On Component-Based Software Development for Multiprocessor Real-Time Systems

Nima Khalilzad, Moris Behnam, Thomas Nolte
MRTC/Mälardalen University
Västerås, Sweden
nima.m.khalilzad@mdh.se

***Abstract*—Component-based software development provides a modular approach to develop complex software systems. In the context of real-time systems, it is desirable to abstract the timing properties of software components using an interface for each component. The timing properties of the whole system, composed of multiple components, is studied using the component interfaces. In this paper we focus on periodic interface models. In the case of components developed for single processor platforms, for examining the system schedulability, the interfaces can be regarded as periodic tasks. Thus, making it possible to use the conventional schedulability analyses for the system level schedulability test. In the case of components developed for multiprocessors, since interfaces may have utilization larger than 100 % of a single processor, it is not possible to directly use the component interfaces for the system schedulability test. Therefore, the interfaces have to be decomposed before performing the system level schedulability test.**

In this paper, we target the special case of partitioned EDF for scheduling the components integrated on a multiprocessor. Therefore, the system level schedulability test is equivalent to finding a feasible allocation of component interfaces on the multiprocessor. We propose two algorithms for allocating the multiprocessor periodic interfaces. In addition, we propose an orthogonal approach for developing component-based real-time systems on multiprocessors in which components with utilization more than 100 % of a single processor are divided into smaller subcomponents before abstracting their interfaces. We show, through extensive evaluations, that our alternative approach significantly reduces the interface overhead.

I. Introduction

Multiprocessor platforms provide a great amount of computational capacity on a single hardware. Therefore, it is possible to design and run large software systems on a single chip. Component-based software development facilitates the development process of large software systems. We consider component models in which a real-time software component is composed of multiple real-time tasks. In this approach, software components are developed independently, possibly by different teams, and later integrated. In the real-time systems arena, component-based development approaches (e.g. [1], [2]) often follow a two step process. Firstly, the processor demand of the tasks within each component is abstracted. This step produces component interfaces. Secondly, the components are integrated and their schedulability is studied using the component interfaces. Abstracting the requirements of components comes at a price. There is often a gap between the processor utilization of the component interfaces and the utilization of the task set within components. This gap (henceforward referred as the abstraction overhead) results in a processor utilization efficiency loss. Efficient utilization of the processor resource is particularly important in resource constrained embedded systems. To this end, it is important to study the abstraction overhead of different approaches to understand their practical applicability.

In this paper we focus on a periodic interface model, namely the Multiprocessor Periodic Resource (MPR) model [3]. The reason behind focusing on the periodic models is that they can easily be implemented in practice. The schedulability of systems, composed of multiple components, is investigated through studying the MPR interfaces. It is desirable to use the same schedulability techniques used for studying the schedulability of real-time tasks, and investigate the schedulability of the components. However, the task schedulability tests cannot be directly applied to the components for which their interface utilizations are more than 100 % of a single processor (i.e. one). This is because the basic assumption in all of the schedulability tests is that the task utilization is less than or equal to one. Therefore, components with interface utilization more than one have to be decomposed to smaller subcomponents with utilization less than or equal to one. The component schedulability test, then, can be performed using the decomposed subcomponent interfaces. In this paper we use the partitioned Earliest Deadline First (pEDF) [4] algorithm for scheduling the components. We propose two algorithms which perform decomposition and allocation simultaneously, each algorithm following a different objective.

In all of the proposed approaches for developing component-based real-time systems on multiprocessors (e.g. [3], [5], [2]) the component decomposition is performed after abstracting the components. In this paper, we investigate an alternative approach. We first decompose components for which their utilization is more than one. Thereafter, we abstract the component processor requirements using an abstraction technique proposed in [6]. We show that, using extensive simulations, performing the decomposition before abstraction significantly reduces the abstraction overhead. Also, we provide three integration algorithms for components abstracted using our approach, i.e., decomposed before the abstraction. Finally, using extensive simulations, we compare the number of processors required for integrating the components developed using the two alternative approaches. We compare the performance of the proposed integration algorithms within each approach.

The research leading to this paper has received funding from the Swedish Research Council (Vetenskapsrådet) under the project ARROWS, the Swedish Foundation for Strategic Research (SSF) via the research project PRESS and the Swedish Knowledge Foundation through the project RV-RED.

Contributions. In this paper we study the complete process of component-based development approaches for real-time systems (focusing only on timing properties) from component abstraction to the system integration using periodic interfaces. We present the following contributions. (i) We propose two integration algorithms for integrating components abstracted using the MPR model. (ii) We propose a new approach in which component decomposition is performed before abstraction. We propose a new interface model as well as three integration algorithms for this new approach. (iii) We present the result of our extensive simulations comparing the approach based on the MPR abstraction model with our alternative approach.

II. SYSTEM MODEL AND DEVELOPMENT APPROACHES

System and task model. We assume a multiprocessor platform with m homogeneous processors. n components are composed on the multiprocessor platform. The slack bandwidth of the j^{th} processor is denoted using S_j. We assume a constrained deadline periodic task model in which the k^{th} task τ_k is characterized using period $T_k \in \mathbb{N}^+$, deadline $D_k \in \mathbb{N}^+$ and Worst-Case Execution Time (WCET) $C_k \in \mathbb{N}^+$ ($C_k \leq D_k \leq T_k$). We assume that the tasks are independent, i.e., except the processor resource, they do not share any other resources.

Scheduling scheme. We assume a hierarchical scheduling scheme in which the scheduling is performed in two levels. At the global level components are scheduled using a *component-scheduler*. In this paper we use pEDF for scheduling the components. Within the components, however, a *task-scheduler* coordinates the execution of the tasks. In the case that a component is assigned to one processor we use EDF as the task-scheduler. When a component is spread over multiple processors, we use global EDF (gEDF) [4] as the task-scheduler. From a resource provisioning vantage point, the multiprocessor resource is partitioned in the time domain. Each component is assigned to a multiprocessor partition which indeed provisions a fraction of the multiprocessor resource to the component. The components, then, distribute their fraction of the resource among their inner tasks.

