ABSTRACT
Hierarchical scheduling is an effective approach developed to support the integration of independently developed applications on the same computing platform. In particular, the M-BROE framework has been recently proposed and analyzed to efficiently support component-based development on multiprocessor platforms through the virtual multiprocessor abstraction implemented by reservation servers, in the presence of shared resources. However, the problems of partitioning applications to virtual processors and defining reservation parameters were not addressed.

This paper fills this gap by proposing a design methodology as an optimization problem for partitioning applications to virtual processors, performing a synthesis of the component interface and allocating virtual processors to physical processors. Experimental results are also presented to evaluate the proposed methodology.

Categories and Subject Descriptors
C.3 [SPECIAL-PURPOSE AND APPLICATION BASED SYSTEMS]: Real-time and embedded systems; D.4.7 [Organization and Design]: Real-time systems and embedded systems

1. INTRODUCTION
In several applications domains, the exponential growth of software functions imposed by the market is pushing software developers towards a hierarchical design approach, where multiple independently developed applications are executed on the same hardware platform. For example, in automotive systems such a hierarchical approach has already begun with the objective of containing the total number of electronic control units (ECUs) installed in a car, which implies a significant reduction of used space, weight, energy, and cost [24].

On the other hand, the concurrent execution of multiple applications on the same hardware generates new problems that must be solved in order to make the hierarchical approach effectively usable by the industry. One of the most serious problems occurring when multiple components share the same resources is caused by the reciprocal interference among them, which may introduce unbounded delays and cause unpredictable performance degradation [25].

An efficient kernel method for reducing the interference among concurrent applications is the resource reservation mechanism [1, 23], according to which each application is executed within a dedicated processor partition, implemented by a reservation server. A reservation server $S_k$ is a time provisioning mechanism that allocates a budget $Q_k$ for the application every period $P_k$. In case, the bandwidth reserved to an application results to be $a_k = Q_k/P_k$. The reservation mechanism must guarantee that the served application receives a given fraction of the processor bandwidth, but at the same time it cannot consume more than the allocated amount, thus protecting the other applications from possible overruns (temporal isolation).

When application tasks make use of mutually exclusive resources shared between reservations (such as I/O devices or global memory buffers) the isolation property could be broken when the server budget expires within a critical section. In this case, in fact, an extra delay would be added to the tasks blocked on the same resource to wait not only for the release of the lock, but also for the next budget replenishment.

To solve this problem, various approaches have been proposed in the literature [8, 9, 11, 18]. Among them, thanks to an improved schedulability analysis [15], the BROE protocol [11] proposed by Bertogna, Fisher and Baruah has been found to perform best.

The BROE protocol, originally developed for uniprocessor systems, has been recently extended by Biondi et al. [14] into M-BROE to support the development of component-based hierarchical systems on multiprocessor platforms in the presence of shared resources. In the M-BROE framework (reviewed in Section 2), the tasks of a software component are statically allocated to virtual processors (implemented via reservation servers), which are in turn allocated to the physical processors at component-integration time. The resulting infrastructure relies on partitioned hierarchical scheduling and non-preemptive FIFO spin locks to regulate the access to shared resources.

Although the authors fully characterized the M-BROE protocol and provided the schedulability analysis of components on given reservation servers, the problems of partitioning...
applications on virtual processors and defining reservation parameters were not addressed.

Other authors solved task partitioning in multicore systems using Integer Linear Program (ILP) formulations [5, 7], but without considering reservations, nor resource sharing. Bini et al. [12] addressed the problem of partitioning parallel applications upon reservation servers for platform virtualization and Khalilzad et al. [21] studied the problem allocating component interfaces in multiprocessor systems under partitioned EDF scheduling, but in both the works no resource sharing has been considered. Wieder and Brandenburg [27] presented an optimal ILP-based partitioning strategy for fixed priority scheduling with shared resources and Al-Bayati et al. [2] addressed the same problem using heuristic approaches.

However, none of these works addressed partitioning methodologies under reservation servers taking resource sharing into account. Resource sharing further complicates task partitioning problems (an example will be shown in Section 4), determining the need for ad-hoc approaches that explicitly take into account blocking times.

This paper fills this gap by proposing the following contributions.

Contributions.

- A methodology based on a Mixed-Integer Linear Program (MILP) formulation is proposed for partitioning applications on virtual processors taking shared resources into account. Once an allocation is found, the synthesis of component interfaces is performed to find the optimal reservation parameters that guarantee the schedulability of the task set.
- An MILP formulation is presented for allocating virtual processors to physical processors depending on the component interfaces.

Paper structure. The rest of the paper is organized as follows: Section 2 presents the system model and the M-BROE framework; Section 3 summarizes the schedulability results derived for the M-BROE framework; Section 4 describes the methodology to address task partitioning and virtual processors design; Section 5 reports an MILP formulation for integrating the virtual processors of all components upon the physical platform; Section 6 presents some experimental results for evaluating the proposed methodology; Section 7 concludes the paper.

2. FRAMEWORK AND MODELING

2.1 System model

We consider a system composed of $N$ software components $\Gamma_1, \Gamma_2, \ldots, \Gamma_N$, also referred to as applications. Each software component $\Gamma_k$ consists of a set $T(\Gamma_k)$ of $n_k$ real-time periodic or sporadic tasks. Each task $\tau_i \in T(\Gamma_k)$ is characterized by a worst-case execution time (WCET) $C_i$, a period (or minimum interarrival time) $T_i$, and a relative deadline $D_i$.

The system runs on a multiprocessor platform consisting of $M$ identical processors, each denoted as $\mathcal{P}_m$, with $m = 1, \ldots, M$. Each component $\Gamma_k$ is statically partitioned over $M$ virtual processors $S^k_j, j = 1, \ldots, M$, each implemented by a reservation server characterized by a budget $Q^k_j$ and a period $P^k_j$. The ratio $\alpha^k_j = Q^k_j / P^k_j$ is referred to as the reservation bandwidth. The virtual processor at which a task $\tau_i \in T(\Gamma_k)$ is assigned is denoted as $S(\tau_i)$.

Each component $\Gamma_k$ is abstracted through a component interface consisting of $M$ pairs $(Q^k_j, P^k_j)$, with $j = 1, \ldots, M$, representing budget and period of the reservation server associated with $S^k_j$. Note that, to keep the interface simple, no information related to tasks and resources is exported.

