Technical Report

A Pseudo-Medium-Wide 8-Competitive Interface for Two-Level Compositional Real-Time Scheduling of Constrained-Deadline Sporadic Tasks on a Uniprocessor

Björn Andersson

HURRAY-TR-091103

Version: 0

Date: 11-01-2009

A Pseudo-Medium-Wide 8-Competitive Interface for Two-Level Compositional Real-Time Scheduling of Constrained-Deadline Sporadic Tasks on a Uniprocessor

Björn Andersson

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

http://www.hurray.isep.ipp.pt

Abstract

Compositional real-time scheduling clearly requires that ”normal” real-time scheduling challenges are addressed but challenges intrinsic to compositionality must be addressed as well, in particular: (i) how should interfaces be described? and (ii) how should numerical values be assigned to parameters constituting the interfaces? The real-time systems community has traditionally used narrow interfaces for describing a component (for example, a utilization/bandwidth-like metric and the distribution of this bandwidth in time). In this paper, we introduce the concept of competitive ratio of an interface and show that typical narrow interfaces cause poor performance for scheduling constrained-deadline sporadictasks (competitive ratio is infinite). Therefore, we explore more expressive interfaces; in particular a class called medium-wide interfaces. For this class, we propose an interface type and show how the parameters of the interface should be selected. We also prove that this interface is 8-competitive.

© IPP Hurray! Research Group

A Pseudo-Medium-Wide 8-Competitive Interface for Two-Level Compositional Real-Time Scheduling of Constrained-Deadline Sporadic Tasks on a Uniprocessor

Björn Andersson
CISTER Research unit
Polytechnic Institute of Porto, Portugal
bandersson@dei.isep.ipp.pt

***Abstract*—Compositional real-time scheduling clearly requires that "normal" real-time scheduling challenges are addressed but challenges intrinsic to compositionality must be addressed as well, in particular: (i) how should interfaces be described? and (ii) how should numerical values be assigned to parameters constituting the interfaces? The real-time systems community has traditionally used narrow interfaces for describing a component (for example, a utilization/bandwidth-like metric and the distribution of this bandwidth in time). In this paper, we introduce the concept of competitive ratio of an interface and show that typical narrow interfaces cause poor performance for scheduling constrained-deadline sporadic tasks (competitive ratio is infinite). Therefore, we explore more expressive interfaces; in particular a class called medium-wide interfaces. For this class, we propose an interface type and show how the parameters of the interface should be selected. We also prove that this interface is 8-competitive.**

I. Introduction

Real-time software is increasing in complexity, both because the size (number of lines of code; number of function points/use cases) is increasing but also because the software is developed by many different individuals, potentially in different teams and in different organizations or companies. Compositional theories address this complexity by subdividing software into components. A component can be highly complex but a component is described with an interface, typically of lower complexity. This interface should have the property that a system integrator should be able to compose an entire system out of components by only knowing the interfaces of the components.

Compositional theories are of interest both for (i) ensuring that components deliver correct results and (ii) ensuring that components deliver results at the correct time. This paper deals only with the latter and in order to focus the discussion, we will assume that:

A1. The system has a single processor;
A2. The software is comprised of a set $\mathrm{COMP}=\{\mathrm{COMP}^1, \mathrm{COMP}^2, \ldots, \mathrm{COMP}^K\}$ of K components;
A3. A component COMP^k is comprised of a set $\tau^k=\{\tau_1^k, \tau_2^k, \tau_3^k, \ldots, \tau_{n^k}^k\}$ of n^k constrained-deadline sporadic tasks (note that in this way, we restrict our attention to a 2-level hierarchy);
A4. A constrained-deadline sporadic task τ_i^k is characterized by the parameters T_i^k, C_i^k, D_i^k with the interpretation that τ_i^k releases a (possibly infinite) sequence of jobs with at least T_i^k time units between successive jobs of task τ_i^k and each job of τ_i^k requires C_i^k units of execution to be performed at least D_i^k time units after the release of the job. It is assumed that the release times of jobs cannot be controlled by the scheduling algorithm;
A5. A task which executes for L time units on a processor of speed S completes $L \cdot S$ units of execution;
A6. If the speed of a processor is not explicitly specified, it is assumed that the speed is one;
A7. The parameters T_i^k, C_i^k, D_i^k are integers (we use the assumption of integer parameters only because it simplifies our discussion about the amount of storage needed for task sets and interfaces); arrivals of tasks are allowed to occur at non-integer times and preemptions are allowed to occur at non-integer times as well;
A8. A task needs no other resource than a processor;
A9. A component COMP^k has a static interface $\mathrm{STATIC\_INTERFACE}^k$ and a dynamic interface $\mathrm{DYNAMIC\_INTERFACE}^k$. The static interface is comprised of variables that remain constant over time (for example, pre-specified bandwidth) whereas the dynamic interface is comprised of variables that may change with time (for example, a variable indicating whether there is any task in the component with unfinished execution at current time);
A10. There is a global scheduler $\mathrm{GLOBAL\_SCHED}$ which decides at run-time, at every instant, which component is assigned the processor. In every component COMP^k, there is a local scheduler $\mathrm{LOCAL\_SCHED}^k$;
A11. The global scheduler takes decisions based on both the static and the dynamic interface of all components;
A12. The global scheduler is EDF [1];
A13. The local scheduler of COMP^k executes a task in