Component-based system development. Component-based development approaches often consider the following two roles for the system development: (i) component developer (ii) system integrator. The component developers develop a task set, and they select a suitable task-scheduler. They also calculate the component interface based on the task set and the task-scheduling algorithm. The component interface indicates the amount and the specifications of the required processor resource fraction. This approach enables independent development of components by different development teams. The system integrator, on the other hand, receives a set of component interfaces. The system integrators use the component interfaces to examine the schedulability of the system. If the component requires a processor fraction more than one, then the integrator divides the component into a number of subcomponents. This step is referred as the transformation of interfaces into interface-tasks in the previous approaches (e.g. [3], [2]). The reason behind performing this transformation is that it is desirable to use the conventional task schedulability analyses for examining the schedulability of the system. The basic assumption in such analyses is that the utilization of tasks is less than or equal to one. Therefore, in order to perform schedulability test using the interfaces, we require interfaces in which their utilization is less than or equal to one. We use the word "decomposition" to refer to the step in which a large component is divided into a number of smaller subcomponents. After the decomposition step, the interfaces of the subcomponents can be used for performing the schedulability test.

Component model. We assume that component $\mathcal{C}_i$ is composed of a set of tasks denoted by $\mathcal{T}_i$. We use $U_{\mathcal{T}_i}$ to denote the task set utilization of $\mathcal{C}_i$. The components are assigned to the processors at the integration phase. We use $\rho_{i,j}$ to denote the amount of the utilization of $\mathcal{C}_i$ that is allocated on the j^{th} processor. In this paper we target components for which their utilizations are more than one $U_{\mathcal{T}_i} > 1$, i.e. they require more than one processor for performing their computations. The following two alternative approaches can be used when dealing with such components. (i) First Abstraction, then Decomposition (FAD): in this approach the component developers first abstract the resource requirements of the entire component. The system integrators have to divide the component into a number of subcomponents at the integration phase.(ii) First Decomposition, then Abstraction (FDA): the component developers first divide the component into a number of subcomponents such that all subcomponents have utilization less than or equal to one. The subcomponent interfaces are then derived by the component developers.We use $\mathcal{C}_{i,r}$ to denote the r^{th} subcomponent of component i. Similar to the components, we use $\rho_{i,r,j}$ to denote the amount of the utilization of $\mathcal{C}_{i,r}$ that is allocated on the j^{th} processor. In the following we explore the above two alternatives and we compare the implications of using each approach.

The FAD approach. In this approach the processor requirements of a component is abstracted using a single interface. The interface essentially indicates the fraction of the multiprocessor capacity required by the component. In doing so, it is assumed that the tasks within one component are allowed to migrate among processors, i.e., they are scheduled using a global multiprocessor scheduling policy. For instance, in the approach proposed by Easwaran *et al.* [3], the MPR model is used for abstracting the processor requirements of the components. In this model the interface of the i^{th} component is denoted by $\Gamma_i^{m'_i} < \Pi_i, \Theta_i >$ where $\Pi_i \in \mathbb{N}^+$, $\Theta_i \in \mathbb{N}^+$ and $m'_i \in \mathbb{N}^+$ characterize the period, total budget and the parallelism level of the component. The parallelism level indicates the maximum number of processors that can contribute in providing the total budget to $\mathcal{C}_i$. The MPR interface imposes the following constraints by definition: $1 \leq m'_i \leq m$ and $\Theta_i \leq m'_i \times \Pi_i$. The fraction of the required multiprocessor, i.e the interface utilization, is denoted using:

$$U_{\Gamma_i^{m'_i}} = \frac{\Theta_i}{\Pi_i}.$$

This model provides a great deal of flexibility at the integration phase. This is because the total utilization can be provided using any m'_i processors. Therefore, the integrators can use the processors' slack status to decide on how to perform the decomposition. For instance, assume that we have two processors with the following slacks $S_1 = S_2 = 0.55$. Suppose that a new component is being integrated with $U_{\Gamma_i^{m'_i}} = 1.1$

and $m'_i = 2$. The only decomposition that can deem the system schedulable, is to divide the component into two subcomponents each with utilization equal to 0.55. The only problem with this model is that it incurs a considerable amount of abstraction overhead (see Section IV). Therefore, in the following we investigate an alternative approach in which we allow the system integrators to trade-off the integration flexibility with the abstraction overhead.

The FDA approach. In this approach the component developers are responsible to decompose the components, for which their utilization is more than one, into a number of subcomponents. The system integrator, then, can directly use the subcomponent interfaces to examine the schedulability of the system. We use the Periodic Resource (PR) [6] model for abstracting the processor requirements of the subcomponents. The PR model can be seen as a special case of the MPR model where $m'_i = 1$. Note that the fact that $m'_i = 1$ allows us to use a different analysis (i.e. single processor EDF schedulability) for deriving the subcomponent interfaces. The FDA approach does not provide any flexibility at the integration phase. This is because the utilization required by one subcomponent has to be provided using exactly one processor ($m'_i = 1$). For instance, assume that, similar to the previous example, we have two processors with the following slacks $S_1 = S_2 = 0.55$. The component developer has decomposed its large component into two subcomponents with utilizations equal to 0.65 and 0.45. Although the total component utilization is equal to the overall processor slack, it is not possible for the integrator to deem the system schedulable. This is because the decomposition is already performed before the abstraction, and the integrator has to perform the integration using the provided subcomponents.

The fact that the FDA approach does not provide flexibility at the integration phase may result in processor utilization loss. In order to mitigate this problem, we propose an altered modeling approach. In the new approach, after decomposing the large components, the component developer uses the PR model to abstract subcomponents' processor requirements. In addition, assuming that it may be impossible to fit one subcomponent in one processor at the integration phase, the component developer derives the MPR model for the subcomponents assuming $m' \in [2, m]$. We refer to this model as the Extended Periodic Resource (EPR) model in the rest of the paper. In the EPR model the component interfaces are denoted using the following matrix:

$$\Omega_i = \begin{bmatrix} \Gamma^1_{i,1} & \Gamma^1_{i,2} & \cdots & \Gamma^1_{i,p_i} \\ \Gamma^2_{i,1} & \Gamma^2_{i,2} & \cdots & \Gamma^2_{i,p_i} \\ \vdots & \vdots & \ddots & \vdots \\ \Gamma^m_{i,1} & \Gamma^m_{i,2} & \cdots & \Gamma^m_{i,p_i} \end{bmatrix},$$

where $\Gamma^j_{i,r}$ denotes the MPR interface of $\mathcal{C}_{i,k}$ given that its parallelism is equal to j. p_i represents the total number of subcomponents of $\mathcal{C}_i$. p_i depends on the decomposition algorithm which is addressed later in this section. The budget and the period of $\Gamma^j_{i,r}$ is denoted using $\Theta^j_{i,r}$ and $\Pi^j_{i,r}$ respectively. Ω_i allows integrators to select an interface which has a suitable parallelism level considering the processor slacks. If the slacks are scattered throughout the processors, then it may be beneficial to use an interface with a large parallelism level. However, this additional flexibility comes at a price. As it is shown in [3], increasing the parallelism level increases the utilization of the MPR interfaces. We use $\Delta^j_{i,k}$ to denote the difference of the utilization required by subcomponent $\mathcal{C}_{i,k}$ in parallelism level j and $j-1$, i.e.:

$$\Delta^j_{i,k} = \frac{\Theta^j_{i,r}}{\Pi^j_{i,r}} - \frac{\Theta^{j-1}_{i,r}}{\Pi^{j-1}_{i,r}},$$

where $\forall j < 1\ \Theta^j_{i,k} = 0$. Informally speaking, $\Delta^j_{i,k}$ denotes the amount of penalty that needs to be paid for gaining an additional level of flexibility.