A component integrator is responsible for admitting or rejecting applications and statically assigning each reservation server to a specific physical processor. The processor on which server $S^k_j$ is assigned is denoted as $\mathcal{P}(S^k_j)$. For the sake of simplicity, this paper assumes that $M = M$. The framework described above is illustrated in Figure 1 for a platform of 4 processors.

![Figure 1: Proposed hierarchical framework.](image)

Tasks can share resources through mutually exclusive critical sections. At a component level, we distinguish between component resources, accessed only by tasks belonging to the same component and system resources, accessed by tasks belonging to different components.

For each resource $R_{\ell}$, $\delta_{i,\ell}$ denotes the length of the longest critical section of task $\tau_i$ related to $R_{\ell}$, while $\eta_{i,\ell}$ denotes the number of critical sections used by $\tau_i$ on $R_{\ell}$. We assume to have $\eta_{i,\ell} = 0$ if a task $\tau_i$ does not access resource $R_{\ell}$.

For a resource $R_{\ell}$ accessed by a task $\tau_i$, the Resource Holding Time $H_{i,\ell}$ is defined as the maximum budget consumed from the lock of $R_{\ell}$ until its unlock in the $\tau_i$’s code. Note that if a resource is accessed non preemptively, its resource holding time is equal to the critical section length (i.e., $H_{i,\ell} = \delta_{i,\ell}$), otherwise it must include all possible pre-emption delays occurring within the critical section [10]. A component $\Gamma_k$ can only be admitted in the system if all its tasks have a resource holding time bounded by $\mathcal{H}$, that is, if

$$\forall \tau_i \in T(\Gamma_k), \forall R_{\ell}, \ H_{i,\ell} \leq \mathcal{H}, \tag{1}$$

For component resources $R_{\ell}$ accessed by tasks allocated to
different virtual processors, the sum of the maximum resource holding times of $R_q$ from each virtual processor must be bounded by $M \mathcal{H}$. Formally, we require that

$$\forall k, \forall q \in \mathcal{R}_k, \sum_{j=1}^{M} \max \{ H_{i,q} | S(\tau_i) = S_k^j \} \leq M \mathcal{H},$$

where $\mathcal{R}_k$ denotes the set of component resources accessed by tasks executing on different virtual processors, formally defined as

$$\mathcal{R}_k = \{ R_q | \exists \tau_1, \tau_2 \text{ using } R_q \land S(\tau_1) \neq S(\tau_2) \}.$$ (3)

### 2.2 Scheduling infrastructure

Tasks allocated to a virtual processor are handled by a local scheduler, which can be any fixed-priority (FP) algorithm or Earliest Deadline First (EDF) [22]. Tasks may include non-preemptive regions in which preemption is disabled for the local scheduler. Each virtual processor is implemented by an M-BROE server [14] and the various M-BROE servers are scheduled under partitioned EDF scheduling on the $M$ processors.

Once reservation servers are mapped to physical processors, three types of shared resources can be distinguished: Local resources, shared only by tasks handled by the same server; Processor-local resources, shared only by tasks executing on the same processor, but on different servers; Global resources, shared by tasks executing on different processors. Local resources are accessed through the SRP [3] protocol, while processor-local resources are accessed by the H-SRP [18] protocol in conjunction with M-BROE, in a local non-preemptive manner. Finally, global resources are accessed by the MSRP [20] protocol in conjunction with M-BROE.

## 3. SUMMARY OF SCHEDULABILITY RESULTS

To make this work self consistent, this section summarizes the schedulability results derived for the M-BROE framework. In particular, after recalling how to derive blocking factors, we summarize the schedulability tests used for local (tasks guarantee upon virtual processors) and global (virtual processors guarantee upon the physical platform) analysis. Please refer to [14] for further details.

### 3.1 Resource sharing

Under the M-BROE framework, global resources are protected by non-preemptive FIFO spinlocks. If a task $\tau_i$ wants to use a global resource locked from a task on another processor, $\tau_i$ starts spinning non-preemptively until the resource is granted. Critical sections on global resources are also executed non-preemptively and simultaneous requests from different processors are served in FIFO order.

To analyze blocking times related to spinlocks we rely on the MSRP [20] analysis; please note that although an improved analysis for spinlocks has been proposed in [26], it cannot be directly used for the M-BROE framework for several reasons as explained in [14] (page 4). According to the MSRP analysis, a bound on the maximum spinning time (denoted as remote blocking) is computed for each global resource $R_\ell$ accessed from a given processor and used to inflate the WCET of the tasks using $R_\ell$. Also, non-preemptive spinning and non-preemptive access to global resources introduce a non-preemptive blocking factor that must be accounted for each task. The access to local and processor local resources is regulated by the SRP [3] and the H-SRP [16] protocols, generating local blocking and additional non-preemptive blocking, respectively.

In the following, we first provide an upper-bound for the spinning time and then report the expressions for computing remote blocking, non-preemptive blocking and local blocking.

**Upper-bound for the spinning time.** According to Lemma 1 in [14], a safe upper bound on the spinning time $\xi_{\ell,j}$ related to system resources $R_\ell$ can be computed as $\xi_{\ell,j} \leq (M - 1) \mathcal{H}$.

For component resources, critical section lengths are known, since they belong to tasks of the same component. However, when tasks are assigned to different virtual processors, it is not possible to infer the physical processor on which they will be executed. For this reason, a safe bound on the spinning time can be computed by assuming that all virtual processors of a component will be assigned to different physical processors. In this case, an upper bound on the spinning time $\xi_{\ell,j}$ for a component resource $R_\ell$ in $\Gamma_k$ can be computed as

$$\xi_{\ell,j} \leq \sum_{s_{\ell}^i \neq s_{\ell}^m} \max \{ \delta_{i,\ell} | S(\tau_i) = s_{\ell}^m \}.$$ (4)

This fact imposes also that processor-local resources have to be always accounted as global resources to capture the worst-case in the local analysis.

