# Optimizing the Resource Requirements of Hierarchical Scheduling Systems

Jin Hyun Kim, Axel Legay
INRIA/IRISA, FRANCE
jin-hyun.kim@inria.fr
axel.legay@inria.fr

Louis-Marie Traonouez
INRIA/IRISA, FRANCE
Louis-Marie.Traonouez@inria.fr

Abdeldjalil Boudjadar
Linköping University, SWEDEN
abdeldjalil.boudjadar@liu.se

Ulrik Nyman, Kim G. Larsen
Aalborg University, DENMARK
{ulrik,kgl}@cs.aau.dk

Insup Lee
University of Pennsylvania, USA
lee@cis.upenn.edu

Jin-Young Choi
Korea University, S. KOREA
choi@formal.korea.ac.kr

## ABSTRACT

Compositional reasoning on hierarchical scheduling systems is a well-founded formal method that can construct schedulable and optimal system configurations in a compositional way. However, a compositional framework formulates the resource requirement of a component, called an interface, by assuming that a resource is always supplied by the parent components in the most pessimistic way. For this reason, the component interface demands more resources than the amount of resources that are really sufficient to satisfy subcomponents. We provide two new supply bound functions which provides tighter bounds on the resource requirements of individual components. The tighter bounds are calculated by using more information about the scheduling system.

We evaluate our new tighter bounds by using a model-based schedulability framework for hierarchical scheduling systems realized as Uppaal models. The timed models are checked using model checking tools Uppaal and Uppaal SMC, and we compare our results with the state of the art tool CARTS.

## 1. INTRODUCTION

In order to reduce the system cost, in the design of modern automotive systems, a manufacturer devotes strong efforts to maximize the number of components that can be integrated on a given platform. This can be achieved by minimizing the resource requirements of individual components.

A hierarchical scheduling systems (HSS) integrates a number of components into a single system running on one execution platform. Hierarchical scheduling systems have been gaining more attention by automotive and aircraft manufacturers because they are practical in minimizing the cost and energy of operating applications [12, 7].

A class of analytical methods has been developed to analyze hierarchical scheduling systems [15, 14]. An interface of a component in an HSS is also a resource requirement specification of the component workload and the schedulability of components is checked in a compositional manner.

Due to their rigorous nature, analytical methods are easy to apply once proven correct, but proving the correctness of a new analytical method can be very difficult. However, they also suffer from the abstractness of the models; they do not deal with any detail of the system behavior and thus grossly overestimate the amount resources needed. Such analytical methods consider all potential cases of components interleaving. For this reason, it has not been possible to find a real minimal requirement of resources that is satisfiable for a workload of a given component in a compositional framework.

Model-based methodologies for schedulability analysis [5, 7] allow modeling more detailed and complicated behavior of the component workload (individual tasks), relative to the analytical methods while powerful analysis tools can be applied. Our previous work [3, 4] introduced a model based methodology for the schedulability analysis of hierarchical scheduling systems. Unfortunately, such work relies also on the pessimistic fact of analytical methods and compositional reasoning. The models can be quickly analyzed using statistical methods (Uppaal SMC), which provide guarantees with a selected statistical margin. Once a satisfying model design has been found, the model can be analyzed using symbolic model checking (Uppaal).

This paper uses a methodology for optimizing the resource requirement (interface) of a component, in the context of hierarchical scheduling systems, using model checking techniques. Our methodology consists of using a light weight statistical model-checking method (SMC) and a costly but absolute certain symbolic model-checking method (MC) that operates on identical models.

The current paper extends our previous work [3, 4] by exploiting a new model of HSS that captures all components together in order to make it possible to explore their actual behavior where every possible interleaving of components can be captured explicitly. Basically, we generate the initial component interfaces using a state of the art tool CARTS [14] then apply our new framework to optimize the interfaces. Design space exploration is carried out at low cost using SMC in order to determine more optimal system parameters, that could be impossible to calculate using analytical methods. Afterwards, the candidate component interface proposed by SMC goes under analysis using a model-checker to obtain absolute certainty.

The main contribution of the paper is the introduction of two new supply bound functions which more tightly estimates the resource requirements of components in a hierar-

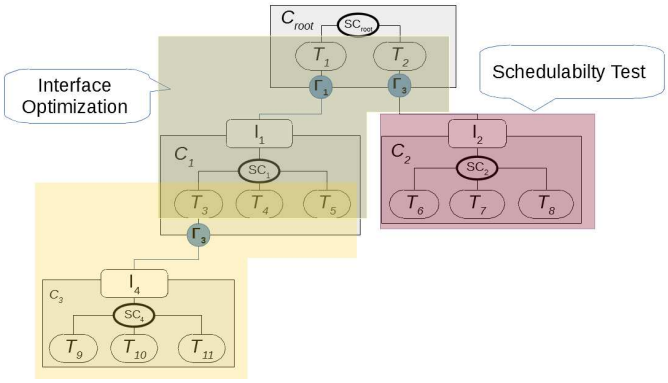



**Figure 1: Optimization and schedulability check**

chical scheduling system.

Moreover, using a model-based methodology also has the advantage that it is possible for system engineers to update the models in order to have a more detailed and realistic analysis of the system, such as overhead. In this way, engineers can utilize specific and detailed knowledge of the system that they are working with; something that cannot be achieved with a classical analytical approach.

The compositional framework of an HSS requires two steps: formulating the component interfaces and checking the schedulability. In the classical methods, both steps are given for a component in a fully compositional way, resulting in pessimistic results. To tighten resource requirements of a component, we utilize information of the upper level components, as shown in Figure 1.