The component decomposition algorithm takes one component $\mathcal{C}_i$ and decomposes it into a set of subcomponents $\{\mathcal{C}_{i,1}, \ldots, \mathcal{C}_{i,p_i}\}$. We use the following three bin packing heuristics for component decomposition: First Fit (FF), Best Fit (BF) and Worst Fit (WF) [4]. In the case of the WF heuristic, we assume that we have $\lceil U_{\mathcal{T}_i} \rceil$ available processors (i.e. $p_i = \lceil U_{\mathcal{T}_i} \rceil$). If the decomposition fails, then we add a new processor and reperform the decomposition.

III. Integration

In this section we present a number of algorithms for integrating components with MPR interfaces as well as components with EPR interfaces. The input to the integration problem is a set of component interfaces. A solution to the integration problem is a set of processor allocations such that (i) the sum of all allocations on each processor is less than or equal to one since we use pEDF for scheduling components; (ii) the constraints specified in the component interfaces are met. In the following we explain the integration algorithms corresponding to each interface model in detail.

A. MPR composition

In the following we formally define the integration problem of the FAD approach in which the components are abstracted using the MPR interface model. This problem is similar to that of what is found in the problem of *bin packing with fragmentable items*. In this variation of the bin packing problem, a number of items have to be packed into a set of bins. It is possible to divide items into smaller chunks. The item division does not incur any overhead, i.e., the sizes of the items do not increase by dividing them. The objective is to minimize the number of fragments when placing the items into the bins. In [7] Bertrand *et al.* proved that this variation of the bin packing problem is strongly NP-complete. In our MPR integration problem the items are the MPR interfaces and the processors are the bins. However, our problem is slightly more complex than the above bin packing problem in the following aspects. Instead of minimizing the number of fragments, our objective is to find an allocation in which the number of fragments of all items is less than or equal to their corresponding parallelism level m'_i. In the following we present a mathematical formulation of the constraints of the MPR integration problem:

$$\sum\nolimits_{i=1}^{n} \rho_{i,j} \leq 1 \qquad \forall j \in [1 \ldots m], \tag{1a}$$

$$\sum\nolimits_{j=1}^{m} \rho_{i,j} = U_{\Gamma^{m'_i}_i} \qquad \forall i \in [1 \ldots n], \tag{1b}$$

$$\sum\nolimits_{j=1}^{m} f_{i,j} \leq m'_i \qquad \forall i \in [1 \ldots n], \forall j \in [1 \ldots m], \tag{1c}$$

$$f_{i,j} \in \{0,1\}, \rho_{i,j} \in \mathbb{Z}_{\geq 0}, \tag{1d}$$

Algorithm 1: MPR compact integration.

```
Input: $\Gamma_i$
Output: matrix of processor allocations $\{\rho\}$ or failure.
 1: sortProcessorsIncreasingSlack();
 2: $j$ = findFirstProcessors($m'_i$, $U_{\Gamma_i^{m'_i}}$);
 3: if $j < 0$ then
 4:     return $FALSE$;
 5: end if
 6: $\mathfrak{U}$ = $U_{\Gamma_i^{m'_i}}$;                    ▷ Unallocated utilization
 7: while $\mathfrak{U} > 0$ and $j \leq m$ do
 8:     $\rho_{i,j}$ = $\max(S_j, \mathfrak{U})$;
 9:     $\mathfrak{U}$ -= $\rho_{i,j}$;
10:     $j$++;
11: end while
12: if $\mathfrak{U} = 0$ then
13:     return $\{\rho\}$;
14: else
15:     return $FALSE$;
16: end if
```

Algorithm 2: MPR balanced integration.

```
Input: $\Gamma_i$.
Output: matrix of processor allocations $\{\rho\}$ or failure.
 1: sortProcessorsDecreasingSlack();
 2: $j$ = findFirstProcessors($m'_i$, $U_{\Gamma_i^{m'_i}}$);
 3: if $j < 0$ then
 4:     return $FALSE$;
 5: end if
 6: $S^T = \sum_{i=j}^{j+m'_i} S_i$;
 7: $\mathfrak{U}$ = $U_{\Gamma_i^{m'_i}}$;                    ▷ Unallocated utilization
 8: while $\mathfrak{U} > 0$ and $j \leq m$ do
 9:     $\rho_{i,j} = \max(S_j - S^T, \mathfrak{U})$;
10:     $\mathfrak{U}$ -= $\rho_{i,j}$;
11:     $j$++;
12: end while
13: if $\mathfrak{U} = 0$ then
14:     return $\{\rho\}$;
15: else
16:     return $FALSE$;
17: end if
```

where $\rho_{i,j}$ is the amount of $U_{\Gamma_i^{m'_i}}$ allocated on processor j. $f_{i,j}$ is equal to one when $\rho_{i,j} > 0$, i.e., $\Gamma_i^{m'_i}$ is partially allocated to processor j.