**Remote blocking.** An upper-bound $\xi_i$ on the remote blocking for task $\tau_i$ is computed by accounting for the maximum spinning time on each critical section of $\tau_i$, with $P(S(\tau_i)) = \mathcal{P}_j$, that is:

$$\xi_i = \sum_{R_\ell} \{ \eta_{i,\ell} \cdot \xi_{\ell,j} | R_\ell \text{ used by } \tau_i \}.$$ (5)

**Non-preemptive blocking.** Such a blocking is bounded by the longest non-preemptive section causing arrival blocking. Under local EDF scheduling, assuming that tasks $\tau_k$ and $\tau_j$ execute on processor $\mathcal{P}_j$, it can be computed as

$$B_{\ell}^{NP} = \max_{k, R_\ell} \{ \xi_{\ell,j} + \delta_{k,\ell} | R_\ell \text{ used by } \tau_k \land D_k > D_\ell \}.$$ (6)

**Local blocking.** The SRP blocking factor for task $\tau_i$ due to local resources is denoted by $B_{\ell}^{L}$ and is given by critical sections of resources that are locked by tasks $\tau_L$ with deadlines greater than $D_i$ and shared with tasks $\tau_H$ with deadlines less than or equal to $D_i$. Formally,

$$B_{\ell}^{L} = \max \{ \delta_{L,\ell} | D_H \leq D_i < D_L \land \tau_L, \tau_H \text{ use } R_\ell \}.$$ (7)

Since non-preemptive blocking and local blocking occur at the task’s release, the resulting blocking is called arrival blocking and can be computed as

$$B_{\ell} = \max \{ B_{\ell}^{NP}, B_{\ell}^{L} \}.$$ (8)

### 3.2 Local analysis

Using the processor demand criterion extended to include resource sharing [4], a task set $T$ is schedulable by EDF under the M-BROE server if:
\[ \forall t > 0 \quad B(t) + \text{dbf}(t) \leq \text{sbf}(t) \]  
where
\[ \text{dbf}(t) = \sum_{i \in T} \left( \left\lceil \frac{t - D_i}{T_i} \right\rceil + 1 \right) \left(C_i + \xi_i\right), \]
and \( \text{sbf}(t) \) is the supply bound function for the M-BROE server reported in [14] and \( (x)_0 \) denotes \( \max(0,x) \). Note that the \( \text{dbf}(t) \) in Equation (10) takes into account the computation time inflation \( \xi \) due to remote blocking.

3.3 Component integration analysis
The component integrator has to ensure the schedulability of the reservation servers assigned to each processor. Each component provides a set of reservation servers to the component integrator according to the specified interface; such reservation servers will be scheduled under partitioned scheduling on the \( M \) physical processors.

The schedulability test for guaranteeing the execution of virtual processors upon physical processors is reported in the following equation:
\[ \forall S_j^k : P(S_j^k) = P_m, \quad \sum_{r,l \leq P_j^k} \frac{Q_r^k}{P_r^k} + B_l^k \leq 1. \]  
Since the simple interface \((Q,P)\) is used for each component, no information is available about shared resources, hence, by Equations (2) and (3), the upper bound \( B_l^k = MH \) can be used. Note that a more accurate upper bound can be computed by adopting a more complex component interface, as the one proposed in [14].

4. PARTITIONING AND SERVER DESIGN
This section addresses the problem of partitioning application tasks on a set of virtual processors implemented by M-BROE reservation servers, and determining their configuration parameters in terms of budgets and periods.

A partitioning methodology for the M-BROE framework must take into account three aspects simultaneously: (i) the computational demand of the application; (ii) the parameters (budget and period) of the reservation servers; and (iii) the blocking times related to resource sharing (also including the effect of spinlocks). In particular, the dependency of partitioning on resource sharing is better illustrated by the following example.

**Example.** Consider a component composed of 3 tasks with periods \( T_1 = 10 \), \( T_2 = 20 \) and \( T_3 = 50 \) ms and implicit deadlines, to be executed on a platform composed of two processors. Tasks \( \tau_1 \) and \( \tau_2 \) share a component resource \( R \). If tasks \( \tau_1 \) and \( \tau_2 \) are assigned to different processors, then the access to \( R \) will be regulated by a spinlock. This solution penalizes the schedulability by generating non-preemptive blocking and increasing the execution times of \( \tau_1 \) and \( \tau_2 \) due to spinning time. Conversely, if the two tasks are allocated on the same processor, the blocking factor related to \( R \) is smaller (due to only uniprocessor SRP blocking). While it seems convenient allocating tasks on the same processor, it is easy to see that allocating \( \tau_2 \) together with the other tasks may not be appropriate, due to the local blocking experienced by \( \tau_2 \) (because \( D_1 < D_2 < D_3 \), from Equation (7)). If \( \tau_2 \) is allocated to the other processor, it will experience no blocking. Note however, that \( \tau_2 \) can experience a different blocking if it has a different period (e.g., \( T_2 = 5 \) ms).

The simple example presented above shows that resource sharing further complicates partitioning (which is intrinsically NP-HARD, being similar to a bin-packing problem), and must be taken into account to identify a “good” task allocation.

In this paper, task partitioning is formulated as an optimization problem. Unfortunately, however, given any performance objective (e.g., minimizing the overall bandwidth for the reservation servers), searching for an optimal solution is practically intractable for the following reasons:

- The EDF schedulability test requires the specification of a pseudo-polynomial number of check points [6] that depends on the parameters of the server upon which tasks execute (as explained in [11, 17]). Being the server parameters part of the output of the optimization problem (hence, unknown), upper bounds on them must be used, thus obtaining a potentially large set of constraints and variables.
- As shown in [17], it is possible to formulate an optimization problem to compute optimal BROE server parameters, minimizing the server bandwidth still ensuring the task schedulability. However, the exact problem formulation involves non-linear constraints, because the supply bound function of the server is non-linear.
- In the M-BROE framework, the access to global resources is protected by FIFO non-preemptive spinlocks, but an exact analysis for spinlocks is still missing, thus preventing the search for an optimal partitioning.

Given such limitations, this paper proposes a sub-optimal methodology for task partitioning and virtual processors design by splitting the problem in two phases: First, an MILP optimization is performed for partitioning tasks to virtual processors; then the optimal server parameters are computed through the approach presented in [17].

To overcome the problems highlighted above, the following approximations are proposed to express partitioning as an MILP optimization problem:

- The EDF schedulability is carried out by the Fully Polynomial Time Approximation Scheme (FPTAS), proposed by Fisher, Baker, and Baruah [19]. According to this approach, the workload of a task is described by the exact demand bound function for the first \( \lambda \) steps, and by a linear upper-bound for the remaining steps. Formal details about this approximation will be reported in Section 4.1.3.
- The reservation servers are approximated as ideal (fluid) virtual processors, running at a given speed \( \alpha \), which represents the server bandwidth. Note that using a classical bounded-delay (\( \alpha - \Delta \)) approximation, the optimization problem results non-linear.
- As done by Wieder and Brandenburg [27], blocking times in the presence of FIFO non-preemptive spinlocks are computed by the original MSRP (sufficient)
analysis proposed by Gai et al. [20].