Our technique is not a fully compositional way but a partially compositional way when optimizing individual components because it uses the information of sibling tasks of a task instantiating a resource model, as shown in Figure 1. However, it is fully compositional when checking the schedulability of a component against an interface using our previous work [3, 4]

For the schedulability test, we provides two new supply bound functions extending the classical one[16] (Section 3). For the optimization of an HSS, we use model checking techniques, SMC and MC (Section 4). Our optimization is based on an HSS evaluated by the classical tool of the compositional framework, CARTS[14]. Starting from the button components of an HSS, as shown in Figure 1, each component is re-evaluated by SMC, a small hammer, to optimize resource requirements, exploring behaviors of all necessary components interleaving with the component under analysis without the state-explosion problem. SMC *automatically* searches for the minimum budget of a given component and returns results even though it takes numerous time. However, it does not guarantee 100% certainty of schedulability. For this reason, the resource requirement evaluated by SMC is then verified by MC, a big hammer, to obtain 100% certainty of schedulability.

The rest of the paper is organized as follows: Section 2 presents the necessary background. Section 3 presents our new and tighter supply bound functions. Section 4 evaluates our supply bound functions in the setting of a model-based hierarchical scheduling framework. Section 5 describes the most relevant related work and finally Section 6 concludes the paper.

# 2. PRELIMINARIES

In this section, we present the necessary background of our work, i.e. Stopwatch automata and the theory of hierarchical scheduling systems.

## 2.1 Stopwatch Automata

Stopwatch automata [6] are a subclass of linear hybrid automata. Basically, a stopwatch automaton is a regular timed automaton augmented with stopwatches. A stopwatch is a continuous variable (clock) that can stop and resume without necessarily performing a reset. This is done by assigning to such clocks a progress rate of 0 or 1, i.e. technically the derivative of the stopwatch is assigned to a constant or an expression which evaluates to 0 or 1, so that the stopwatch progresses with the same rate as logical time. The expression assigned to a stopwatch derivative can be a condition on other variables. Such an expression will be evaluated on-the-fly according to the dynamics characterizing each state, every time the condition is evaluated to 1 the stopwatch starts progressing until either the condition is determined to be 0 or encountering a reset.

An important result on the expressiveness of stopwatch automata is that any behavior (accepted language) expressed by a linear hybrid automaton can also be specified using stopwatch automata [6].

Let $X$ be a finite set of continuous variables. A *variable valuation* over $X$ is a mapping $\nu : X \to \mathbb{R}$, where $\mathbb{R}$ is the set of reals. By default, all continuous variables are initialized to 0 (initial value). We write $\mathbb{R}^X$ for the set of valuations over $X$. Valuations over $X$ evolve over time according to *delay functions* $F : \mathbb{R}_{\geq 0} \times \mathbb{R}^X \to \mathbb{R}^X$, where for a delay $d$ and valuation $\nu$, $F(d, \nu)$ provides the new valuation after a delay of $d$. As is the case for delays in timed automata, delay functions are assumed to be time additive in the sense that $F(d_1, F(d_2, \nu)) = F(d_1 + d_2, \nu)$. To allow for communication between different stopwatch automata, we assume a set of actions $\Sigma$, which is partitioned into disjoint sets of input and output actions, i.e. $\Sigma = \Sigma_i \uplus \Sigma_o$.

Definition 1. *A Stopwatch Automaton (SWA) $\mathcal{S}$ is a tuple $\mathcal{S} = (L, \ell_0, X, \Sigma, E, F, I)$, where: (i) $L$ is a finite set of locations, (ii) $\ell_0 \in L$ is the initial location, (iii) $X$ is a finite set of continuous variables, (iv) $\Sigma = \Sigma_i \uplus \Sigma_o$ is a finite set of actions partitioned into inputs ($\Sigma_i$) and outputs ($\Sigma_o$), (v) $E$ is a finite set of edges of the form $(\ell, g, a, \phi, \ell')$, where $\ell$ and $\ell'$ are locations, $g$ is a predicate on $\mathbb{R}^X$, action label $a \in \Sigma$ and $\phi$ is a binary relation (update) on $\mathbb{R}^X$, (vi) for each location $\ell \in L$ $F(\ell)$ is a* delay function*, and (vii) $I$ assigns an invariant predicate $I(\ell)$ to any location $\ell$.*

The semantics of a SWA $\mathcal{S}$ is a timed labeled transition system, whose states are pairs $(\ell, \nu) \in L \times \mathbb{R}^X$ with $\nu \models I(\ell)$, and whose transitions are either delay transitions $(\ell, \nu) \xrightarrow{d} (\ell, \nu')$ with $d \in \mathbb{R}_{\geq 0}$ and $\nu' = F(d, \nu)$, or discrete transitions $(\ell, \nu) \xrightarrow{a} (\ell', \nu')$ if there is an edge $(\ell, g, a, \phi, \ell')$ such that $\nu \models g$ and $\phi(\nu, \nu')$. We write $(\ell, \nu) \rightsquigarrow (\ell', \nu')$ if there is a finite sequence of delay and discrete transitions from $(\ell, \nu)$ to $(\ell', \nu')$.

In the above definition, the clock rate in a location $\ell$ may be either $x' = 1$ or $x' = 0$ (the latter to be annotated explicitly). In order to ensure decidability and efficiency of over-approximation techniques, guards $g$ and invariants $I$ are restricted to conjunctions of simple integer bounds on

individual clocks, and the update predicate are simple assignments of the form $x = e$, where $e$ is an expression only depending on the discrete part of the current state.

Throughout this paper, we consider stopwatch automata with both continuous (clocks) and discrete variables. The initial values of all variables can be user-provided as parameters, and not necessarily be the default values.