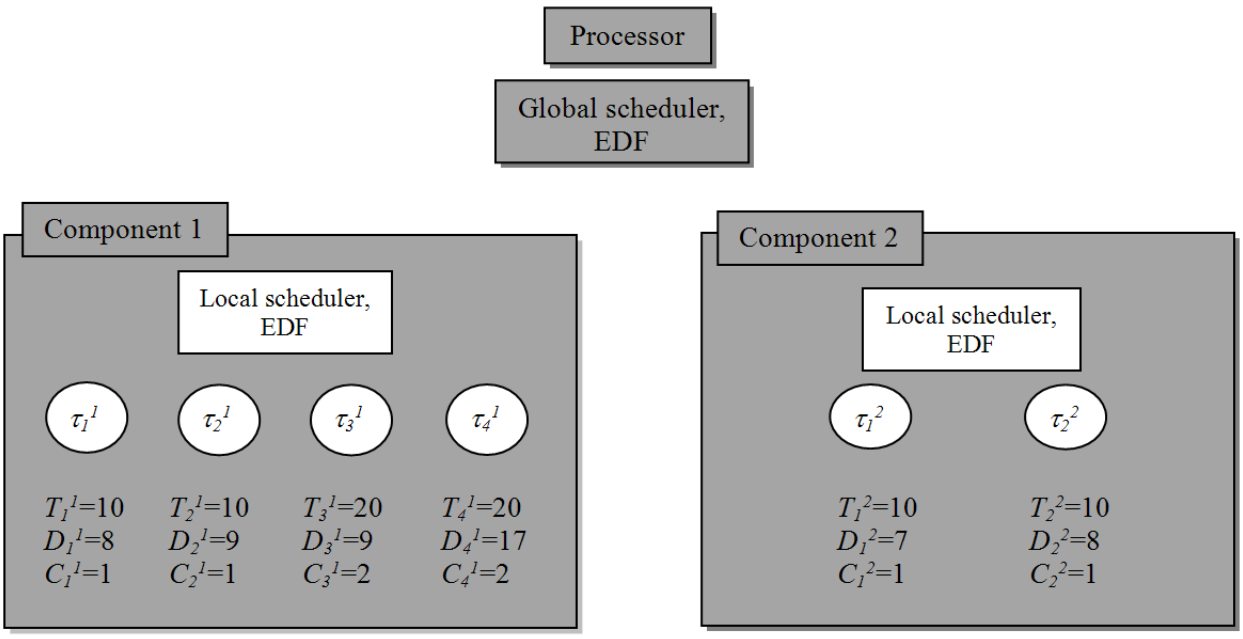

Figure 1. A small example. (The interfaces are not shown).

τ^k at time t if the global scheduler has assigned the processor to $\mathrm{COMP^k}$ at time t;

A14. The local scheduler in a component takes decisions based only on the properties of the tasks in the component;

A15. The local scheduler in each component is EDF [1];

A16. There is a schedulability test for the global scheduler and a schedulability test for each local scheduler; these schedulability tests are used before run-time;

A17. The schedulability test for the global scheduler does not know tasks in each component and it does not know the dynamic interface of components. It only knows the static interface of each component;

A18. The schedulability test for the local scheduler of $\mathrm{COMP^k}$ only knows the tasks in τ^k.

Figure 1 shows an example of such a system.

We address the problem of deciding which parameters should be used to represent the interfaces and how to select parameters for them and how the dynamic interface should be used at run-time. We are interested in doing so and fulfilling the following two (often conflicting) requirements:

R1. The loss in schedulability should be small;

R2. The interface of a component should reveal as little as possible about the tasks in the component.

For the purpose of our discussion, we need to quantify how well an interface fulfills the two requirements above. Therefore, we will define the concepts *competitive ratio* and *"narrowness"*. The former is related to R1 and the latter is related to R2.

We say that an interface generation algorithm A has competitive ratio R if R is the smallest number such that it holds that for every constrained-deadline sporadic task set partitioned into components, that if this task set can be scheduled on a single processor with EDF directly on the processor (that is, without components and without a global scheduler) then this task set can be guaranteed to meet its deadlines as well with interface generation algorithm A provided that the processor is R times as fast. Clearly, a low competitive ratio is desired. R=1 is the best one can get. R=∞ suggests that we pay a high price for compositionality.

In order to characterize the "narrowness" of the interface, we consider the amount of storage needed to describe the interface as compared to the amount of storage needed to describe the tasks in the component. Figure 2 illustrates this.

We say that an interface of a component is:

- narrow if the amount of storage needed by the dynamic and static interface of the component is independent of the amount of storage needed to represent the tasks in the component;
- wide if the amount of storage needed by the dynamic and static interface of the component is proportional to the amount of storage needed to represent the tasks in the component;
- medium-wide if the amount of storage needed by the dynamic and static interface of the component