Please, note that approaching the problem through an MILP formulation guarantees that the achieved partitioning is optimal with regard to the assumed approximations.

4.1 Optimization problem formulation

This section presents an MILP formulation to solve the problem of task partitioning upon virtual processors. The formulation is inspired by the one proposed by Wieder and Brandenburg [27] in the context of classical partitioned fixed-priority scheduling without reservation mechanisms.

As stated in Section 3, the worst-case blocking related to component resources is computed assuming that all the virtual processors of a component are assigned to different physical processors: without loss of generality, the index \( k = 1, \ldots, M \) will be used for both physical processors and virtual processors. In the following we refer to a single component, hence the component index is removed from all the terms used below. All the real variables used in the optimization problem are implicitly constrained as greater than or equal to zero. For such variables, lower-bounds expressing the minimum (safe) value will be used to ensure schedulability.

### 4.1.1 Decision variables for task allocation

The following decision (binary) variables are defined to decide on the task’s allocation.

- \( A_{i,k} \): binary variable that is set to 1 if and only if task \( \tau_i \) is assigned to server \( S_k \).

Since each task is assigned to exactly one server, the following constraint can be derived:

**Constraint 1.** \( \forall \tau_i \), \( \forall k \), \( \sum_{k=1}^{M} A_{i,k} = 1 \).

This variable can be derived from variables \( A_{i,k} \) by using the following constraint:

**Constraint 2.** \( \forall \tau_i, \forall k \), \( V_{i,j} \geq 1 - (2 - A_{i,k} - A_{j,k}) \).

Proof. If tasks \( \tau_i \) and \( \tau_j \) are assigned to the same server \( S_k \), then \( A_{i,k} = A_{j,k} = 1 \), so obtaining \( V_{i,j} \geq 1 \). If \( \tau_i \) and \( \tau_j \) are assigned to different servers, we have \( A_{i,k} \neq A_{j,k}, \forall k \), which results in \( (2 - A_{i,k} - A_{j,k}) \geq 1, \forall k \) so degenerating the constraint to \( V_{i,j} \geq 1 \).

### 4.1.2 Resource sharing - variables and constraints

Blocking times related to resource sharing are crucial to express the task schedulability as a constraint of the optimization problem.

First of all, decision variables are provided to identify which resource can cause arrival blocking on tasks.

- \( Z_{i,t} \): binary variable that is set to 1 if and only if resource \( R_t \) can cause arrival blocking on task \( \tau_i \).

To define these variables it is necessary to distinguish between component and system resources. If \( R_t \) is a component resource, it can result in a local or a global resource depending on the allocation of tasks using \( R_t \). If a component resource \( R_t \) results in a local resource, the arrival blocking is equal to the classical case of uniprocessor SRP, leading to

**Constraint 3.** \( \forall \tau_i, \forall R_t, \forall \tau_L \), \( D_L > D_i \land \eta_{L,t} > 0 \), \( \forall \tau_H \), \( D_H \leq D_i \land \eta_{H,t} > 0 \), \( Z_{i,t} \geq 1 - (2 - V_{i,L} - V_{i,H}) \).

**Proof.** This constraint derives directly from the expression of local blocking for SRP reported in Equation (7). Being a blocking related to local resources, tasks \( \tau_i, \tau_L \) and \( \tau_H \) are involved only if they are allocated on the same virtual processor: hence, the term \( (2 - V_{i,L} - V_{i,H}) \) is designed to be zero if and only if all the three tasks are allocated on the same virtual processor (as done in Constraint 2); otherwise it results in a positive value, thus degenerating the constraint to \( Z_{i,t} \geq 0 \).

If a component resource \( R_t \) results in a global resource, it will be protected by a FIFO non-preemptive spinlock generating non-preemptive blocking on tasks. Such a kind of blocking is accounted through the following constraint:

**Constraint 4.** \( \forall \tau_i, \forall R_t, \forall \tau_L \), \( D_L > D_i \land \eta_{L,t} > 0 \), \( \forall \tau_H \), \( \eta_{H,t} > 0 \), \( Z_{i,t} \geq 1 - (1 - V_{i,L} - V_{i,H}) \).

**Proof.** A task \( \tau_i \) can incur in arrival blocking if a task \( \tau_L \) with deadline greater than \( \tau_i \) \( (D_L > D_i) \) is accessing a global resource \( R_t \); this is because \( \tau_i \) will use non-preemptive spinning and a non-preemptive critical section for \( R_t \). First of all, this is possible only if \( \tau_i \) and \( \tau_L \) are allocated to the same virtual processor: to this purpose, the term \( (1 - V_{i,L}) \) will cause a degeneration of the constraint to \( Z_{i,t} \geq 0 \) if this is not true. At the same time, there must exist a task \( \tau_H \) accessing \( R_t \) that is not allocated on the virtual processor of \( \tau_i \) and \( \tau_L \), otherwise \( R_t \) would not be a global resource: to this purpose, the term \( -V_{i,H} \) is provided to not have arrival blocking if such a task \( \tau_H \) does not exist.

Consider now system resources. These resources are accessed from all the components; hence, a safe bound for their blocking is computed by assuming that they will always result in global resources, independently of the task allocation on the other processors. Overall, a system resource \( R_t \) leads to definition of the following constraint:

**Constraint 5.** \( \forall \tau_i, \forall R_t, \forall \tau_L \), \( D_L > D_i \land \eta_{L,t} > 0 \), \( Z_{i,t} \geq 1 - (1 - V_{i,L}) \).

**Proof.** As discussed in Section 3, since system resources are accessed by all the components, we have to assume that they will always result in global resources (thus involving non-preemptive spinning and non-preemptive critical sections). Hence, to have arrival blocking on a task \( \tau_i \), it is sufficient to have a task \( \tau_L \) executing on the same virtual processor of \( \tau_i \), having \( D_L > D_i \) and accessing a system resource \( R_t \). The term \( -(1 - V_{i,L}) \) is provided to degenerate the constraint to \( Z_{i,t} \geq 0 \) if such a task \( \tau_L \) does not exist; otherwise \( Z_{i,t} \geq 1 \), thus expressing the presence of arrival blocking.

We now provide constraints to quantify the arrival blocking on tasks, expressed through the following variables for the optimization problem:

- \( B_{i,t} \): real variable expressing a lower-bound on the arrival blocking imposed on \( \tau_i \) by critical sections on resource \( R_t \).
• $B_{i,k,ℓ}$: real variable expressing a lower-bound on the arrival blocking imposed on $τ_{i}$ by critical sections on resource $R_{ℓ}$, executed by tasks running on processor $P_{k}$.