## 2.2 Compositional Framework for HSSs

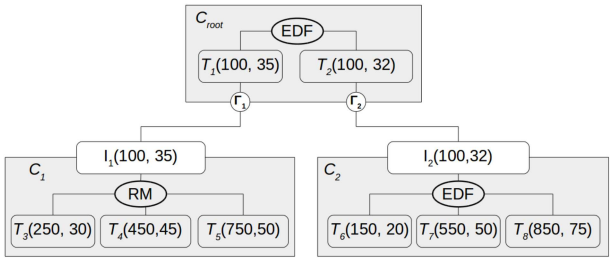



**Figure 2: A hierarchical scheduling system**

Formally, a hierarchical scheduling system (HSS) $S$ is defined as a set of *scheduling components (units)*. A scheduling unit $C(W, A)$ is composed of a *workload* $W$ a set of tasks, and a *scheduling algorithm* $A$. A task $T_i(p, e, d)$ is characterized by inter-arrival time ($p_i$), execution time ($e_i$), and deadline ($d_i$). An inter-arrival time can be the same as the deadline, which is called implicit deadline. This paper uses implicit deadlines as long as deadlines are not explicitly specified. For a given component, its collective resource requirement is represented by an interface ($I_j$). A task that a sub-component relies on is called a resource model $\Gamma_j$ ($Task_1$ and $Task_2$ in Figure 2), such that the component can execute only when the task executes.

Figure 2 shows an HSS, where components are organized in a parent-child relationship making a hierarchy. Individual components consist of one or more tasks. A child component requires certain amount of resources from its parent component via its interface, which is a collective resource requirement of the child component. A parent component executes, regardless of child components connected to its tasks, i.e. it does not synchronize with its child components. In other words, a parent component can execute even though no child one executes, and vice versa.

The schedulability test for an HSS can be performed in a compositional way. A child component requires its parent component to provide a specific amount of resources through an interface, and the schedulability of individual components, except the root component, is checked by verifying whether their interfaces are satisfied by the corresponding resource models. The root component is checked by the classical schedulability test methods.

Note that the classical compositional frameworks for HSSs provide a way of decomposing the schedulability analysis. However, that costs some over-approximation because resource models are designed to consider the worst-case resource supply assuming all preemptions.

In our previous work [3, 4], we provided a model-based compositional framework that relies on a formal model of HSSs. So that each scheduling unit can be individually investigated w.r.t. a resource model that instantiates a resource-supplying task respecting an interface. Thus, our previous work do not return more optional resource requirements of a component while a model of workloads is flexible enough to encompass any settings of HSSs.

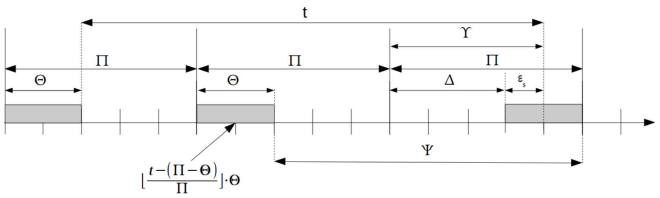



**Figure 3: Resource supply pattern**

In the classical frameworks, a resource model is represented by a supply bound function (`sbf`) and resource requirements of a child component is estimated by a demand bound function (`dbf`). An interface $I$ of a component satisfiable for the workload of that component is computed by the relation between $\texttt{dbf}_A(W, t)$ and $\texttt{sbf}_\Gamma(t)$ depending on the resource model $\Gamma$, where $\texttt{dbf}_A(W, t)$ returns an amount of resources that are demanded by a workload for a time interval $t$ and $\texttt{sbf}_\Gamma(t)$ returns an amount of resources supplied by the resource model $\Gamma$ for a time interval $t$.

The Periodic Resource Model (PRM) proposed by Shin et al., in [16, 15], assigns a particular amount of resources every specific period. An interface observing the PRM consists of two attributes, a period and a budget, and is computed by searching for a resource model ($\Gamma$) that satisfies the following schedulability condition:

$$\forall 0 < t \leq 2 \cdot LCM_p \quad \texttt{dbf}_A(W, t) \leq \texttt{sbf}_\Gamma(t) \tag{1}$$

where $t$ is a time interval, and $LCM_p$ is the least common multiplier of periods of all the tasks in $W$.

The supply bound function $\texttt{sbf}_{PRM}(t)$ of a PRM is defined by:

$$\texttt{sbf}_{PRM}(t) = \left\lfloor \frac{t - (\Pi - \Theta)}{\Pi} \right\rfloor \cdot \Theta + \epsilon_s \tag{2}$$

$$\epsilon_s = \max\left(t - 2(\Pi - \Theta) - \Pi \left\lfloor \frac{t - (\Pi - \Theta)}{\Pi} \right\rfloor, 0\right) \tag{3}$$

For the conservative analysis, PRM assumes that the maximum service time (MST, $\Psi$ in Figure 3) between two completions of the resource supplies is $2(\Pi - \Theta) + \Theta = 2\Pi - \Theta$, and the MST of a component in an HSS varies according to the preemption caused by other components. This paper finds a more realistic MST to optimize HSSs.

For the scheduling policies EDF (Earliest Deadline First) and RM (Rate Monotonic), the demand bound functions are respectively defined by:

$$\texttt{dbf}_{EDF}(W, t) = \sum_{T_i \in W} \left\lfloor \frac{t}{p_i} \right\rfloor \cdot e_i \tag{4}$$

$$\texttt{dbf}_{RM}(W, t, i) = e_i + \sum_{T_k \in HP_W(i)} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k \tag{5}$$

where $HP(i)$ is a set of tasks whose priorities are higher than that of $T_i$.