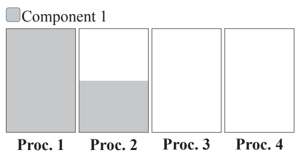
The FAD approach postpones the decomposition to the integration phase. Therefore, we provide two algorithms in which decomposition and allocation is performed simultaneously. In these algorithms each component is treated separately. The component decomposition is performed based on the current status of the slack utilizations on the multiprocessor. We present two algorithms referred as *compact integration* and *balanced integration*. In the compact integration algorithm, the objective at each step is to (i) use a minimum number of the processors (ii) use processors that already have other components assigned on them. The compact integration algorithm is presented in Algorithm. 1. We first sort processors based on increasing slacks. The next step is to find the first m''_i processors which can accommodate the current component, where $m''_i \leq m'_i$. Line 2 returns the index of the first processor in the set that can accommodate the component. Once the processors are sorted it is easy to find m''_i. Function `findFirstProcessors`($m'_i, U_{\Gamma_i^{m'_i}}$) loops through the processors starting from the first processors. At each iteration, the following sum is calculated: $\sum_j^{j+m'_i} S_j$. If the above sum is more than the utilization of the current component being integrated $\mathcal{C}_i$, then `findFirstProcessors` returns the current processor index j. If this function fails to find the candidate set of processors, then it returns -1 and the algorithm returns failure. Otherwise, the algorithm starts filling each processor until either the processor is full or the component is completely allocated. This algorithm is called for all components. The while loop Lines 7 to 11 has at most m iterations. Therefore, since sorting the processors and finding the first processor can be done in polynomial time, the entire algorithm runs in polynomial time. The exact complexity, however, depends on the particular implementations of the sort algorithm.

We present an alternative algorithm for integrating components with MPR interfaces in Algorithm 2. This algorithm is referred as balanced integration. The objective in this approach is to evenly distribute the slack at each step. The algorithm first sorts the processors based on decreasing slack. Thereafter, it finds the first m''_i processors that can fit the components, where $m''_i \leq m'_i$. Once the m''_i target processors are selected, the algorithm calculates the target slack S^T on each processor. Finally, it fills each processor until its target slack is reached. Similar to the compact integration algorithm, the balanced integration algorithm also runs in polynomial time.

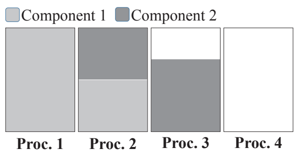
We present an example for elaborating the above two algorithms. Assume that we want to integrate two components with the following interface utilizations: $U_{\Gamma_1^2} = 1.5$ and $U_{\Gamma_2^2} = 1.2$. Assuming that we start with $\mathcal{C}_1$, the compact integration algorithm decomposes the interface into two subcomponents with utilizations equal to one and 0.5. The result of this step is illustrated in Figure 1a. Thereafter, $\mathcal{C}_2$ is integrated. At this stage the `findFirstProcessors` function returns processor 2 because $\mathcal{C}_2$ fits in the slack utilization of processor two and three. $\mathcal{C}_2$ is decomposed into two subcomponents with utilizations equal to 0.5 and 0.7 (Figure 1b). On the other hand, the balanced integration divides $\mathcal{C}_1$ into two identical subcomponents with utilizations equal to 0.75, and it allocates them on the first two processors. The result of this step is illustrated in Figure 1c. When integrating $\mathcal{C}_2$, the `findFirstProcessors` function returns three because the overall slack on processors {1,2} and {2,3} is not enough for integrating $\mathcal{C}_2$. The algorithm, then, divides $\mathcal{C}_2$ into two subcomponents with identical utilizations 0.6, and allocates them on the third and forth processors (Figure 1d). As illustrated in Figure 1, the balanced integration algorithm resulted in fragmented slacks, while the compact integration resulted in one entirely free processor and one partially free processor.

B. EPR integration

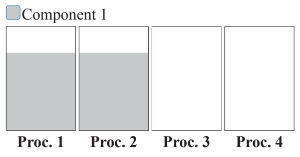
In the following we formally define the integration problem of the FDA approach in which the components are abstracted



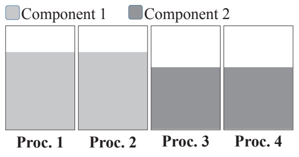
(a) Algorithm 1: after integrating $\mathcal{C}_1$.



(b) Algorithm 1: after integrating $\mathcal{C}_2$.



(c) Algorithm 2: after integrating $\mathcal{C}_1$.



(d) Algorithm 2: after integrating $\mathcal{C}_2$.

Fig. 1: The steps of the two MPR integration algorithms.

Algorithm 3: EPR integration.

```
Input: An EPR interface $\Omega_i$.
Output: matrix of processor allocations $\{\rho\}$ or failure.
 1: sortInterfaces();            ▷ Based on $U_{\Gamma_r^1}$
 2: $\forall r \in [1, n']$ $q_r \leftarrow 1$;
 3: $FLAG \leftarrow FALSE$;
 4: while $FLAG = FALSE$ and isfeasible($\mathcal{Q}$) do
 5:     for $r = 1; r < n'; k$++ do
 6:         $FLAG \leftarrow$ allocate($\Gamma_k^{q_r}$);
 7:         if $FLAG = FALSE$ then
 8:             $\mathcal{Q} \leftarrow$ IncreaseFlexibility($\mathcal{Q}$);
 9:             break;
10:         end if
11:     end for
12: end while
13: return $FLAG$;
```

using the EPR interface model. This problem is analogous to the problem of *bin packing with size-increasing fragmentation*. In this variation of the bin packing problem the items are allowed to be fragmented while fragmenting an item is associated with a cost. Menakerman and Rom [8] showed that this problem is also NP-hard. The EPR integration problem is more complex because instead of a fixed fragmentation cost, the fragmentation cost varies for different subcomponents. For components with parallelism level equal to one, the integration algorithm only has to allocate subcomponents on the multiprocessor. This problem is equivalent to partitioning implicit deadline periodic tasks on multiprocessors. However, if the allocation fails, the integration algorithm can fragment a subcomponent while adding a fragmentation cost. The fragmentation cost for $\Gamma_{i,r}^j$ is denoted using $\Delta_{i,r}^j$. Since we treat each subcomponent separately, solving the EPR integration problem for one component is equivalent to solving this problem for all components in the systems. For notational convenience, we drop the component index when referring to the EPR interfaces in the rest of this section. Let q_r be the parallelism level of subcomponent $\mathcal{C}_r$, and let $\mathcal{Q}$ be the set of parallelism levels $\mathcal{Q} = \{q_1, \ldots, q_p\}$. Also, assume that the total number of subcomponents is represented using n'. The EPR integration problem is to find $\mathcal{Q}$ and allocations such that:

$$\sum\nolimits_{r=1}^{n'} \rho_{r,j} \leq 1 \qquad \forall j \in [1 \ldots m], \tag{2a}$$

$$\sum\nolimits_{j=1}^{m} \rho_{r,j} = U_{\Gamma_r^{q_r}} \qquad \forall r \in [1 \ldots n'], \tag{2b}$$

$$\sum\nolimits_{j=1}^{m} f_{r,j} \leq q_r \qquad \forall r \in [1 \ldots n'], \forall j \in [1 \ldots m], \tag{2c}$$