The blocking $B_{i,k,ℓ}$ can be computed as

**Constraint 6.** $∀τ_{i}, ∀R_{ℓ}, ∀P_{k}, B_{i,k,ℓ} = \sum_{k=1}^{M} B_{i,k,ℓ}$.

The key part consists in providing constraints for the contributions of arrival blocking $B_{i,k,ℓ}$ coming from each processor. Also in this case, we have to distinguish between component and system resources. Considering component resources $R_{ℓ}$, the blocking contribution $B_{i,k,ℓ}$ coming from the same virtual processor on which task $τ_{i}$ is allocated can be expressed as follows:

**Constraint 7.** $∀τ_{i}, ∀R_{ℓ}, ∀P_{k}, ∀τ_{L} \mid D_{L} > D_{i} ∧ η_{L,ℓ} > 0, B_{i,k,ℓ} ≥ δ_{L,ℓ} - δ_{L,ℓ} \cdot (1 - Z_{L,ℓ}) - δ_{L,ℓ} \cdot (1 - A_{L,k}) - δ_{L,ℓ} \cdot (1 - A_{i,k})$.

**Proof.** This constraint expresses the blocking contribution for $τ_{i}$ coming from the virtual processor at which such a task is located. Concerning arrival blocking, a task $τ_{i}$ can be blocked by critical sections of tasks $τ_{L}$ (i.e., $δ_{L,ℓ}$), either in the case $R_{ℓ}$ is a local resource (imposing SRP blocking) or a global resource (imposing non-preemptive blocking).

Each term preceded by a minus is provided to impose zero blocking depending on decision variables, thus degenerating the constraint to $B_{i,k,ℓ} ≥ 0$. First of all, the blocking has to be zero if $R_{ℓ}$ cannot impose arrival blocking on $τ_{i}$; this is enforced by the term $-δ_{L,ℓ} \cdot (1 - Z_{L,ℓ})$ which becomes equal to $-δ_{L,ℓ}$ if $Z_{L,ℓ} = 0$. Without loss of generality we assume that $τ_{i}$ is allocated on $S_{k}$. Then, the arrival blocking from such a server exists only if $τ_{i}$ and $τ_{L}$ are allocated on $S_{k}$; to this purpose, the terms $-δ_{L,ℓ} \cdot (1 - A_{L,k})$ and $-δ_{L,ℓ} \cdot (1 - A_{i,k})$ are provided to degenerate the constraint to $B_{i,k,ℓ} ≥ 0$ if this is not true.

If a component resource $R_{ℓ}$ results in a global resource, there is a blocking contribution coming from other virtual processors (i.e., the ones at which $τ_{i}$ is not allocated). These blocking contributions are reflected as non-preemptive spinning that prevents $τ_{i}$ from executing. A lower-bound on such a blocking $B_{i,k,ℓ}$ can be formulated as follows:

**Constraint 8.** $∀τ_{i}, ∀R_{ℓ}, ∀P_{k}, ∀τ_{L} \mid η_{L,ℓ} > 0, B_{i,k,ℓ} ≥ δ_{L,ℓ} - δ_{L,ℓ} \cdot (1 - Z_{L,ℓ}) - δ_{L,ℓ} \cdot (1 - A_{x,k}) - δ_{L,ℓ} \cdot A_{i,k}$.

**Proof.** Each term preceded by a minus is provided to impose zero blocking depending on decision variables, thus degenerating the constraint to $B_{i,k,ℓ} ≥ 0$. Like Constraint 7, the blocking has to be zero if $R_{ℓ}$ cannot impose arrival blocking on $τ_{i}$; this is enforced by the term $-δ_{L,ℓ} \cdot (1 - Z_{L,ℓ})$. If a component resource $R_{ℓ}$ results in a global resource, there exists a remote task $τ_{L}$ accessing $R_{ℓ}$ that is allocated on a virtual processor different from the one of $τ_{i}$. Without loss of generality, assume that $S_{k}$ is the virtual processor at which $τ_{i}$ is allocated. The remote task $τ_{L}$ has to be allocated on a virtual processor different from $S_{k}$; hence the term $-δ_{L,ℓ} \cdot (1 - A_{x,k})$ is provided to have zero blocking if this is not true. Similarly, the term $-δ_{L,ℓ} \cdot A_{i,k}$ enforces zero blocking from the same virtual processor $S_{k}$ at which $τ_{i}$ is allocated.

When blocking on a **system resource** $R_{ℓ}$ is considered, we assume that a critical section of maximum length $H$ is present on each virtual processor (please refer to Section 3), so obtaining the following constraint:

**Constraint 9.** $∀τ_{i}, ∀R_{ℓ}, ∀P_{k}, B_{i,k,ℓ} ≥ H - H \cdot (1 - Z_{i,ℓ}) - H \cdot A_{i,k}$.

**Proof.** Each term preceded by a minus is provided to impose zero blocking depending on decision variables, thus degenerating the constraint to $B_{i,k,ℓ} ≥ 0$. First of all, the term $-H \cdot (1 - Z_{i,ℓ})$ is provided to have zero blocking if task $τ_{i}$ cannot incur in arrival blocking related to $R_{ℓ}$. Then, we have a blocking contribution equal to the maximum critical section length $H$ on each remote processor, that is, all processors except the one at which $τ_{i}$ is allocated, which is excluded through the term $-H \cdot A_{i,k}$.

As stated in Section 3, the use of non-preemptive FIFO spinlocks is accounted by inflating tasks WCETs with the spinning time $ξ_{i,k}$ generated by remote blocking. We decompose the spinning time $ξ_{i,k}$ of a task $τ_{i}$ by using the following variables:

• $ξ_{i,k}$: real variable expressing a lower-bound on the spinning time for task $τ_{i}$ originated from processor $P_{k}$;

• $ξ_{i,k,ℓ}$: real variable expressing the contribution of the spinning time $ξ_{i,k}$ in accessing the resource $R_{ℓ}$ originated from processor $P_{k}$.