# 3. TIGHTER SUPPLY BOUND FUNCTIONS

This section presents a metrics that can be used to determine the capability of a resource model's supply. In addition, for the schedulability check, we present two new supply bound functions that can gain more resource assignments from a resource model.

## 3.1 Maximum Service Time

Let $\Theta_i$ and $\Pi_i$ be the execution time and period of a resource-supplying task $T_i$. In case of the classical PRM (periodic resource models), the maximum service time (MST) is always $2(\Pi_i - \Theta_i) + \Theta_i = 2\Pi_i - \Theta_i$ regardless of scheduling policy.

For a given resource model, the MST denoted by $\Psi$ in Figure 3 is the maximum interval between two completions of the resource supply. In fact, the MST of a resource model depends on its realizing (server) task, whose the execution can be intervened by the sibling tasks of the server task, in the same component, and the resource model on which the component relies.

For instance, the resource model $\Gamma_3$ in Figure 1 is associated with the task $T_3$, the MST of $\Gamma_3$ thus depends on $T_3$. The execution of $T_3$ can be preempted by $T_4$ and/or $T_5$ and stops/resumes by the resource model $\Gamma_1$. For a given PRM $\Gamma(\Pi, \Theta)$, the MST is computed by:

$$MST_\Gamma = (\Pi - \Theta) + \lambda + \sum_{T_k \in HP(i)} \left\lceil \frac{\Pi}{p_k} \right\rceil \cdot e_k + \Theta \quad (6)$$

$$= \Pi + \lambda + \sum_{T_k \in HP(i)} \left\lceil \frac{\Pi}{p_k} \right\rceil \cdot e_k \quad (7)$$

where $\lambda$ is a delay that is caused by the resource model that the component of $T_i$ relies on, and $HP(i)$ is the set of tasks whose priorities are higher than the priority of task $T_i$, and $T_i$ is the task to which $\Gamma$ is associated.

Note that $\lambda$ depends on the component that the resource model relies on, thus it is necessary to require more information from the upper level component to tighten the MST.

Put simply, if we can find a more tighter MST of a resource model by exploiting the information of the task instance of the resource model, it is possible to tighten the resource requirements of a resource-supplied component while keep it schedulable.

## 3.2 Supply Bound Function for RM

Based on the above observation of the MST, we refine the supply bound function of [16] with the information on the component encapsulating the task that instantiates a resource model: sibling tasks of the task, a scheduling algorithm and a resource model of the component.

For a given PRM $\Gamma(\Pi, \Theta)$, let $T_i$ be a task of the component $C$ instantiating $\Gamma$ with inheriting the same parameters of $T_i$.

For the RM scheduling algorithm, the *supply bound function* ($\mathsf{sbf}$) of the resource model depends on $T_i$ that is included by the workload $W$ of $C$ employing the RM scheduling algorithm. The $\mathsf{sbf}$ is defined by:

$$\mathsf{sbf}_{\Gamma(T_i,W,RM)}(t) = \left\lfloor \frac{t - (\Pi_i - \Theta_i)}{\Pi_i} \right\rfloor \cdot \Theta_i + \epsilon_s \quad (8)$$

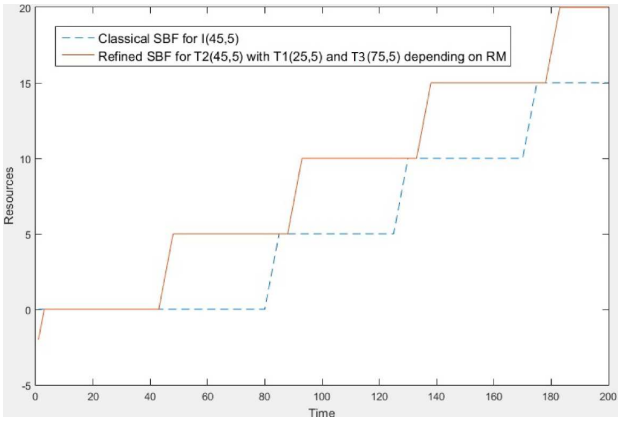



**Figure 4: Comparison of supply bound functions**

where

$$\epsilon_s = max(min(\Phi, \Theta_i), 0) \quad (9)$$

$$\Phi = \begin{cases} \Upsilon - \Delta, & \text{if } C \text{ is the root} \\ \mathsf{sbf}_{\Gamma HIGH}(\Upsilon) & \\ \quad - \sum_{T_k \in HP(i)} \lceil \frac{\Upsilon}{\Pi_k} \rceil \cdot \Theta_k, & \text{otherwise} \end{cases} \quad (10)$$

$$\Upsilon = t - (\Pi_i - \Theta_i) - \Pi_i \left\lfloor \frac{t - (\Pi_i - \Theta_i)}{\Pi_i} \right\rfloor \quad (11)$$

$$\Delta = \sum_{T_k \in HP(i)} \left\lceil \frac{\Upsilon}{\Pi_k} \right\rceil \cdot \Theta_k \quad (12)$$

where $HP(i)$ is the set of tasks with higher priorities over that ($pri_i$) of $T_i$. $\mathsf{sbf}_{\Gamma HIGH}$ is a resource supply that is provided by the component that $C$ depends on, and $t$ is a time duration. $\Upsilon$ of Eq. 11 is the remaining time of $t$ that is less than the period $\Pi$, as shown in Figure 3. For $\Upsilon$ time units, the amount of resources supplied by this resource model can be in one of two cases: In the first case where the resource model is running in the root component, the amount of resources provided by the resource model is the deduction of $\Delta$ from $\Upsilon$, where $\Delta$ of Eq.12 is, as shown in Figure 3, the preemption time of $T_i$ by $T_k$ whose priorities are higher than $T_i$'s. In the second case where the resource model is not running in the root component, the amount of resources is computed by Eq. 10 that is the deduction of resources preempted by higher priority tasks from resources given by its parent task for the time $\Upsilon$.