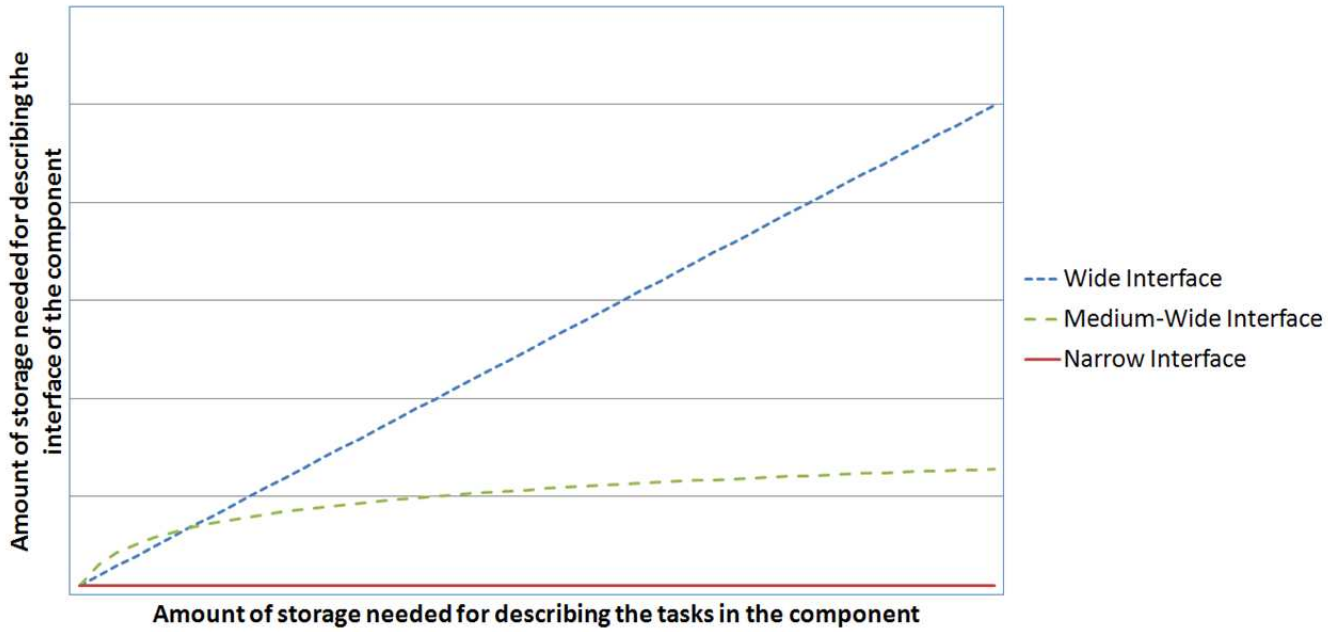

Figure 2. The different "narrowness" of interfaces.

is at most a polynomial of the logarithm of the amount of storage needed to represent the tasks in the component or less.

Within the class of medium-wide, we make a special distinction: We say that an interface is *pseudo-medium-wide* if the interface is medium-wide for the case that T, D, C parameters are upper bounded.

We will say that the system is narrow if all components have narrow interfaces. We will say that the system is medium-wide if all components have narrow or medium-wide interfaces. Otherwise the system is wide. Clearly, narrow interfaces are preferable compared to medium-wide interfaces and medium-wide interfaces are preferred compared to wide interfaces because the former takes greater advantage of compositionality than the latter.

The real-time systems community has addressed the problem of characterizing interfaces and selecting parameters extensively during recently years. A common approach is to represent the interface of a component with two parameters (a bandwidth-like metric and figure of the granularity of the distribution of the bandwidth in time). This approach has the advantage that it is often easy to apply and it is easy for designers to comprehend (it is narrow). Unfortunately, we will see (in Section 2) that this approach can lead to an infinite competitive ratio (R=∞).

We will discuss wide interfaces (in Section 3) and see that this allows us to attain a competitive ratio of one (R=1). But this is undesired as it does not take as much advantage of compositionality for reducing the complexity of systems integration as one does with narrower interfaces.

Therefore, in this paper, we advocate that medium-narrow interfaces should be investigated further. In this spirit, we propose an algorithm for generating pseudo-medium-wide interfaces. We show that this offers a competitive ratio of eight (R=8). Consequently, this offers a reasonable trade-off; it offers the advantages of compositionality in terms of simplifying system design but it does so without sacrificing too much in terms of efficiently managing the resource (the processor).

The remainder of this paper is organized as follows. We initially (in Sections 2-5) ignore policing in order to only focus on compositionality. Section 2 discusses narrow interfaces. Section 3 discusses concepts that we use for interfaces that are not narrow. Section 4 discusses wide interfaces. It presents a new result but it is not particularly useful in its own right; its value lies in providing an understanding of new interface concepts which will be used (in Section 5) when discussing medium-wide interfaces. Section 6 will then discuss how policing can be achieved. Section 7 gives conclusions.

II. Narrow Interfaces

Narrow interfaces have been well-studied. Typically they describe the interface of a component with a (i) capacity metric (bandwidth or pre-specified utilization type of metric) and (ii) a metric specifying how this capacity is distributed

in time. The periodic resource model, also called (Π,Θ) model [2], is one of them and it is frequently used. It describes each component with a server period Π and a server budget Θ. A designer must therefore calculate Π and Θ for a component.

Consider a system comprising three components COMP^1, COMP^2, COMP^3 and each component has a single task. These tasks are characterized[1] as C_1^1=C_1^2=C_1^3=1 and T_1^1=T_1^2=T_1^3=∞ and D_1^1=1;D_1^2=2;D_1^3=3. It is easy to show [3] that this task set can be scheduled to meet deadlines by EDF [1] if components and a global scheduling algorithms were not used. But with the (Π,Θ) model, we have to assign a server period Π and a budget Θ to each component and only Π and Θ are allowed to be used by the global schedulability test. It is easy to see that in order for tasks to be guaranteed to meet deadlines by a local schedulability test, we have to choose:

$$\forall k \in \{1,2,3\} : \frac{\Theta^k}{\Pi^k} \geq \frac{C_1^k}{D_1^k} = \frac{1}{k} \tag{1}$$

Note that the superscripts of the variables in (1) refer to the index of the component. It follows from (1) that:

$$\frac{\Theta^1}{\Pi^1} + \frac{\Theta^2}{\Pi^2} + \frac{\Theta^3}{\Pi^3} \geq 1 + \frac{1}{2} + \frac{1}{3} = \frac{11}{6} \tag{2}$$

And consequently, no global schedulability test can ensure that each component meets its deadlines.