The per-processor spinning time $ξ_{i,k}$ can be expressed as

**Constraint 10.** $∀τ_{i}, ∀P_{k}, ξ_{i,k} = \sum_{k=1}^{M} ξ_{i,k}$.

Similarly, the overall spinning time $ξ_{i}$ of a task $τ_{i}$ is formulated as

**Constraint 11.** $∀τ_{i}, ξ_{i} = \sum_{k=1}^{M} ξ_{i,k}$.

Then, the key part consists in identifying a lower-bound on the spinning time $ξ_{i,k,ℓ}$. Again, it necessary to distinguish between component and system resources. For a component resource $R_{ℓ}$, the following constraint is used:

**Constraint 12.** $∀τ_{i}, ∀P_{k}, ∀R_{ℓ}, ∀τ_{r} \mid τ_{r} ≠ τ_{i}, ξ_{i,k,ℓ} ≥ δ_{x,ℓ} \cdot η_{i,ℓ} \cdot A_{x,k} - B \cdot A_{i,k}$.

**Proof.** This constraint derives directly from the computation of the spinning time for component resources, as defined in Equations (4) and (5). The constraint collects the maximum critical section on $R_{ℓ}$ of tasks $τ_{r}$ allocated on virtual processor $S_{k}$. To account for the overall spinning time, the critical section length is multiplied for the number of critical sections on $R_{ℓ}$ for $τ_{r}$ (see Equation (5)). Thanks to the decision variable $A_{x,k}$, the first term becomes zero if task $τ_{r}$ is not allocated on server $S_{k}$. The remaining term $-B \cdot A_{i,k}$ is provided to have zero spinning time contribution from the same processor at which $τ_{i}$ is allocated (critical sections executed on the same processor of $τ_{i}$ do not cause spinning). In this case, $B$ represents a numerically large constant that dominates all possible values for the term $δ_{x,ℓ} \cdot η_{i,ℓ}$, and can be formally defined as $B = \max_{x,ℓ}(δ_{x,ℓ} \cdot \max_{i,ℓ}(η_{i,ℓ}))$.

When a **system resource** $R_{ℓ}$ is considered, the spinning time can be expressed as follows:

**Constraint 13.** $∀τ_{i}, ∀P_{k}, ∀R_{ℓ}, ∀τ_{r} \mid τ_{r} ≠ τ_{i}, ξ_{i,k,ℓ} ≥ H \cdot η_{i,ℓ} \cdot A_{x,k} - B \cdot A_{i,k}$. 


Proof. The proof follows from the one of Constraint 12, assuming critical sections of length \( \mathcal{H} \).

4.1.3 Schedulability - variables and constraints

This section presents the constraints for the optimization problem expressing the schedulability of a task set upon a reservation server. As stated in Section 3, the local schedulability upon an M-BROE server can be checked by Equation (9); however, as expressed at the beginning of Section 4, the exact test is not easily tractable in an optimization problem. To solve this problem, the workload of a task set is approximated using the FPTAS [19] approach. According to the FPTAS, the demand bound function is expressed as

\[
\text{dbf}_i(t) = \begin{cases} 
\text{dbf}_i(t), & \text{if } t \leq (\lambda - 1)T_i + D_i \\
C_i + \xi_i + (t - D_i)U_i, & \text{otherwise,}
\end{cases}
\]

where \( U_i = (C_i + \xi_i)/T_i \), to account for the WCET inflation related to the use of spinlocks.

Using this approximation, the EDF schedulability has to be considered in \( \lambda + 1 \) time points for each task. The resulting EDF schedulability test for a set of tasks \( T \) is expressed as follows:

\[
\forall p = 0, 1, \ldots, \lambda, \forall t \in \text{tSet}(p), \quad B(t, p) + \sum_{\tau_i \in T} \text{dbf}_i'(t) \leq \text{sbf}(t),
\]

where \( \text{tSet}(p) \) is the set of schedulability check-points [19] defined as

\[
\text{tSet}(p) = \bigcup_{\tau_i \in T} \{ pT_i + D_i \},
\]

and \( B(t, p) \) is defined to approximate the blocking term of Equation (11) as

\[
B(t, p) = \begin{cases} 
B(t), & \text{if } 0 \leq p < \lambda \\
\max\{ B_j \}, & p = \lambda.
\end{cases}
\]

As stated by Baruah in [4], the blocking term \( B(t) \) is zero for values of \( t \) larger than the maximum relative deadline of the tasks under analysis. As a consequence, the exact blocking function \( B(t) \) can be used (i.e., \( B(t, p) = B(t) \), \( \forall p \geq 0 \)) in the MILP formulation if a sufficiently large number \( \lambda \) of check-points is used for the FPTAS.

We now define a set of variables and constraints to express the EDF schedulability according to the FPTAS approach. All the variables contain the processor index \( k \), since the schedulability has to be checked for each processor addressing a partitioned scheduling scheme. First of all, we introduce the following variables to account for the WCET inflation related to spinlocks:

- \( J_{i,k} \): real variable expressing the inflated WCET of a task \( \tau_i \) executing on virtual processor \( S_k \).

Such a variable can be defined by using the following constraint:

\[
\text{Constraint 14. } \forall \tau_i, \forall P_k, \quad J_{i,k} = C_i + \xi_i - B \cdot (1 - A_{i,k}).
\]

Proof. This constraint simply adds \( C_i \) to the overall spinning time \( \xi_i \). Term \(-B \cdot (1 - A_{i,k})\) is provided to have a null inflated WCET if \( \tau_i \) is not allocated to server \( S_k \) (i.e., when \( A_{i,k} = 0 \)). \( B \) represents a numerically large constant that dominates all possible values of the term \( C_i + \xi_i \), and can be defined as \( B = \max\{ C_i \cdot (M - 1)H \cdot \max\{ \eta_{i,t} \} \} \).

Now note that Equations (14) involves the blocking time \( B(t) \), defined in Equation (16). Having to express the schedulability in a limited number of time points, we introduce variables to quantify the blocking time at the \( p \)th time point of a task:

- \( PB_{k,p,j} \): real variable expressing the blocking time on virtual processor \( S_k \) at schedulability check-point \( pT_j + D_j \) of task \( \tau_i \), with \( p = 0, 1, \ldots, \lambda \).