If a task instantiating a resource model is located in the root component, $\epsilon_s$ depends only on the sibling tasks of $T_i$. Otherwise, $\epsilon_s$ depends on both the sibling tasks and the resource model that the child component relies on.

EXAMPLE 1. *For the root resource-supplying component* $C = T_1(25, 3), T_2(45, 5), T_4(75, 5)$, *Figure 4 compares the classical and refined supply bound functions of task* $T_2(45, 5)$.

## 3.3 Supply Bound Function for EDF

Our supply bound function for EDF policy over-approximates the delay caused by the sibling tasks of the task instantiating a resource model because of the non-determinism of task priorities according to the EDF.

The maximum amount of resources demanded by tasks scheduled according to EDF can be computed by the demand bound function proposed in [1].

For a given time duration $t$ and a workload $W$, the preemption time of a task $T_i$ by its sibling task $T_k$ according to the EDF can be computed by the *demand bound function* of [1] excluding $T_i$ from $W$ as follows:

$$\mathsf{dbf}_{(EDF,T_i,W)}(t) = \sum_{T_K \in W} \left\lfloor \frac{\Upsilon}{\Pi_k} \right\rfloor \cdot \Theta_k \quad (13)$$

, where $T_i, T_k \in W, i \neq k$

For a given time duration $t$, the sbf depending on $T_i(\Pi_i, \Theta_i)$ scheduled by EDF is:

$$\mathsf{sbf}_{\Gamma(T_i,W,EDF)}(t) = \left\lfloor \frac{t-(\Pi_i-\Theta_i)}{\Pi_i} \right\rfloor \cdot \Theta_i + \epsilon_s \quad (14)$$

where

$$\epsilon_s = max(min(\Phi, \Theta_i), 0) \quad (15)$$

$$\Phi = \begin{cases} \Upsilon - \Delta, & \text{if } C \text{ is the root} \\ \mathsf{sbf}_{\Gamma^{HIGH}}(\Upsilon) & \\ \quad - \sum_{T_K \in W} \left\lfloor \frac{\Upsilon}{\Pi_k} \right\rfloor \cdot \Theta_k & \text{otherwise} \end{cases} \quad (16)$$

$$\Upsilon = t - (\Pi_i - \Theta_i) - \Pi_i \left\lfloor \frac{t-(\Pi_i-\Theta_i)}{\Pi_i} \right\rfloor \quad (17)$$

$$\Delta = \sum_{T_K \in W} \left\lfloor \frac{\Upsilon}{\Pi_k} \right\rfloor \cdot \Theta_k \quad (18)$$

and $T_i, T_k \in W, i \neq k$

Note that the difference between the sbf for EDF and the one for RM is that the preemption time by sibling tasks is computed using the dbf in Eq. 13.

# 4. MODEL-BASED EVALUATION

Although the new supply bound functions presented in the previous section can be used on their own, we in this section intend to evaluate them in the setting of a model-based hierarchical scheduling framework.

We introduce a novel model of an HSS in order to optimize component resource requirements. Indeed, our model of an HSS verifies even more tight resource requirements of a component. It is because our model of an HSS simulates the actual HSS with all necessary information of a resource-supplying component that influences the execution of a component.

## 4.1 Stopwatch Automata Model of HSSs

Our model of HSSs consists of one task template (model) and two scheduling mechanism templates. A process of the task model behaves as a periodic task if it belongs to a leaf component of an HSS. A process representing a task, within a parent component, controls a component that would be served by that task.

Figure 5 shows the SWA template of the task tid. It has five locations where the time can progress: DlyIOffset, Ready, Executing, JobDone, and MissingDeadline. When the initial job is released, there is a delay at location DlyIOffset for a time of the constant tstat[tid].ioffset. Afterwards, the task moves to the location Ready and asks for a CPU resource by sending a request event req_sched[tstat[tid].cid] to the relevant scheduler. If the task is allowed to use a CPU i.e. the function isSched() returns 1, it joins the location Executing. At that location the task can execute, i.e. the stopwatch clock t_et[tid] can progress, only when that a CPU resource is available to it. The execution of task tid is denoted by the progress of the stopwatch clock t_et[tid] bounded by BCET and WCET.The availability of a CPU resource is denoted by the function isSchedSuped() that is manipulated by the associated scheduler.

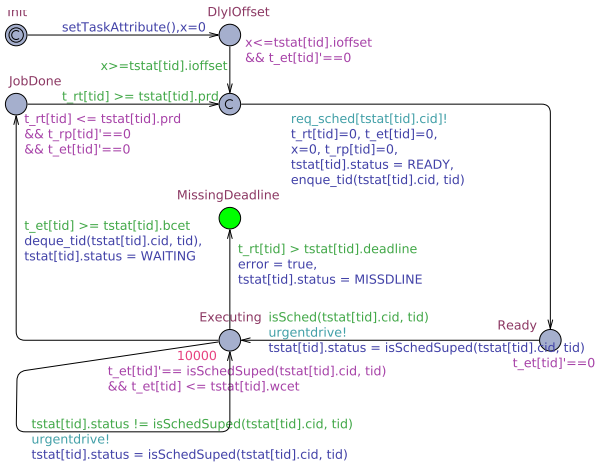



**Figure 5: A task model in SWA**

In short, the clock t_et[tid] grows only when the task tid is scheduled to use a CPU resource and isSchedSuped() returns 1. If the BCET is satisfied, the task can leave the location Executing, but it cannot stay longer than the WCET. When the task leaves that location, it releases the CPU resource using the function deque_tid(). When the task joins location JobDone, then it finishes one period. When the deadline is reached at location Executing, the task joins the location MissingDeadline with reporting its missing deadline by setting the global variable error to true.