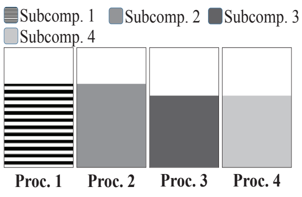
$$f_{r,j} \in \{0, 1\}, \rho_{r,j} \in \mathbb{Z}_{\geq 0}, \tag{2d}$$

The EPR integration algorithm is presented in Algorithm 3. First the subcomponents are sorted based on decreasing first parallelism utilizations ($m'_r = 1$). The algorithm assigns the parallelism levels of all subcomponents to one in Line 2. The `isfeasible(`$\mathcal{Q}$`)` function is called in Line 4. This function performs the following utilization test based on the current parallelism levels, i.e. $\mathcal{Q}$:

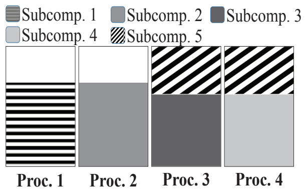
$$\sum\nolimits_{r=1}^{n'} U_{\Gamma_r^{q_r}} \leq m. \tag{3}$$

The algorithm loops through all subcomponents in Line 5. In Line 6, the algorithm calls the `allocate` function. The following two cases may happen: (i) parallelism level equal to one; (ii) parallelism level more than one. In the case of parallelism level equal to one, the allocation is similar to allocating implicit deadline periodic tasks on multiprocessors. We use different versions of the allocation function, each version implementing a different bin packing heuristic. In the evaluations we present the result of using the following three heuristics: FF, BF and WF. In the case of parallelism more than one, however, we use the MPR integration algorithms for allocating the subcomponents on the multiprocessor. If the allocation fails, then the algorithm calls the `IncreaseFlexibility(`$\mathcal{Q}$`)` function. This function selects one subcomponent, and it increments its parallelism level. It selects the subcomponent which has the smallest $\Delta_r^{q_r}$. In other words, it selects a subcomponent that provides one extra level of flexibility with a minimum overhead penalty. Since the `IncreaseFlexibility` function only increases the parallelism levels, in the worst-case the algorithm tries $n' \times m$ different $\mathcal{Q}$. However, in our evaluations, we observed that the `isfeasible` function detects the infeasibility in the early stages and it terminates the algorithm. For each $\mathcal{Q}$, the algorithm calls an allocation heuristic which runs in polynomial time. Thus, the EPR integration algorithm runs in polynomial time.

We present an example to further elaborate the EPR integration algorithm. Suppose we want to integrate five subcomponents with the following utilization for their first parallelism level: $U_{\Gamma_1^1} = U_{\Gamma_2^1} = 0.7$, $U_{\Gamma_3^1} = U_{\Gamma_4^1} = 0.6$ and $U_{\Gamma_5^1} = 0.5$. Also, assume that the second parallelism utilizations are as follows: $U_{\Gamma_1^2} = U_{\Gamma_2^2} = 0.9$, $U_{\Gamma_3^2} = U_{\Gamma_4^2} = 0.85$ and $U_{\Gamma_5^2} = 0.8$. We call Algorithm 3 for all subcomponents, starting from $\mathcal{C}_1$. $\mathcal{C}_1$ to $\mathcal{C}_4$ are allocated to processor one to four respectively. The result of integrating the first four subcomponents is illustrated in Figure 2a. When integrating $\mathcal{C}_5$, the allocation cannot be performed using the first level parallelism. Therefore, the algorithm calls the `IncreaseFlexibility` function, and it increases the parallelism level of $\mathcal{C}_5$ to two. Thereafter, $\mathcal{C}_5$ is divided into two chunks and it is allocated on the third and forth processors (Figure 2b).



(a) Algorithm 3: after integrating $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$ and $\mathcal{C}_4$.



(b) Algorithm 3: after integrating $\mathcal{C}_5$.

Fig. 2: The steps of the EPR integration algorithm.

IV. Evaluations

In this section we present two types of evaluations. In the first set of evaluations our aim is to compare the abstraction overhead of the MPR model against the abstraction overhead of the EPR model. We used the interface calculation method presented in [3] to calculate optimal MPR interfaces. Note that the optimal MPR interfaces have the minimum possible parallelism level. The abstraction overhead of $\mathcal{C}_i$ abstracted using the MPR model is calculated using the following equation:

$$O_i^{\Gamma} = 100 \times \frac{U_{\Gamma_i^{m'_i}} - U_{\mathcal{T}_i}}{U_{\mathcal{T}_i}}, \tag{4}$$

where O_i^{Γ} represents the percentage of abstraction overhead of $\mathcal{C}_i$ abstracted using the MPR model. In addition, for the EPR model, we calculate the abstraction overhead only for parallelism level equal to one because the EPR integration algorithm tries to use the first level parallelisms. We have:

$$O_i^{\Omega} = 100 \times \frac{\sum_{r=1}^{p_i} \Gamma_{i,r}^{1} - U_{\mathcal{T}_i}}{U_{\mathcal{T}_i}}, \tag{5}$$

where O_i^{Ω} represents the percentage of the first parallelism level abstraction overhead of $\mathcal{C}_i$ abstracted using the EPR model. In the second set of simulations we intend to answer the following question: "given a set of components, which combination of the abstraction models and integration techniques requires the lowest number of processors for composing the component set"?

Simulation setup. We generated components with specific task set utilizations. Each task is assigned to a random period between 100 and 200. The utilization of τ_i is selected randomly using a uniform distribution between zero and the maximum allowed task utilization. Except one evaluation in which we varied the maximum allowed task utilization, in the rest of the experiments this parameter was set to 0.9. The execution time of tasks is derived by multiplying the period and the utilization. We assigned deadlines equal to the periods for all evaluations. For generating task sets with a target utilization, we kept generating tasks until the remainder utilization was less than the maximum allowed task utilization 0.9. Then we generated the last task with the remainder utilization. Except one evaluation in which we varied the component periods, we set $\Pi_i = 50$ for components in the rest of the evaluations.

