\ell$ |
| $X_{si\ell}$ | Largest resource holding time of $\tau_{si}$ to $R_\ell$ |

## Author details

Martijn M. H. P. van den Heuvel[1], Reinder J. Bril[1*], Johan J. Lukkien[1], Moris Behnam [2] and Thomas Nolte[2]

*Address all correspondence to: r.j.bril@tue.nl

1 Eindhoven University of Technology, Eindhoven, Netherlands

2 The Netherlands Mälardalen University, Västerås, Sweden

## References

[1] HolenderskiM., BrilR. J., and LukkienJ. J.. An efficient hierarchical scheduling framework for the automotive domain. In Morteza BabamirSeyed, editor, *Real-Time Systems, Architecture, Scheduling, and Application*, pages 67–94. InTech, 2012.

[2] ShinI. and LeeI.. Compositional real-time scheduling framework with periodic model. *ACM Trans. on Embedded Computing Systems (TECS)*, 70 (3):0 1–39, April 2008.

[3] FengX.A. and MokA.K.. A model of hierarchical real-time virtual resources. In *Real-Time Systems Symposium (RTSS)*, pages 26–35, December 2002.

[4] van den HeuvelM. M. H. P., BrilR. J., and LukkienJ. J.. Transparent synchronization protocols for compositional real-time systems. *IEEE Transactions on Industrial Informatics (TII)*, 80 (2):0 322–336, May 2012.

[5] López MartinezP., BarrosL., and DrakeJ.. Scheduling configuration of real-time component-based applications. In *Reliable Software Technology—Ada-Europe*, volume 6106 of *Lecture Notes in Computer Science (LNCS)*, pages 181–195. Springer, 2010.

[6] MercerC.W., SavageS., and TokudaH.. Processor capability reserves: operating system support for multimedia applications. In *International Conf. on Multimedia Computing and Systems (ICMCS)*, pages 90–99, May 1994.

[7] RajkumarR., JuvvaK., MolanoA., and OikawaS.. Resource kernels: a resource-centric approach to real-time and multimedia systems. In *SPIE/ACM Conference on Multimedia Computing and Networking (CMCN)*, pages 150–164, January 1998.

[8] DengZ. and LiuJ.W.-S.. Scheduling real-time applications in open environment. In *Real-Time Systems Symposium (RTSS)*, pages 308–319, December 1997.

[9] KuoT.-W. and LiC.-H.. A fixed-priority-driven open environment for real-time applications. In *Real-Time Systems Symposium (RTSS)*, pages 256–267, December 1999.

[10] LipariG. and BaruahS.K.. Efficient scheduling of real-time multi-task applications in dynamic systems. In *Real-Time Technology and Applications Symposium (RTAS)*, pages 166–175, May 2000.

[11] WandelerE. and ThieleL.. Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling. In *Conference on Embedded Software (EMSOFT)*, pages 80–89, September 2005.

[12] EaswaranA., AnandM., and LeeI.. Compositional analysis framework using EDP resource models. In *Real-Time Systems Symposium (RTSS)*, pages 129–138, December 2007.

[13] AlmeidaL. and PeidreirasP.. Scheduling with temporal partitions: response-time analysis and server design. In *Conference on Embedded Software (EMSOFT)*, pages 95–103, September 2004.

[14] LipariG. and BiniE.. A methodology for designing hierarchical scheduling systems. *Journal of Embedded Computing (JEC)*, 10(2):257–269, April 2005.

[15] FisherN. and DewanF.. A bandwidth allocation scheme for compositional real-time systems with periodic resources. *Real-Time Systems*, 480(3):223–263, 2012.

[16] ShinI. and LeeI.. Compositional real-time scheduling framework. In *Real-Time Systems Symposium (RTSS)*, pages 57–67, December 2004.

[17] de NizD., AbeniL., SaewongS., and RajkumarR.. Resource sharing in reservation-based systems. In *Real-Time Systems Symposium (RTSS)*, pages 171–180, December 2001.

[18] ShaL., RajkumarR., and J.P. Lehoczky. Priority inheritance protocols: an approach to real-time synchronisation. *IEEE Transactions on Computers (TC)*, 390(9):1175–1185, September 1990.

[19] SteinbergU., WolterJ., and HärtigH.. Fast component interaction for real-time systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 89–97, July 2005.