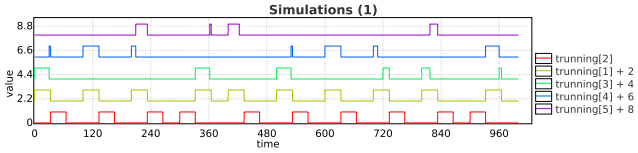


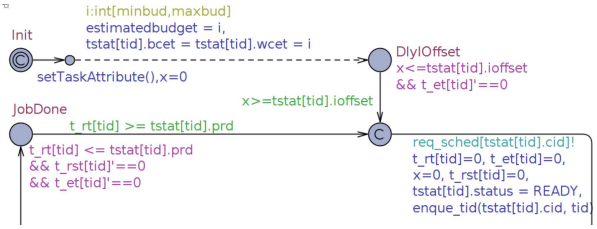

**Figure 6: Simulation of the running example**

Figure 6 shows a simulation result of the running example of Figure 2. The first and second graphs from the bottom show the executions of $T_2$ and $T_1$, respectively. The third to the fifth show the executions of $T_3$, $T_4$ and $T_5$ respectively. Notice that the executions of $T_2$ and $T_1$ interleave with each other, but the executions of $T_3$, $T_4$ and $T_5$ are clearly dependent on the execution of $T_1$, i.e. their running is only permitted when $T_1$ is scheduled.

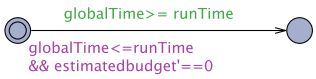
## 4.2 SWA Model for Interface Optimization

To find out the minimum execution time of a parent task that enables a child component to be schedulable, we provide two extra templates (Figure 7).

The task model of Figure 7(a) is an extension of the original task model of Figure 5 and shows only the difference with the original task model of Figure 5. It adds to the original one a stochastic transition between the locations Init and DlyOffset that varies execution times of a parent task while checking if its child tasks miss deadlines. Compared to the task model of Figure 5, the task model of Figure 7 picks up an execution time (on the dotted edge) when the task is instantiated as a process. SMC repeats the simulation of the model to generate traces and finds out a bound of the execution time that leads child tasks to miss deadlines. The model




(a) A task model with a random execution time




(b) Simulation Finalizer

**Figure 7: Requirement estimating processes**

of Figure 7(b) collaborates with the following SMC query to generate a probability distribution that shows a time bound where the schedulability of child tasks is violated.

$$Pr[estimatedBudget <= runTime] \\ (<> globalTime >= runTime \ and \ error)$$

where a clock variable estimatedBudget denotes a sample of the execution time for each trace. The condition variable error indicates that the schedulability is violated.

Given the above query and SWA models of Figure 7, SMC generates a probability distribution in terms of the variable estimatedBudget, showing probabilities of violating the schedulability for a given sample value of estimatedBudget.

According to the probability distribution, the least execution time (of a parent task) that is greater than the upper bound of the execution time disproving the schedulability is selected as the minimum execution time of the parent task that enables child tasks to be schedulable. The minimum execution time would then be analyzed using model checking (MC) to see whether this can be proven to be a sufficient budget.

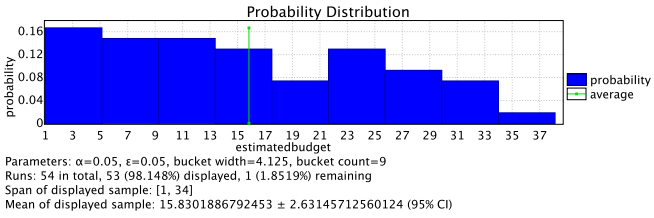



**Figure 8: A probability distribution for the execution time of $T_1$ of Figure 2**

Figure 8 shows the probability distribution for the execution times of task $T_1$ (Figure 2) that cause the child tasks of $C_1$ to miss their deadlines. It shows that the values between [0,34] of the span of the displayed sample lead tasks (at least one task) to miss deadlines, we thus conclude that 35 is the minimum budget that can satisfy child tasks. All runs that do not miss a deadline are not included in the plot and thus only 82.305% of the runs are included in the plot.

## 4.3 Optimization of HSSs

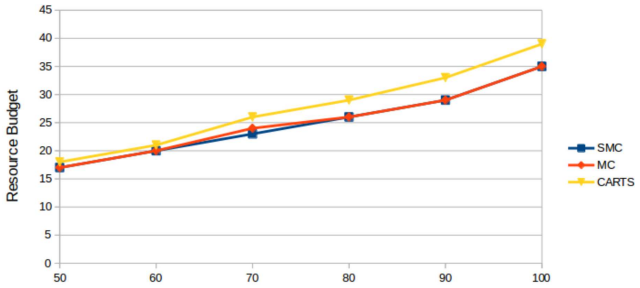



**Figure 9: Sufficient resource requirements of interface $I_1$ (in Figure 2) for different periods.**

Our optimization is conducted in a compositional manner, starting from the interfaces of components as computed by the CARTS tool. As shown in Figure 1, the optimization starts from the bottom components by exploiting the information of a child component and its parent component.

Figure 9 shows the estimated results of interface $I_1$ (Figure 2) obtained by CARTS, SMC and MC when varying the period.