A. Abstraction overhead

We evaluated the influence of increasing task set utilization on the interface overhead. In this experiment, we generated components with task set utilizations from 1.5 to 8 with step size 0.1. For each utilization, we generated 1000 random task sets. Figure 3a shows the result of our evaluation. Note that in this figure the y-axis shows the abstraction overhead. Using the same data, we plotted the relation between the number of tasks and the interface overhead in Figure 3b. These results show that (i) in average O_i^{Ω} is significantly lower the O_i^{Γ}; (ii) O_i^{Γ} increases with respect to the task set utilization. Recall that we use gEDF for the MPR interfaces and we use single processor EDF for the first level EPR interfaces. The reason behind the above result is the following. Firstly, the fixed-job priority algorithms (e.g. gEDF) are not optimal for multiprocessors while EDF is optimal for single processors. Secondly, the analysis used for deriving the MPR interfaces are based on sufficient schedulability tests in global algorithms. For the class of partitioned algorithms, however, the exact schedulability tests are available. Therefore, the first parallelism level EPR interface calculation is based on the exact tests.

In order to evaluate the impact of the interface period on the interface overhead, we performed another experiment. In this experiment we fixed the task set utilization to 1.2, and we generated random tasks as explained above. We varied the component period from 10 to 200 with step size equal to 10. We generated 10000 task sets for each period. The result is illustrated in Figure 3c. This figure suggests that increasing the interface period has a larger impact on the EPR interfaces than on the MPR interfaces. However, even for a very large interface period, the EPR interfaces still incur smaller overhead than the MPR interfaces.

We performed another experiment to evaluate the impact of individual task utilizations on the interface overhead. In other words, we wanted to understand whether or not heavyweight tasks and lightweight tasks have different impact on the interface overhead. We fixed the task set utilization to 2.5, and we varied the maximum task utilization from 0.3 to 0.9. We generated 1000 random components for each maximum task utilization. The results, presented in Figure 3d, show that (i) the MPR interfaces are more sensitive to the task utilizations; (ii) components with heavyweight tasks incur more abstraction overhead.

B. Integration

In this part we generated random systems composed of a number of components. Each component is generated randomly using the method explained above. For each system we had a target task set utilization. The task set utilization of each component $U_{\mathcal{T}_i}$ was selected randomly using a uniform distribution between 1.5 and 3. We kept generating new components until the remaining system utilization was less than 1.5 in which we generated a component with the remaining utilization. Note that by target utilization we refer to the task set utilizations as the interfaces were not derived at the system generation phase. We generated systems with target utilization from 5 to 10. We generated 10000 random systems for each target utilization. Once we generated a system, we calculated the MPR and EPR interfaces. We then ran the integration algorithms presented in the previous section. We ran the compact (CP) and the balanced (BL) algorithms for the MPR integration. Note that in the legends of the figures we use the abbreviation, i.e., CP and BL. For the EPR integration, on the other hand, we examined different combinations of decompositions and integration algorithms. Since we used the FF, BF and WF algorithms, there are nine possible combinations. We denote each combination by combining the decomposition algorithm with the integration algorithm in the legend of the figures. For instance, FFBF means that we used FF for the decomposition and BF for the integration.

In the next evaluation, we studied the performance of the two integration algorithms comparing the number of required processors by each algorithm against the minimum number of processors. We define the ratio of extra required processors by algorithm $\mathcal{A}$ as follows:

$$R^{\mathcal{A}} = 100 \times \frac{\text{\# required processors by } \mathcal{A} - \Psi^{\text{MPR}}}{\Psi^{\text{MPR}}}, \tag{6}$$

where Ψ^{MPR} is the minimum number of processors required for integrating a set of MPR interfaces, and it is calculated using the following equation:

$$\Psi^{\text{MPR}} = \Big\lceil \sum_{i=1}^{n} U_{\Gamma_i^{m'_i}} \Big\rceil.$$

Figure 3e presents R^{BL} and R^{CP}. Each point in the figure is representing the average of 10000 samples. This figure illustrates that both algorithms perform very closely to an optimal algorithm. This shows that the flexibility provided by the MPR interfaces has been exploited well by the two algorithms. Also, the CP algorithm performs better than the BL algorithm.

Let us define a new metric for evaluating the performance of different approaches. We define the ratio of extra required processors by approach $\mathcal{A}$ as follows:

$$R'^{\mathcal{A}} = 100 \times \frac{\text{\# required processors by } \mathcal{A} - \Psi^{\mathcal{T}}}{\Psi^{\mathcal{T}}}, \tag{7}$$

where $\Psi^{\mathcal{T}}$ represents the minimum number of processors required based on the overall task set utilizations, and it is calculated using the following equation:

$$\Psi^{\mathcal{T}} = \Big\lceil \sum_{i=1}^{n} U_{\mathcal{T}_i} \Big\rceil.$$

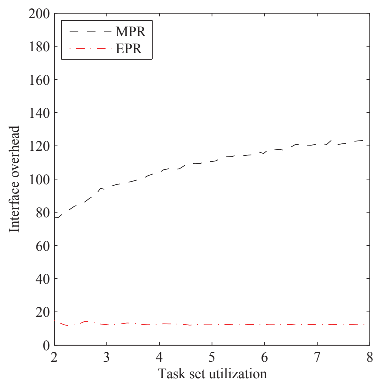
Note that we overloaded symbol $\mathcal{A}$, and it refers to a combination of the abstraction and integration techniques in the FDA approach. Since we have 11 combinations in total, and to keep the figures readable, we present the MPR integration algorithms with three EPR algorithms in one single figure. Figure 3f presents $R'^{\mathcal{A}}$ against the task set utilization for the case where we used FF decomposition. Similarly, Figure 3g and Figure 3h present the cases in which we used BF and WF decompositions respectively. We took the best algorithms of the above three figures and we plotted them in Figure 3i to make the comparison easier. This figure shows that the BFBF combination provided the best result among the studied combination of the algorithms, although it has a very close performance to BFFF. The two algorithms based on WF decomposition considered in this figure performed better than the two algorithms that are based on FF decomposition. The best MPR algorithm (i.e. CP) required 19.26 processors in average when the collective task set utilization was equal to 10. While, the best EPR algorithm combination (i.e. BFBF) required 13.87 processors in average for the same collective task set utilization. In other words, the FDA approach, in average, incurred $53.92\ \%$ less overhead than the FAD approach for this particular target task set utilization.