Such a blocking time is expressed by the following constraint, making use of the blocking time \( B_{i,t} \) expressed in Constraint 6:

\[
\text{Constraint 15. } \forall S_k, \forall p = 0, 1, \ldots, \lambda, \forall \tau_i, \forall R_i
\]

\[
\forall \tau_i \mid D_i \leq pT_j + D_j \land p < \lambda \quad PB_{k,p,j} \geq B_{i,t} - B \cdot (1 - A_{i,k}).
\]

Proof. According Equations (11) and (16), the blocking at the schedulability check-point \( t \) involves blocking times of tasks having deadlines less than or equal to \( t \) for \( p = 0, 1, \ldots, \lambda \). The \( p \)th check-point originated from \( \tau_i \) is \( pT_j + D_j \) (see Equation (15)); this constraint considers all tasks \( \tau_i \) having \( D_i \leq pT_j + D_j \) (thus contributing to blocking) excluding the ones that are not allocated on virtual processor \( S_k \). Such tasks are excluded through the term \(-B \cdot (1 - A_{i,k})\), where \( B \) is a numerically large constant that dominates all possible values for the term \( B_{i,t} \) and can be formally defined as \( B = M \cdot H \). Similarly, all tasks \( \tau_i \) allocated to \( S_k \) are considered for \( p = \lambda \) accounting for the maximum blocking on \( S_k \).

The computational supply provided by each virtual processor \( S_k \) is approximated by assuming ideal (fluid) virtual processors with bandwidth \( \alpha_k \). The following variables are introduced to support this choice:

- \( \alpha_k \): real variable representing the bandwidth of virtual processor \( S_k \), with \( 0 \leq \alpha_k \leq 1 \).

At this point we have all the variables and the constraints to express the EDF schedulability on each virtual processor \( S_k \):

\[
\text{Constraint 16. } \forall S_k, \forall p = 0, \ldots, \lambda, \forall \tau_j
\]

\[
PB_{k,p,j} + \sum_{\tau_{i,k} \in T} \text{dbf}_{i,k}(pT_j + D_j) \leq \alpha_k \cdot (pT_j + D_j),
\]

where

\[
\text{dbf}_{i,k}(t) = \begin{cases} 
\left\lceil \frac{t - D_i}{T_i} \right\rceil + 1 \cdot J_{i,k}, & \text{if } t \leq (\lambda - 1)T_i + D_i \\
J_{i,k} + (t - D_i) \cdot \frac{J_{i,k}}{T_i}, & \text{otherwise}.
\end{cases}
\]

Proof. This constraint derives directly from the FPTAS schedulability test in Equation (14), for each virtual processor \( S_k \). Thanks to the definition of variables \( J_{i,k} \), the contribution in terms of demand bound function of tasks \( \tau_i \) is null if \( \tau_i \) is not assigned to virtual processor \( S_k \).
4.1.4 Objectives
We now propose two alternative allocation strategies for the optimization problem, aiming at different objectives. The first one, denoted as (A), aims at allocating the tasks of a component on a small set of virtual processors having high bandwidth; the second one, denoted as (B), aims at distributing tasks among a larger set of virtual processors with lower bandwidth. Clearly, each allocation strategy leads to a different instance of the component interface. We now formalize the objectives for both strategies.

- Strategy (A): minimize the overall bandwidth required by a software component, that is the sum of the bandwidths required by its virtual processors:

  \[ \text{minimize } \sum_{k=1}^{M} \alpha_k. \]  

- Strategy (B): minimize the maximum bandwidth required by the virtual processor of a component:

  \[ \text{minimize } \max \{ \alpha_k \} = \text{minimize } \Lambda, \]  

where \( \Lambda \) is an additional real variable of the optimization problem defined by the following constraint:

**Constraint 17.** \( \forall k = 1, \ldots, M \quad \Lambda \geq \alpha_k. \)

4.2 Interface synthesis
Given the allocation of the component tasks to the virtual platform, produced by the MILP solution, the design of the reservation server parameters is performed using the approach presented in [17]. Such an approach computes the optimal budget and period (under the assumed scheduling infrastructure) that guarantee the application schedulability while minimizing the bandwidth for each virtual processor, taking server context switch overhead into account. The resulting server parameters for each virtual processor constitute the component interface exported to the component integrator.

5. VIRTUAL PROCESSOR ALLOCATION
The goal of this section is to propose a methodology to partition the virtual processors of all the components to the \( M \) physical processors, that is the task performed by the component integrator. This is done by a MILP optimization program formulation that is able to find an exact solution for the allocation problem, based on the interfaces exported by the components.

As done for the task allocation problem presented in Section 4, we introduce a set of (binary) decision variables to decide on the virtual processors allocation:

- \( A_{j,k}^{l} \): binary variable that is set to 1 if and only if the reservation \( S_{j}^{l} \) of component \( \Gamma_{j} \) is assigned to physical processor \( P_{k} \).

Since a reservation must be assigned to only one processor, the following constraint can be derived:

**Integration Constraint 1.** \( \forall S_{j}^{l}, \ \sum_{k=1}^{M} A_{j,k}^{l} = 1. \)

To express the schedulability at the integration level in the optimization problem formulation, we have to derive constraints from the test reported in Equation (12). The idea is to provide variables and constraints representing the contribution of bandwidth on each physical processor; such variables will be defined to be equal to zero if a virtual processor is not allocated to a processor:

- \( Q_{j,k}^{l} \): real variable representing the budget of server \( S_{j}^{l} \) executing on processor \( P_{k} \). Such a budget will be zero if \( P(S_{j}^{l}) \neq P_{k} \).
- \( B_{j,k}^{l} \): real variable representing the blocking imposed on server \( S_{j}^{l} \), executing on processor \( P_{k} \). Such a blocking will be zero if \( P(S_{j}^{l}) \neq P_{k} \).

The following constraints formulate the value for these variables.

**Integration Constraint 2.**

\[ \forall k = 1, \ldots, M, \ \forall S_{j}^{l}, \quad Q_{j,k}^{l} \geq Q_{j}^{l} - B \cdot (1 - A_{j,k}^{l}). \]

**Proof.** The term \(-B \cdot (1 - A_{j,k}^{l})\) is provided to degenerate the constraint to \( Q_{j,k}^{l} \geq 0 \) if virtual processor \( S_{j}^{l} \) is not allocated to physical processor \( P_{k} \), where \( B \) is a numerically large constant formally defined as \( B = \max_{j} \{ Q_{j}^{l} \} \).

**Integration Constraint 3.**

\[ \forall k = 1, \ldots, M, \ \forall S_{j}^{l}, \quad B_{j,k}^{l} \geq B_{j}^{l} - B \cdot (1 - A_{j,k}^{l}). \]

**Proof.** It follows from the proof of Integration Constraint 2, with \( B = M \). Now it is possible to present the main constraint expressing the schedulability of the reservation servers.