**Table 1: Comparison of resource requirements**

| | Interface ID. Sched. Alg | Interface ID. Sched. Alg | $C_{root}$ Sched. Alg |
|---|---|---|---|
| Configuration 1a (CARTS) | $I_1$(100,39) (RM) | $I_2$(100,35) (EDF) | (EDF) |
| Configuration 1b (MC) | $I_1$(100,35) (RM) | $I_2$(100,32) (EDF) | (EDF) |
| Configuration 2a (CARTS) | $I_1$(100,39) (RM) | $I_2$(100,35) (EDF) | (RM) |
| Configuration 2b (MC) | $I_1$(100,33) (RM) | $I_2$(100,32) (EDF) | (RM) |
| Configuration 3a (CARTS) | $I_1$(100,31) (EDF) | $I_2$(100,35) (EDF) | (RM) |
| Configuration 3b (MC) | $I_1$(100,30) (EDF) | $I_2$(100,32) (EDF) | (RM) |

Table 1 exhibits the optimized interfaces of Figure 2 that are proven by MC and the comparison with the outcomes of CARTS. Each pair, e.g. (1a, 1b), uses the same scheduling algorithms for different components. The choice of scheduling algorithms for each configuration pair is arbitrary.

*Observation 1.*

Using EDF in child components always requires less resources than using RM, which is well known. However, as shown in Configurations 1b and 2b in Table 1, the resource requirements of child components can be tightened by varying the scheduling algorithm of the root component $C_{root}$. In other words, using RM in the parent component can make the child components requiring less resources compared to the case when EDF is used in the parent component.

In the case where $C_{root}$ uses RM and $T_1$ has higher priority than $T_2$, the MST of the resource model instantiated by $T_1$ is 100, and the MST served by $T_2$ is also 100. When the root component $C_{root}$ uses EDF, $T_1$ and $T_2$ have the same priorities all the time because they have the same implicit deadlines. Therefore, the MST instantiated by $T_1$ is 133, and the MST served by $T_2$ is 135. Thus, the MST of EDF varies more than that of RM.

**Table 2: The analysis times of SMC and MC**

| Period | 50 | 60 | 70 | 80 | 90 | 100 |
|---|---|---|---|---|---|---|
| SMC(sec) | 20.04 | 33.78 | 32.87 | 29.98 | 25.87 | 5.15 |
| MC (sec) | 228.77 | 184.87 | 503.51 | 145.27 | 129.13 | 350.7 |

*Observation 2.*

To check how much resources the explicit deadline can save [8], we varied the deadline of task $T_1$ and found the necessary budgets satisfying the interface $I_1$ as shown int Table 3 and 4.

**Table 3: The satisfying budgets of $I_1$ varied by the deadline of $T_1$ scheduled using EDF**

| Periods | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
|---|---|---|---|---|---|---|---|---|
| Deadlines | 100 | 99 | 98 | 95 | 90 | 50 | 35 | 33 |
| Budgets | 35 | 33 | 33 | 33 | 33 | 33 | 33 | 33 |

**Table 4: The satisfying budgets of $I_1$ varied by the deadline of $T_1$ scheduled using RM**

| Periods | 100 | 100 | 100 |
|---|---|---|---|
| Deadlines | 100 | 65 | 33 |
| Budgets | 33 | 33 | 33 |

In the case of EDF, the satisfying budget could be reduced from 35 to 33 by 1 time unit deduction from the original deadline. This is because the deadlines determine priorities of tasks scheduled by EDF, thus ensuring that $T_1$ has a shorter deadline will give it higher priority, in effect reducing the MST.

On the contrary, in the case of RM, any deduction from the original deadline could not reduce the satisfying budget. The deadlines of tasks scheduled by RM do not affect the MST of the tasks, thus shortening the task deadlines when using RM does not help enhancing the resource utilization of the child components.

# 5. RELATED WORK

An analytical compositional framework was presented in [17] as a basis for the schedulability analysis of hierarchical scheduling systems. It relies on the abstraction and composition of system components given by periodic interfaces. The authors of [15] extend their previous work [17] by considering multiprocessors based on cluster-based scheduling while using analytical methods to perform the analysis. However, in both [17] and [15], the system entities are given in terms of periodic interfaces without any specification of the tasks behavior and dependency.

CARTS (Compositional Analysis of Real-Time Systems) tool [14] is an implementation of the theory given in [17, 15]. Compared to our work, CARTS is a mature tool that is easy to use. On the other hand, we describe more detailed modeling and analysis.

Lipari et al [11] provide an analytical framework for the formal specification and schedulability analysis of hierarchical scheduling systems. They also present a methodology of how to compute the timing requirements of the intermediate levels (servers) making a set of tasks feasible. The framework only considers static priority scheduling (Fixed Priority Scheduling). Compared to that, we generalize the analysis and estimation of the timing requirements to both static and dynamic scheduling mechanisms.

Recently, model-based settings [2, 5, 7, 3] are getting intensively used for the schedulability analysis of real-time systems due to: 1) the ability to describe more concrete behaviors; 2) rigorous analysis (model-checking) with absolute certainty; 3) simulation of the system execution which can be helpful for the improvement and understandability.

The resource models represent an interface between a component and the rest of the system. In [9], the authors introduced the Dual Periodic Resource Model (DPRM) and presented an algorithm for computing the optimal resource interface, reducing the overhead suffered by the classical periodic resource models. However, since the actual workload behavior is not considered the resource gain is basically obtained through the reduction of the overhead, i.e. it is not measured on the task execution.

In [13], the authors introduced a technique for improving the schedulability of scheduling systems by reducing the resource interferences between tasks. The authors use shapers to reduce the resource interference between higher-priority and lower-priority tasks, and thus enable more lower-priority tasks to be scheduled. However, this work is restricted to fixed priority scheduling only.