V. Related work

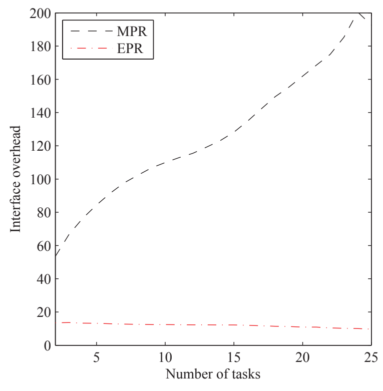
Component-based development approaches have been the subject of several studies in the real-time time scheduling community. The basic idea behind most of these approaches is to abstract the processor requirements of the components, composed of multiple real-time tasks, in an interface. The schedulability of the real-time systems, composed of multiple components, are examined using the component interfaces. These approaches are also referred to as hierarchical scheduling frameworks since the component scheduling and task scheduling are performed in two different levels. For realization of such component based systems, the processors can be time partitioned, while each partition is assigned to a single component. The processor partitions have to be compliant with the requirements specified in the component interfaces. In doing so, the components are isolated from each other with respect to their timing behavior. A timing anomaly in one component will not be propagated to the other components. Several modeling techniques have been proposed for abstracting the processor requirements of the components. In the following we review a subset of such modeling techniques related to our work.

Single processor platforms. The bounded delay abstraction, introduced in [9], specifies the bandwidth along with the maximum blackout time of the processor supply. The maximum blackout time indicates the largest time interval that the processor may be unavailable. The component schedulability test under fixed-priority scheduling and EDF, based on the bounded delay model, is presented in [10]. Shin *et al.* [6] presented another abstraction model for the processor supply of single processors, namely the Periodic Resource (PR) model. The PR model specifies a budget and a replenishment period in its interface. Easwaran *et al.* [11] proposed using a deadline in the component interface to minimize the abstraction overhead. In the case of single processor components abstracted using a periodic model, the component integration problem is equivalent to the task scheduling problem. Therefore, the schedulability analyses previously developed for examining the schedulability of the periodic tasks, can be directly applied to the components assuming that the component budget is equal to the task execution time.

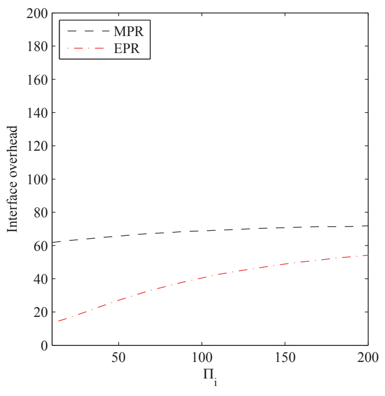
Multiprocessor platforms. With the advent of multiprocessors, it became possible to develop components that require more than one processor for their computations. Therefore, researchers proposed abstraction techniques that can abstract the processor demand of such components. Bini *et al.* presented the Multi Supply Function (MSF) model in [12] for modeling the resource supply of multiprocessor platforms. The Parallel Supply Function (PSF) model [13] is also proposed as an alternative for modeling the resource supply of hierarchical multiprocessor systems. This model indicates a set of supply functions where each of them represent the minimum available supply at a certain parallelism level (from 1 to m). Leontyev and Anderson [14] proposed a model that only specifies bandwidth w in the component interface. In this model $\lfloor w \rfloor$



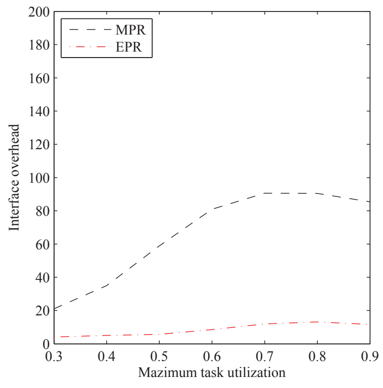
(a) Interface overhead (O_i^{Γ} and O_i^{Ω}) against task set utilization. The step size was set to 0.1.



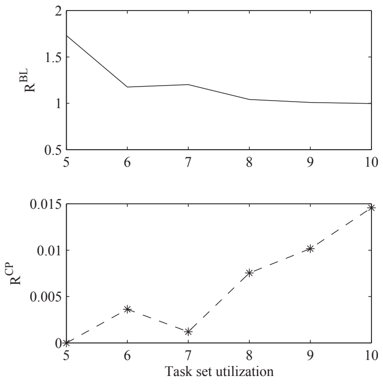
(b) Interface overhead against the number of tasks.



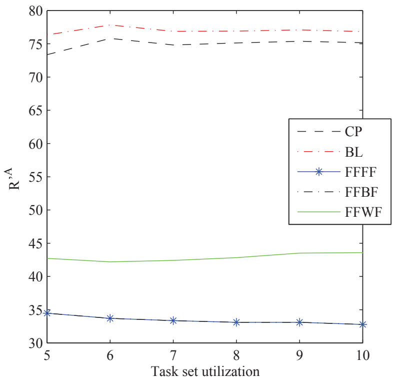
(c) Interface overhead against Π_i. The step size was set to 10.



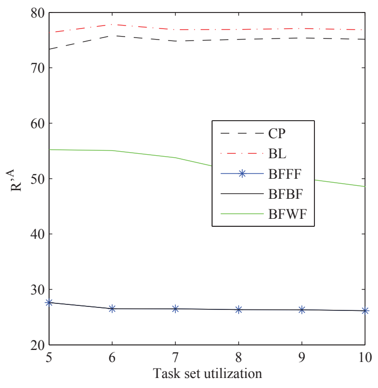
(d) Max task utilization against interface overhead. The step size was set to 0.1.



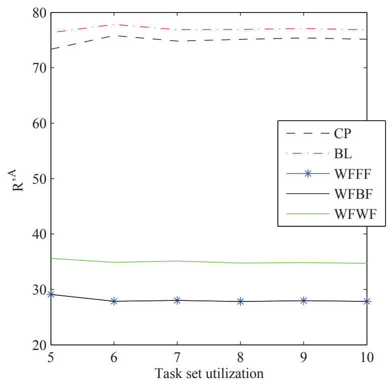
(e) R^{BL} and R^{CL} (Equation 6) versus task set utilization.



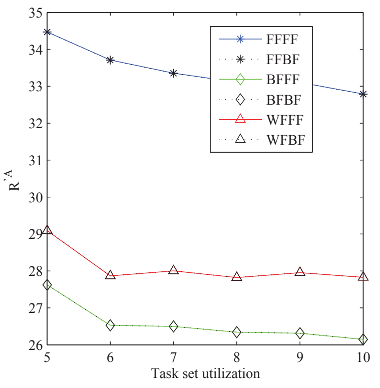
(f) $R'^{\mathcal{A}}$ versus task set utilization using the FF decomposition algorithm.



(g) $R'^{\mathcal{A}}$ versus task set utilization using the BF decomposition algorithm.



(h) $R'^{\mathcal{A}}$ versus task set utilization using the WF decomposition algorithm.



(i) $R'^{\mathcal{A}}$ versus task set utilization for the best six algorithms.