**Integration Constraint 4.**

\[ \forall k = 1, \ldots, M, \ \forall S_{j}^{l}, \quad \sum_{r,w,p_{r}^{w} \leq p_{r}^{w}} \frac{Q_{r,k}^{w}}{P_{r}^{w}} + \frac{B_{j,k}^{l}}{P_{j}^{l}} \leq 1. \]

**Proof.** It follows directly from Equation (12) and Integration Constraints 2 and 6. If a reservation server is not assigned to processor \( P_{k} \), its contribution to the sum will be zero.

As stated in Section 3, the blocking factor \( B_{j}^{l} \) can be considered constant and equal to \( B_{j}^{l} = M \). If a more complex component interface is used (e.g., the one proposed in Section VI of [14]), it is possible to setup additional variables and constraints to compute allocation-dependent blocking times, as done in Section 4 through Constraints 3-9. This extension is not addressed in this paper for space limitations.

6. EXPERIMENTAL RESULTS
This section presents some experimental results aimed at evaluating the proposed methodology. Experiments have been conducted to (i) evaluate the performance in terms of schedulability ratio for the whole methodology, and (ii) measure the run time of the procedure for partitioning an application upon a virtual processor. The proposed MILP formulations have been implemented with the IBM CPLEX solver running on an 8-core Intel Xeon at 2.8 GHz.

6.1 Workload generation
Given an overall system utilization \( U \), the utilizations \( U_{k} \) of the \( N \) components have been generated using the UUniform fast [13] algorithm, limiting their values in the range \([0, 15, 1.5]\). All random variables were generated with uniform distribution in a given range. The task set \( T_{k} \) in each component \( \Gamma_{k} \) was generated by fixing a random number of
tasks \( n_k \in [4, 8] \), each with utilization \( u_i \leq 0.8 \) generated by UUnifast. Tasks periods \( T_i \) were randomly chosen in the set \([5, 10, 20, 30, 50, 80, 100, 150, 200]\) ms, and tasks computation times were computed as \( C_i = u_i T_i \). We assumed the presence of \( NR^S \) system resources and \( NR^C \) component resources. For each resource \( R_c \), a random number of tasks in the range \([1, rsf \cdot n_k] \) was selected to access \( R_c \). The rsf parameter (resource sharing factor) indicates how many tasks use a resource. For each task \( \tau_i \) accessing \( R_c \), we generated \( \eta_{i, \ell} \in [1, \eta^\text{MAX}] \) critical sections of length \( \delta_{i, \ell} \in (0, \mathcal{H}] \). To have realistic task sets, we enforced \( C_i > \sum R_c (\eta_{i, \ell} \cdot \delta_{i, \ell}), \forall \tau_i \in T_k \).

### 6.2 Experiment 1

This experiment was carried out to evaluate the performance of the proposed approach comparing the two allocation strategies described in Section 4 (here denoted as MILP-A and MILP-B), and a mixed strategy, denoted as A \( \lor \) B, which selects the best of the two interfaces (resulting from strategy A and B), in terms of component integrations schedulability. Simulations consider systems composed of \( N = 5 \) components to be executed on a physical platform including \( M = 4 \) processors. The performance of the proposed methodology was evaluated by measuring the schedulability ratio (i.e., the ratio of schedulable systems and the total number of generated systems) as a function of the system utilization \( U \), varied from 0.75 to \( M \) with step 0.25. For each value of \( U \), the schedulability ratio was computed over 250 systems, hence testing 1250 components and generating a number of tasks between 5000 and 10000.

Figure 2 reports the results of this experiment obtained with the following parameters: \( NR^S = 2 \), \( NR^C = 2 \), \( \lambda = 30 \), \( rsf=0.3 \), \( \eta^\text{MAX} = 2 \), \( \mathcal{H} = 100\mu s \).

![Figure 2: Schedulability ratio as a function of the system utilization \( U \).](image)

Note that, for this particular setting of parameters, MILP-A is slightly more effective than MILP-B for all system utilizations, and both of them are able to guarantee more than 80 percent of the generated systems for overall utilizations up to 3. As expected, the mixed strategy (A \( \lor \) B) outperforms the others, being able to admit more than 40 percent of the generated systems for utilizations up to 3.5.

To evaluate the impact of resource sharing on the schedulability ratio, we performed another test with an increased number of resources \( NR^S = 3 \), \( NR^C = 3 \). The result of this test is reported in Figure 3. As can be noted from the graphs, the performance of both MILP-A and MILP-B is slightly degraded, while the mixed strategy (A \( \lor \) B) does not show a significant degradation.

![Figure 3: Schedulability ratio as a function of the system utilization \( U \).](image)

### 6.3 Experiment 2

In this experiment we measured the run time needed to solve the MILP formulation for task partitioning as a function of the number of tasks \( n \) in a component. For each value of \( n \), the average and the maximum run time was measured over 500 randomly generated components. Figure 4 reports run time taken by strategies (A) and (B). The results are collected under the same parameter configuration used in Figure 2. The maximum run time shows an exponential trend as the number of tasks increases. This result is expected since the number of the variables on the MILP formulation increases with the number of tasks.

![Figure 4: Average and maximum run time for solving the MILP formulation for task partitioning with strategies (A) and (B).](image)

### 7. CONCLUSIONS AND FUTURE WORK

This paper presented a component-based design methodology for supporting the integration of independently developed real-time applications upon multiprocessor platforms in the presence of shared resources. Applications consists of a set of periodic and sporadic real-time tasks that can use both component and system-level resources. The physical platform is abstracted through a number of virtual platforms, one for each component, each consisting of a set of virtual processors implemented by reservation servers.

The proposed methodology uses an MILP formulation to partition each application upon the virtual multiprocessor
platform taking shared resources into account. Once an allocation is found for each component, the synthesis of the virtual processors is performed to find the optimal reservation parameters that can guarantee the schedulability of the applications. Then, a component integrator uses an MILP formulation for allocating all virtual processors to the physical processors to preserve the schedulability of the system.

Simulation experiments on synthetic applications have been carried out to validate the effectiveness of the approach. The achieved results showed that the proposed design methodology (in particular the mix of the proposed allocation strategies) is able to admit 90 percent of the generated systems having utilization up to 3 on a quad-processor platform, in the presence of shared resources and reservations.

As a future work we plan to investigate non-linear optimization problem formulations for the task partitioning, as well as resource sharing driven heuristics, hence proposing a comparison study through extensive simulation experiments.

Acknowledgements
This work has been partially supported by the RETINA Eurostars Project E310171 and received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement 688860. The authors like to thank Enrico Bini for the fruitful discussions that helped this work.

8. REFERENCES