In this particular example, it can be seen that it is necessary to multiply the processor speed by at least $11/6$ in order to meet deadlines with a compositional framework using the (Π,Θ) model. This penalty is caused, not because we made a particularly poor choice in the selection of the parameters assigned to Π and Θ but the penalty is caused by the limitations of the (Π,Θ) model.

We can generalize this reasoning as follows. Let us consider K components and there is one task in each component. The tasks are characterized as C_1^1=C_1^2=...=C_1^K=1 and T_1^1=T_1^2=...=T_1^K=∞ and D_1^1=1;D_1^2=2;D_1^3=3;D_1^4=4;...;D_1^K=K. This task set can be scheduled without components and without a global scheduler. With similar reasoning as in the previous example, we obtain that:

$$\frac{\Theta^1}{\Pi^1} + \frac{\Theta^2}{\Pi^2} + \ldots + \frac{\Theta^K}{\Pi^K} \geq 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{K} \tag{3}$$

It is known that for the summation on the right-hand side of (3), it holds that as K approaches infinity, the summation approaches infinity[2]. Therefore we have that:

[1]This observation is an adaptation of the example in [4] which is based on the original (unpublished) observations by Sanjoy Baruah in 1990s.

[2]The right-hand side of (3) is known as a harmonic series. It is known (from works by Nicole Oresme) that infinite harmonic series are infinite (divergent). For large K, the right-hand side of (3) is approximately $\ln K$.

$$\lim_{k\to\infty} \frac{\Theta^1}{\Pi^1} + \frac{\Theta^2}{\Pi^2} + \ldots + \frac{\Theta^K}{\Pi^K} = \infty \tag{4}$$

and consequently, the competitive ratio is infinite (R=∞).

We have now discussed the (Π,Θ) model and that it gives us an infinite competitive ratio. It should be noted however that one can show, with similar arguments, that other models, bounded-delay model [5], [6] and the EDP model [7], suffer from the same drawback: an infinite competitive ratio.

It is therefore interesting to explore other types of interfaces. The next section (Section 3) introduces some useful concepts. These concepts will be used (in Section 4) which shows a simple idea on how to obtain much better competitive ratio by using wide interfaces. We will adapt this idea in the section thereafter (Section 5). This adaptation gives us lower performance (higher competitive ratio) but this can be achieved with medium-wide interfaces.

III. General Concepts we use for Interfaces that are not Narrow

The concepts of wide interface and medium-wide interface only stipulate how much storage is needed for representing the interface; they do not stipulate exactly what the interfaces look like. Therefore, in this section, we discuss some concepts we will use later in the paper for discussing medium-wide interfaces and wide interfaces.

The main idea is to represent the interface of a component k with a task set $\tau^{k\prime}$. The parameters for this task set are derived from τ^k, the task set inside the component k. Figure 3 shows this.

A. Concepts

About a task, we define the following. We define $\mathrm{active}_{\mathrm{i}}^{\mathrm{k}}(\mathrm{t})$ as being true if task τ_i^k is active at time t; otherwise it is false. We say that a task is active if it is either running or ready. We define $\mathrm{arrival}_{\mathrm{i}}^{\mathrm{k}}(\mathrm{t})$ as the maximum time when task τ_i^k arrives such that this arrival time is no greater than t. We define $\mathrm{abs\_deadline}_{\mathrm{i}}^{\mathrm{k}}(\mathrm{t})$ as $\mathrm{arrival}_{\mathrm{i}}^{\mathrm{k}}(\mathrm{t}) + D_i^k$.

About a component, we define the following. We define $\mathrm{active\_tasks}^{\mathrm{k}}(\mathrm{t})$ as $\{\ \tau_i^k\ :\ (\ \tau_i^k \in \tau^k)\ \wedge\ \mathrm{active}_{\mathrm{i}}^{\mathrm{k}}(\mathrm{t})\ \}$. We also define $\mathrm{exist\_active\_tasks}^{\mathrm{k}}(\mathrm{t})$ as a boolean variable being true if $\mathrm{active\_tasks}^{\mathrm{k}}(\mathrm{t})$ is non-empty; false otherwise. For each component $\mathrm{COMP}^{\mathrm{k}}$, we define a task set τ_k' = $\{\ \tau_1^{k\prime},\ \tau_2^{k\prime},\ \ldots,\ \tau_{n_k\prime}^{k\prime}\ \}$ with $n_{k\prime}$ tasks. These tasks are constrained-deadline sporadic tasks and hence they are described with T,C,D parameters. This primed task set will be used as an interface and the non-primed task set is not used as an interface; they are the real tasks in the component.