The authors of [10] introduce an interesting observation that suggests an insight toward pessimism reduction in the schedulability analysis of multi-core systems running under Earliest Deadline first until Zero-Laxity (EDZL) policy. This work demonstrates that the proposed EDZL test not only has lower time complexity than existing EDZL schedulability tests, but also significantly improves the schedulability of systems running under EDZL policy.

In a similar way to the classical schedulability analysis where demand and supply bound functions can be calculated for individual components (viewed as regular scheduling systems) while considering the most pessimistic cases, the authors of [3] introduce a compositional schedulability analysis framework for component-based scheduling systems. For compositionality purposes, the resource model (supplier) behavior is set to be non-deterministic in order to simulate all potential supply scenarios. However, some of the supply scenarios may not happen if the system components are analyzed together. This may lead a system to be non-schedulable when using compositional analysis while it could be schedulable if one analyzes the entire system at once.

In [4], the authors introduce a new resource supplier model, called Synchronous Periodic Resource Model, in order to increase the resource utilization and make more systems schedulable. Such a resource model delays the resource supply in order to avoid supplying resource when it is not needed. Another observation established by the authors is that restricting the potential task offsets leads to reduce the resource requirements of system components. However, because such a framework does not capture the whole system together, and rather analyzes each component individually, it considers all potential preemption scenarios when analyzing components which may lead to an over-approximation of the resource supply.

In this paper, we proposed a new model of an HSS which enables to capture the whole system together in order to reduce the resource requirements of individual components, by calculating more optimal interfaces compared to the state

of the art. Moreover, we introduced a new theoretical calculation of the resource supply bound function in order to validate our model-based claim.

# 6. CONCLUSION

In the design of modern automotive systems, a manufacturer tries to maximize the number of components, provided by different suppliers, to be integrated on a given platform in order to reduce the system cost. This paper's contribution focuses on reducing the resource requirements of individual components, by calculating more optimal interfaces compared to the state of the art, thus enhancing the resource utilization of the whole system.

In this paper we have presented two new tighter supply bound functions for checking the resource requirements and schedulablility of components in hierarchical scheduling systems. The new supply bound functions are evaluated in the context of a model-based hierarchical scheduling framework.

The system architecture we considered is given in terms of hierarchical components, while the basic workload is given by tasks. Besides, we used stopwatch automata and UPPAAL tools (symbolic and statistical model checkers) to model and analyze such hierarchical systems. To evaluate our contribution, we compare our analysis results to a state of the art tool CARTS.

From our optimization results, we observed that the maximum service time of a resource model is significant for evaluating a resource model in optimizing an HSS evaluated by CARTS. Conclusively, the MST can be used to evaluate and optimize resource requirements of an HSS component.

One of the most important aspects for further study is the scalability of the method we have presented in this paper. Other future work could be the modeling and analysis of an industrial case study with concerns of overheads in order to study the applicability of the framework.

# 7. REFERENCES


[1] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 182–190, Dec 1990.

[2] M. Behnam, T. Nolte, I. Shin, M. Åsberg, and R. Bril. Towards hierarchical scheduling in VxWorks. In *OSPERT 2008*, pages 63–72.

[3] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikučionis, U. Nyman, and A. Skou. Hierarchical scheduling framework based on compositional analysis using uppaal. In *Proceedings of FACS 2013*, LNCS Volume 8348. Springer, 2013.

[4] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikucionis, U. Nyman, and A. Skou. Widening the schedulability of hierarchical scheduling systems. In *FACS 2014,*, pages 209–227, 2014.

[5] L. Carnevali, A. Pinzuti, and E. Vicario. Compositional verification for hierarchical scheduling of real-time systems. *IEEE Transactions on Software Engineering*, 39(5):638–657, 2013.

[6] F. Cassez and K. G. Larsen. The impressive power of stopwatches. In C. Palamidessi, editor, *CONCUR*, volume 1877 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 2000.

[7] A. David, K. G. Larsen, A. Legay, and M. Mikucionis. Schedulability of herschel-planck revisited using statistical model checking. In *ISoLA (2)*, volume 7610 of *LNCS*, pages 293–307. Springer, 2012.

[8] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using edp resource models. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 129–138, Dec 2007.

[9] J. Lee, L. T. X. Phan, S. Chen, O. Sokolsky, and I. Lee. Improving resource utilization for compositional scheduling using dprm interfaces. *SIGBED Rev.*, 8(1):38–45, Mar. 2011.

[10] J. Lee and I. Shin. Edzl schedulability analysis in real-time multicore scheduling. *Software Engineering, IEEE Transactions on*, 39(7):910–916, July 2013.

[11] G. Lipari and E. Bini. A methodology for designing hierarchical scheduling systems. *J. Embedded Comput.*, 1(2):257–269, Apr. 2005.

[12] R. J. B. Mike Holenderski and J. J. Lukkien. An efficient hierarchical scheduling framework for the automotive domain. In *Real-Time Systems, Architecture, Scheduling, and Application*, pages 67–94. InTech, 2012.

[13] L. T. X. Phan and I. Lee. Improving schedulability of fixed-priority real-time systems using shapers. In *Proceedings of RTAS '13*, pages 217–226, Washington, DC, USA, 2013. IEEE Computer Society.

[14] L. T. X. Phan, J. Lee, A. Easwaran, V. Ramaswamy, S. Chen, I. Lee, and O. Sokolsky. CARTS: a tool for compositional analysis of real-time systems. *SIGBED Rev.*, 8(1):62–63, Mar. 2011.

[15] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *ECRTS*, pages 181–190. IEEE Computer Society, 2008.

[16] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *RTSS'03*, pages 2–13. IEEE Computer Society, 2003.

[17] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embedded Comput. Syst.*, 7(3), 2008.