[20] LipariG., LamastraG., and AbeniL.. Task synchronization in reservation-based real-time systems. *IEEE Transactions on Computers (TC)*, 530(12):1591–1601, December 2004.

[21] BakerT.P.. Stack-based scheduling of realtime processes. *Real-Time Systems*, 30(1):67–99, March 1991.

[22] DavisR.I. and BurnsA.. Resource sharing in hierarchical fixed priority pre-emptive systems. In *Real-Time Systems Symposium (RTSS)*, pages 257–267, December 2006.

[23] BehnamM., ShinI., NolteT., and NolinM.. SIRAP: a synchronization protocol for hierarchical resource sharing in real-time open systems. In *Conference on Embedded Software (EMSOFT)*, pages 279–288, October 2007.

[24] BertognaM., FisherN., and BaruahS.. Resource-sharing servers for open environments. *IEEE Transactions on Industrial Informatics (TII)*, 50(3):202–219, August 2009.

[25] GhazalieT. M. and BakerT. P.. Aperiodic servers in a deadline scheduling environment. *Real-time Systems*, 90(1):31–67, July 1995.

[26] CaccamoM. and ShaL.. Aperiodic servers with resource constraints. In *Real-Time Systems Symposium (RTSS)*, pages 161–170, December 2001.

[27] HolmanP. and AndersonJ.H.. Locking in pfair-scheduled multiprocessor systems. In *Real-Time Systems Symposium (RTSS)*, pages 149–158, December 2002.

[28] BehnamM., NolteT., and BrilR. J.. Bounding the number of self-blocking occurrences of SIRAP. In *Real-Time Systems Symposium (RTSS)*, pages 61–72, December 2010 a .

[29] BiondiA., MelaniA., BertognaM., and ButtazzoG.. Optimal design for reservation servers under shared resources. In *Euromicro Conf. on Real-Time Systems (ECRTS)*, pages 153–164, July 2014.

[30] BehnamM., NolteT., SjodinM., and ShinI.. Overrun methods and resource holding times for hierarchical scheduling of semi-independent real-time systems. *IEEE Transactions on Industrial Informatics (TII)*, 60 (1):93–104, February 2010 b.

[31] LiuC.L. and LaylandJ.W.. Scheduling algorithms for multiprogramming in a real-time environment. *Journal of the ACM*, 200(1):46–61, January 1973.

[32] MokA.K.-L.. *Fundamental design problems of distributed systems for the hard-real-time environment*. Phd thesis, Massachusetts Institute of Technology, May 1983. http://www.lcs.mit.edu/publications/pubs/pdf/MIT-LCS-TR-297.pdf.

[33] ShinI., BehnamM., NolteT., and NolinM.. Synthesis of optimal interfaces for hierarchical scheduling with resources. In *Real-Time Systems Symposium (RTSS)*, pages 209–220, December 2008.

[34] M. M. H. P. van den Heuvel, BehnamM., BrilR. J., LukkienJ. J., and NolteT.. Optimal and fast composition of resource-sharing components in hierarchical real-time systems. In *IEEE Int. Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–12, Aug. 2014.

[35] BehnamM., NolteT., and BrilR. J.. Tighter schedulability analysis of synchronization protocols based on overrun without payback for hierarchical scheduling frameworks. In *International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 35–44, April 2011.

[36] BertognaM., FisherN., and BaruahS.. Static-priority scheduling and resource hold times. In *Parallel and Distributed Processing Symposium (IPDPS)*, March 2007.

[37] BaruahS. K.. Resource sharing in EDF-scheduled systems: a closer look. In *Real-Time Systems Symposium (RTSS)*, pages 379–387, December 2006.

[38] LehoczkyJ. P., ShaL., and DingY.. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Real-Time Systems Symposium (RTSS)*, pages 166–171, December 1989.

[39] M. M. H. P. van den Heuvel, BehnamM., BrilR. J., LukkienJ. J., and NolteT.. Opaque analysis for resource sharing in compositional real-time systems. In *4th Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)*, pages 3–10, Nov. 2011.

[40] BehnamM., NolteT., and FisherN.. On optimal real-time subsystem-interface generation in the presence of shared resources. In *Conference on Emerging Technologies and Factory Automation (ETFA)*, September 2010 c.