We define $\mathrm{intf\_task}(\tau_{\mathrm{i}}^{\mathrm{k}})$ as a function mapping a task τ_i^k $\in \tau^k$ to a task in $\tau^{k\prime}$ ("intf" means "interface"). We define $\mathrm{intf\_abs\_deadline}_{\mathrm{i}}^{\mathrm{k}}(\mathrm{t})$ as $\mathrm{arrival}_{\mathrm{i}}^{\mathrm{k}}(\mathrm{t})$ + $D_q^{k\prime}$, where q is the index of the task $\mathrm{intf\_task}(\tau_{\mathrm{i}}^{\mathrm{k}})$. Intuitively, the function $\mathrm{intf\_task}$ performs a mapping from a task in the component

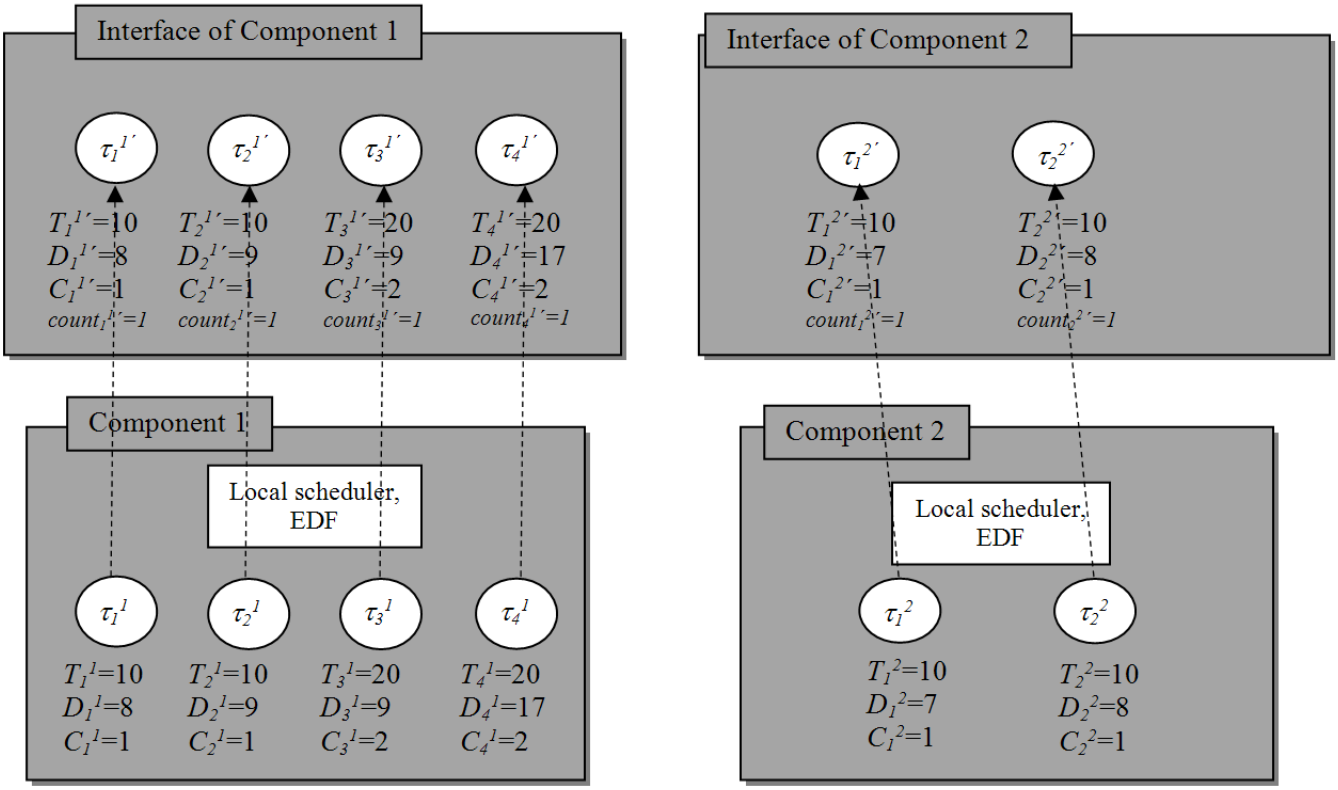

Figure 3. A small example from Figure 1. Interfaces (wide) are shown. Arrows indicate the mapping performed by the function intf_task.

to a task in the interface. Also, intuitively, the function $\text{intf\_abs\_deadline}_\text{i}^\text{k}(\text{t})$ gives us the absolute deadline of task τ_i^k but instead of calculating the absolute deadline of task τ_i^k using the relative deadline of τ_i^k, we use the relative deadline of the task to which τ_i^k maps to in the interface.

For a component, we define $\text{intf\_abs\_deadline}^\text{k}(\text{t})$ as the minimum of all $\text{intf\_task}(\tau_\text{i}^\text{k})$ for all tasks τ_i^k in $\text{active\_tasks}^\text{k}(\text{t})$ if $\text{active\_tasks}^\text{k}(\text{t})$ is non-empty; otherwise $\text{intf\_abs\_deadline}^\text{k}(\text{t})$ is infinite.

B. Static Interface

The static interface of a component COMP^k is variables representing $\tau^k\prime$ = { $\tau_1^k\prime$, $\tau_1^k\prime$, ..., $\tau_{n_k}^k\prime$ }. For each task in $\tau^k\prime$, there is also a counter; that is task $\tau_i^k\prime$ has the counter $\text{counter}_\text{i}^\text{k}\prime$.

C. Dynamic Interface

The dynamic interface of a component COMP^k is variables representing $\text{exist\_active\_tasks}^\text{k}(\text{t})$ and $\text{intf\_abs\_deadline}^\text{k}(\text{t})$.