Fig. 3: Evaluation of the interface overheads as well as integration algorithms. In all figures, the y-axis indicates the percentage of imposed overhead.

of a dedicated processor is assigned to the components and the remaining $w - \lfloor w \rfloor$ bandwidth is provided using a periodic server. This model provides limited flexibility at the integration stage for the system integrator as it requires $\lfloor w \rfloor$ dedicated processors. Lipari and Bini proposed the Bounded Delay Multi-partition (BDM) abstraction model in [5]. This model specifies the maximum blackout time and a bandwidth for each parallelism level in its interface. They also provided an algorithm for allocating the interfaces on multiprocessors. In our work, we addressed periodic interface models. In addition, we proposed a new approach in which component decomposition is performed before interface abstraction.

Periodic interface models for multiprocessor platforms. Zhu *et al.* [15] presented an approach in which a Deferrable Server (DS) is attached to each processor. They provided response time analysis for tasks assigned to the DSs which can migrate across the multiprocessor platform. In this work, the authors assumed that there can exist at most one DS per processor. Thus, their approach is not suitable for complex systems composed of several components. Shin *et al.* proposed the MPR model [1]. The MPR model specifies a budget, a replenishment period and a parallelism level in its interface. Easwaran *et al.* [3] proposed an optimal component scheduling algorithm for the MPR interfaces assuming that all components have identical periods. Xu *et al.* proposed the Deterministic MPR (DMPR) model in [16]. This model is different from the MPR model in the following aspect. The DMPR model, similar to [14], allows at most one partial processor allocation. Xi *et al.* [17] have investigated the application of the MPR modeling technique in the Xen virtual machine manager. The authors have also reported some benefits of using partitioned task-scheduling over global task-scheduling. On the other hand, the Generalized MPR (GMPR) model [2] specifies a budget for each parallelism level in the interfaces. This additional information in the interface make it possible to reduce the abstraction overhead.

Our work is different from the aforementioned works in the following aspects. (i) All of the aforementioned approaches perform component decomposition after the abstraction phase, while in this paper we presented an approach for performing the decomposition before the abstraction. Recall that, in order to examine the schedulability of the systems using the task schedulability analyses, the components with utilization more than 100 % of a single processor have to be decomposed to a number of smaller components. (ii) We have quantitatively studied the overhead of using the MPR model considering the whole compositional development processes, i.e., both component abstraction and system integration.

VI. Conclusions and Future Work

In this paper we investigated two alternative approaches for developing real-time software components on multiprocessor platforms. The two approaches vary in the following aspect. The first approach abstracts the component interfaces before decomposing them at the integration phase. The second one, however, first decomposes the components and then abstracts their interfaces. Through extensive simulations, we showed that the second approach utilizes the processor resource significantly better than the first approach. For instance, we showed that given a total task set utilization equal to 10, the second approach in average incurs around 53 % less overhead compared to the first approach.

In the future, we intend to propose an integration algorithm for the GMPR interface model, and we propose to evaluate the GMPR model against the two approaches presented in this paper. We only considered pEDF for scheduling the components in this paper. It is interesting to consider other algorithms including global scheduling algorithms for component-scheduling, and to compare their performances against the partitioned component-scheduling algorithms. Finally, we would like to incorporate resource sharing in our approach and compare the abstraction overhead of our approach with the current state-of-the-art (e.g. [18]).

References

[1] I. Shin, A. Easwaran, and I. Lee, "Hierarchical scheduling framework for virtual clustering of multiprocessors," in *ECRTS'08*, July 2008, pp. 181–190.

[2] A. Burmyakov, E. Bini, and E. Tovar, "Compositional multiprocessor scheduling: the GMPR interface," *Real-Time Systems*, vol. 50, no. 3, pp. 342–376, 2014.

[3] A. Easwaran, I. Shin, and I. Lee, "Optimal virtual cluster-based multiprocessor scheduling," *Real-Time Systems*, vol. 43, no. 1, pp. 25–59, 2009.

[4] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, "A categorization of real-time multiprocessor scheduling problems and algorithms," *Handbook on Scheduling Algorithms, Methods, and Models*, 2004.

[5] G. Lipari and E. Bini, "A framework for hierarchical scheduling on multiprocessors: From application requirements to run-time allocation," in *RTSS'10*, December 2010, pp. 249–258.

[6] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *RTSS'03*, December 2003, pp. 2–13.

[7] B. Lecun, T. Mautor, F. Quessette, and M.-A. Weisser, "Bin packing with fragmentable items: Presentation and approximations," January 2013. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00780434

[8] N. Menakerman and R. Rom, "Bin packing with item fragmentation," *Algorithms and Data Structures*, vol. 2125, pp. 313–324, 2001.

[9] A. Mok, X. Feng, and D. Chen, "Resource partition for real-time systems," in *RTAS'01*, May 2001, pp. 75–84.

[10] I. Shin and I. Lee, "Compositional real-time scheduling framework," in *RTSS'04*, December 2004, pp. 57–67.

[11] A. Easwaran, M. Anand, and I. Lee, "Compositional analysis framework using EDP resource models," in *RTSS'07*, December 2007, pp. 129–138.

[12] E. Bini, G. Buttazzo, and M. Bertogna, "The multi supply function abstraction for multiprocessors," in *RTCSA'09*, August 2009, pp. 294–302.

[13] E. Bini, M. Bertogna, and S. Baruah, "Virtual multiprocessor platforms: Specification and use," in *RTSS'09*, December 2009, pp. 437–446.

[14] H. Leontyev and J. Anderson, "A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees," in *ECRTS'08*, July 2008, pp. 191–200.

[15] H. Zhu, S. Goddard, and M. Dwyer, "Response time analysis of hierarchical scheduling: The synchronized deferrable servers approach," in *RTSS'11*, December 2011, pp. 239–248.

[16] M. Xu, L. T. Phan, I. Lee, O. Sokolsky, S. Xi, C. Lu, and C. Gill, "Cache-aware compositional analysis of real-time multicore virtualization platforms," in *RTSS'13*, December 2013, pp. 1–10.

[17] S. Xi, M. Xu, C. Lu, L. Phan, C. Gill, O. Sokolsky, and I. Lee, "Real-time multi-core virtual machine scheduling in xen," in *EMSOFT'14*, Oct 2014, pp. 1–10.

[18] F. Nemati, M. Behnam, and T. Nolte, "Independently-developed real-time systems on multi-cores with shared resources," in *ECRTS'11*, July 2011.