D. Run-time behavior

At run-time, the global scheduler, assigns the processor to COMP^k at time t if for every other component, COMP^q, it holds that $\text{intf\_abs\_deadline}^\text{k}(\text{t}) \leq \text{intf\_abs\_deadline}^\text{q}(\text{t})$. In each component COMP^k, at time t, the local scheduler is EDF meaning that the task with smallest $\text{intf\_abs\_deadline}_\text{i}^\text{k}(\text{t})$ is selected for execution from $\text{active\_tasks}^\text{k}(\text{t})$. Note that the interface deadline is used by the global scheduler but the interface deadline is not used by the local scheduler.

E. Schedulability Analysis

A schedulability test is performed for the global scheduler. This schedulability test is based on the static interface of all components, that is, $\tau^1\prime$, $\tau^2\prime$, $\tau^3\prime$, ... $\tau^K\prime$. A local schedulability test is not performed; it is not needed.

IV. Wide Interfaces

Recall that for each component COMP^k, we represent $\text{STATIC\_INTERFACE}^\text{k}$ as a task set $\tau^k\prime$ = { $\tau_1^k\prime$, $\tau_2^k\prime$, ..., $\tau_{n_k}^k\prime$ }. In this section, we define this as:

$$\begin{aligned} T_i^k\prime &= T_i^k \\ D_i^k\prime &= D_i^k \\ C_i^k\prime &= C_i^k \\ \text{counter}_\text{i}^\text{k}\prime &= 1 \end{aligned} \tag{5}$$

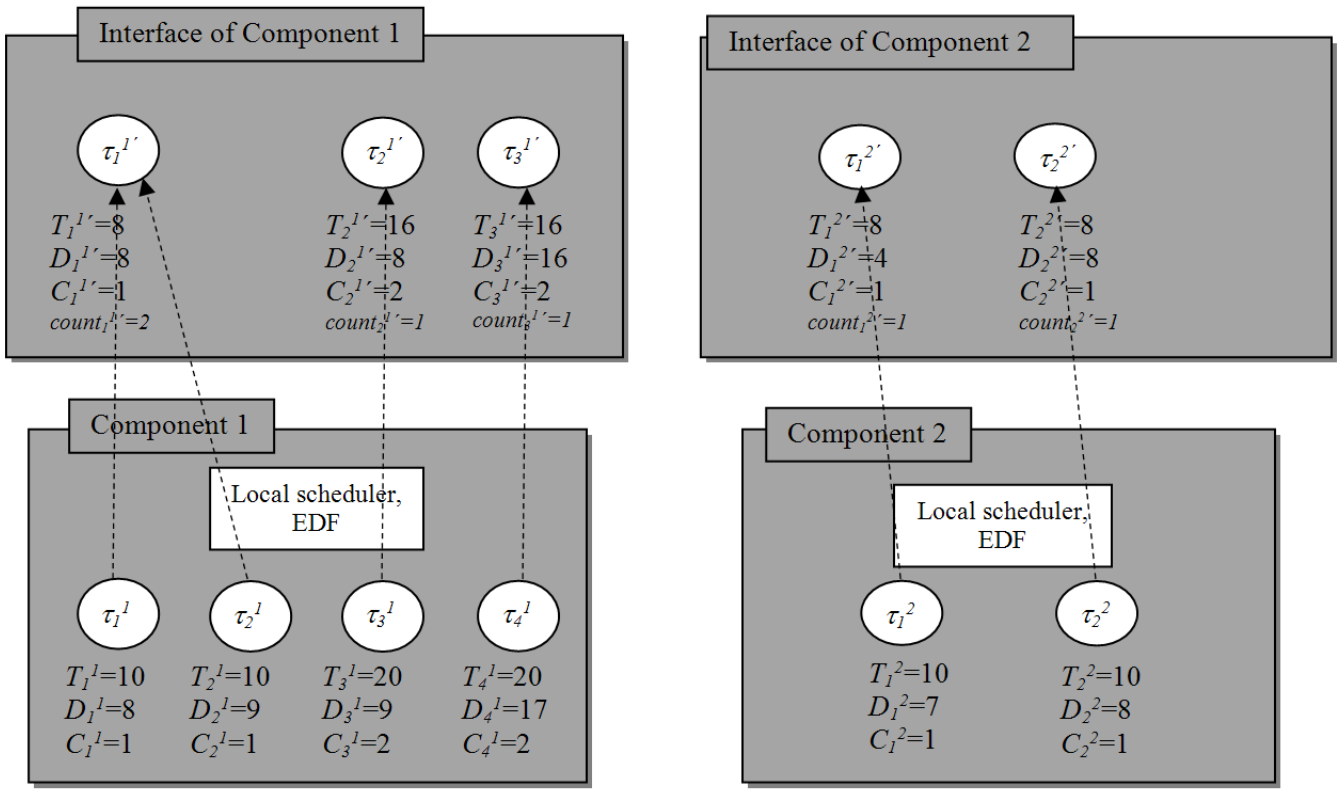

Figure 4. A small example from Figure 1. Interfaces (pseudo-medium-wide) are shown. Arrows indicate the mapping performed by the function intf_task.

that is, the task set in the interface of a component is the same as the tasks in the component. We also define intf_task(τ_i^k) = $\tau_i^{k\prime}$. Figure 3 illustrates this based on the example in Figure 1. It can be seen that this interface mimics the way tasks would be scheduled if all tasks were scheduled directly on the processor (that is no components and no global scheduler). Therefore, this approach has competitive ratio equal to one (R=1). But unfortunately, the interface requires an amount of storage that grows linearly with the number of tasks in the component and therefore it is a wide interface. The next section will show how we can adapt this idea to make it a medium-wide interface.

V. Medium-Wide Interfaces

We can observe that the dynamic interface described in Section 3C is narrow so there is no need to reduce the size of that. We are however interested in keeping a small static interface and this is where the interface stated by (5) is insufficient. Therefore, we are interested in reducing the number of tasks in $\tau^{k\prime}$. We will use three ideas:

I1. If the interface as specified by (5) is used and then the parameters $T_i^{k\prime}$ and $D_i^{k\prime}$ are reduced and $C_i^{k\prime}$ are increased and if the condition of the global schedulability test is true after the change of $T_i^{k\prime}$, $D_i^{k\prime}$, $C_i^{k\prime}$ then we can still be sure that all deadlines of tasks in components are met.

I2. Consider two tasks τ_u^k and τ_v^k in the same component k but the tasks may have different characteristics (T,D,C). If the corresponding interface tasks of these two tasks have the same characteristics then we can let both tasks τ_u^k and τ_v^k be associated to the same interface task. It does not impact the run-time behavior.

I3. If an interface has no task in a component associated to it then this interface task can be removed. It does not impact the run-time behavior.

Figure 4 shows how to use these ideas to create an interface for the example in Figure 1. Figure 5 illustrates Algorithm 1, which shows the algorithm we will use for generating static interfaces. Idea I1 is used on lines 2-3 in Algorithm 1 in Figure 5. Idea I2 is used on lines 4-8. Idea I3 is used on line 9. Note that line 2 which sets the parameters of the task in the interface creates a constrained-deadline sporadic task; that is, it is still ensured that for a task in the interface, it holds that D for this task does not exceed T for this task. Given Algorithm 1, we will now discuss (i) the storage needed by the interface, (ii) whether it guarantees

Algorithm 1.

Output: for each component, a static interface is generated.

1. **for** each component $COMPONENT^k$ **do**
2. create $STATIC\_INTERFACE^k$ as

$$T_i^{k\prime} = 2^{\lfloor \log_2 T_i^k \rfloor}$$
$$D_i^{k\prime} = 2^{\lfloor \log_2 D_i^k \rfloor}$$
$$C_i^{k\prime} = 2^{\lceil \log_2 C_i^k \rceil}$$
$$count_i^{k\prime} = 1$$

3. and set $intf\_task(\tau_i^k) = \tau_i^{k\prime}$.
4. **while** there is a pair of tasks (τ_p^k, τ_q^k) **and** $(\tau_u^{k\prime}, \tau_v^{k\prime})$ such that $(\tau_p^k$ in $\tau^k)$ **and** $(\tau_q^k$ in $\tau^k)$ **and** $(\tau_u^k$ in $\tau^{k\prime})$ **and** $(\tau_v^k$ in $\tau^{k\prime})$ **and** $(intf\_task(\tau_p^k) = \tau_u^{k\prime})$ **and** $(intf\_task(\tau_q^k) = \tau_v^{k\prime})$ **and** $((T_u^{k\prime} = T_v^{k\prime})$ **and** $(D_u^{k\prime} = D_v^{k\prime})$ **and** $(C_u^{k\prime} = C_v^{k\prime}))$ **and** $((count_u^{k\prime} \geq 1)$ **or** $(count_v^{k\prime} \geq 1))$ **do**
5. $count_u^{k\prime} := counter_u^{k\prime} + count_v^{k\prime}$
6. $count_v^{k\prime} := 0$
7. $intf\_task(\tau_q^k) = \tau_u^{k\prime}$
8. **end while**
9. remove all tasks in $\tau^{k\prime}$ for which $count_i^{k\prime} = 0$
10. **end for**

Figure 5. Algorithm for generating pseudo-medium-wide interfaces.

that deadlines and met and (iii) its competitive ratio.

A. Storage

Let TMAX^k be defined as

$$\mathrm{TMAX}^k = \max_{\tau_i^k \in \tau^k} T_i^k$$

DMAX^k and CMAX^k are defined analogously. Let $\mathrm{TMAX}^{k\prime}$ be defined as:

$$\mathrm{TMAX}^{k\prime} = \max_{\tau_i^k \in \tau^k} T_i^{k\prime}$$

$\mathrm{DMAX}^{k\prime}$ and $\mathrm{CMAX}^{k\prime}$ are defined analogously. Then the number of tasks in the static interface of component k is at most:

$$((\log_2 TMAX^{k\prime}) + 1) \cdot ((\log_2 DMAX^{k\prime}) + 1) \cdot ((\log_2 CMAX^{k\prime}) + 1)$$

We know (from line 2 in Algorithm 1) that the the primed and unprimed values of T and D differ by at most a factor of two. We also know that the primed C is larger than the non-primed C. Therefore, we have:

$$((\log_2 TMAX^k) + 2) \cdot ((\log_2 DMAX^k) + 2) \cdot ((\log_2 CMAX^k) + 1) \quad (6)$$

Let n^k denote the number of tasks in component k. Clearly, the count variable of a task in the interface cannot exceed n^k. Also, storing a value which is at most n^k requires at most $((\log_2 n^k)+1)$ bits. Therefore, the amount of storage needed for the interface of component k is at most:

$$((\log_2 n^k) + 1) \cdot ((\log_2 TMAX^k) + 2) \cdot ((\log_2 DMAX^k) + 2) \cdot ((\log_2 CMAX^k) + 1) \quad (7)$$

B. Meeting deadlines

Theorem 1. *Consider a system according to the model as described in Section 1 and the interfaces are as described in Section 3 and where for each component* COMP^k, $\tau^{k\prime}$ *is computed using Algorithm 1. Then, if it holds for all* $L > 0$ *that*

$$\sum_{\forall \mathrm{COMP}^k \in \mathrm{COMP}} \sum_{\tau_j^{k\prime} \in \tau^{k\prime}} \max(\lfloor \frac{L - D_j^{k\prime}}{T_j^{k\prime}} \rfloor, 0) \cdot C_j^{k\prime} \cdot count_j^{k\prime} \leq L$$

then all deadlines of are met.

Proof: Follows from the fact that line 2 in Algorithm 1 associates for each task in the component a task in the interface with higher demand-bound-function [3]. ■

C. Competitive ratio

Lemma 1. *Consider a system according to the model as described in Section 1 and the interfaces are as described in Section 3 and where for each component* $\mathrm{COMP^k}$*,* $\tau^k\prime$ *is computed using Algorithm 1. Then, if it holds for all* $L > 0$ *that*

$$\sum_{\forall \mathrm{COMP^k} \in \mathrm{COMP}} \sum_{\tau_j^k \in \tau^k} \max(\lfloor \frac{L - D_j^k}{T_j^k} \rfloor, 0) \cdot C_j^k \leq L \quad (8)$$

then it holds that for all $L > 0$ *that*

$$\frac{\sum_{\forall \mathrm{COMP^k} \in \mathrm{COMP}} \sum_{\tau_j^k\prime \in \tau^k\prime} \max(\lfloor \frac{L - D_j^k\prime}{T_j^k\prime} \rfloor, 0) \cdot C_j^k\prime \cdot count_j^k\prime}{8} \leq L \quad (9)$$

Proof: Follows from the fact that line 2 in Algorithm 1 causes the demand-bound-function [3] of a task in a the interface of a component to be at most eight times as high as the sum of the tasks that associates to this task in the component. ■

Theorem 2. *Consider a system according to the model as described in Section 1 and the interfaces are as described in Section 3 and where for each component* $\mathrm{COMP^k}$*,* $\tau_k\prime$ *is computed using Algorithm 1. This interface generation has competitive ratio at most eight.*

Proof: Follows from Lemma 1. ■

VI. Policing

Policing can either be performed by the local scheduler inside each component or by the global scheduler.

Policing can be performed inside a component. For example, the run-time scheduling algorithm inside the component can easily check that jobs are not released more frequently than specified by T or that jobs do not execute for longer than as specified by C.

Policing by the global scheduler has the advantage that a systems integrator can allow non-trusted components to be added to a system; if this component would violate its specified behavior then the policing in the global scheduler will ensure that this does not jeopardize the timing requirements of the other components. Policing tasks based on checking their demand-bound function is currently an open question [8]. If this would be solved then it can be used for the solutions presented in this paper.

VII. Conclusion

We discussed how to specify interfaces of components comprising constrained-deadline sporadic tasks. We saw that a narrow interface based on bandwidth offers a competitive ratio infinity. A wide interface can achieve a competitive ratio equal to one but this defeats the purpose of compositional design. An interesting "sweet-spot" is the medium-wide interfaces. We designed a pseudo-medium-wide interface and saw that it is possible to achieve a finite competitive ratio (eight) and still retain the advantages of compositional scheduling theory.

We left the following three important questions open:

- Q1. Can a competitive ratio lower than eight be achieved with pseudo-medium-wide interface?
- Q2. Can a medium-wide interface (that is, not a pseudo-medium-wide interface) be designed with a competitive ratio that is finite?
- Q3. How should policing be performed on the global scheduling level when this approach is used?

Acknowledgment

This work was partially funded by the Portuguese Science and Technology Foundation (Fundação para a Ciência e a Tecnologia - FCT) and the European Commission through grant ArtistDesign ICT-NoE- 214373.

References

[1] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", Journal of the ACM, vol. 20, pp. 46 - 61, 1973.

[2] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees", In Proc.of Proceedings of IEEE Real-Time Systems Symposium, 2003.

[3] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Scheduling Hard-Real-Time Sporadic Tasks on One Processor", In Proc.of IEEE Real-Time Systems Symposium, pp. 182-190, 1990.

[4] B. Andersson, "Synthetic utilization in online aperiodic scheduling", In Proc.of IEEE Real-Time Systems Symposium, Work-in-Progress Session, 2003.

[5] I. Shin and I. Lee, "Compositional Real-Time Scheduling Framework", In Proc.of IEEE Real-Time Systems Symposium, pp. 57-67, 2004.

[6] A. Mok, X. Feng, and D. Chen, "Resource partition for real-time systems", In Proc. of IEEE Real-Time Technology and Applications Symposium, 2001

[7] A. Easwaran, M. Anand, and I. Lee, "Compositional Analysis Framework using EDP Resource Models". In Proc.of IEEE Real-Time Systems Symposium (RTSS 2007), 2007.

[8] S. K. Baruah, "Component-based design of hard-real-time systems on multiprocessor platforms: issues and ideas". In Proc.of 1st Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (Co-located with RTSS), 2008.