Hierarchical Scheduling Framework Based on Compositional Analysis Using Uppaal

Boudjadar, Jalil; David, Alexandre; Kim, Jin Hyun; Larsen, Kim Guldstrand; Mikučionis, Marius; Nyman, Ulrik; Skou, Arne

Published in:
Proceedings of Formal Aspects of Component Software (FACS), 2013

DOI (link to publication from Publisher):
10.1007/978-3-319-07602-7_6

Publication date:
2014

Document Version
Accepted author manuscript, peer reviewed version

Citation for published version (APA):
Boudjadar, J., David, A., Kim, J. H., Larsen, K. G., Mikučionis, M., Nyman, U., & Skou, A. (2014). Hierarchical Scheduling Framework Based on Compositional Analysis Using Uppaal. In J. L. Fiadeiro, Z. Liu, & J. Xue (Eds.), *Proceedings of Formal Aspects of Component Software (FACS), 2013* (Vol. 8348, pp. 61-78). Article 6 Springer Publishing Company. https://doi.org/10.1007/978-3-319-07602-7_6

Hierarchical Scheduling Framework Based on Compositional Analysis Using Uppaal[⋆]

Abdeldjalil Boudjadar, Alexandre David, Jin Hyun Kim, Kim. G. Larsen, Marius Mikučionis, Ulrik Nyman, Arne Skou

Computer Science, Aalborg University, Denmark

Abstract. This paper introduces a reconfigurable compositional scheduling framework, in which the hierarchical structure, the scheduling policies, the concrete task behavior and the shared resources can all be reconfigured. The behavior of each periodic preemptive task is given as a list of timed actions, which are some of the inputs for the parameterized timed automata that make up the framework. Components may have different scheduling policies, and each component is analyzed independently using Uppaal. We have applied our framework for the schedulability analysis of an avionics system.

1 Introduction

Embedded systems are involved in many applications, software systems in cars and planes, on which our lives depend. Ensuring the continually correct operation of such systems is an essential task. Avionics and automotive systems consist of both safety-critical and non safety-critical features, which are implemented in components that might share resources (e.g. processors). Resource utilization is still an issue for safety-critical systems, and thus it is important to have both an efficient and reliable scheduling policy for the individual parts of the system. Scheduling is a widely used mechanism for guaranteeing that the different components of a system will be provided with the correct amount of resources. In this paper, we propose a model-based approach for analyzing the schedulability of hierarchical scheduling systems. In fact, our framework is implemented using parameterized timed automata models.

A hierarchical scheduling system consists of a finite set of components, a scheduling policy and (global) resources. Each component, in turn, is the parallel composition of a finite set of entities which are either tasks or other components together with a scheduling policy to manage the component workload. One can remark that we do not consider component local resources. System tasks are instances of the same timed automaton with different input parameters. A special parameter of the task model is a list of timed actions [5], specifying the concrete behavior of the given task. This list includes abstract computation steps, locking and unlocking resources. Thanks to the parameterization, the framework can

⋆ The research presented in this paper has been partially supported by EU Artemis Projects CRAFTERS and MBAT.

easily be instantiated for a specific hierarchical scheduling application. Similarly, each scheduling policy (e.g. EDF: Earliest Deadline First, FPS: Fixed Priority Scheduling, RM: Rate Monotonic) is separately modeled and can be instantiated for any component.

Compositional analysis has been introduced [4, 10], as a key model-checking technology, to deal with state space explosion caused be the parallel composition of components. We are applying compositional verification to the domain of schedulability analysis.

We analyze the model in a compositional manner, the schedulability of each component including the top level, is analyzed together with the interface specifications of the level directly below it. In this analysis, we non-deterministically supply the required resources of each component, i.e. each component is guaranteed to be provided its required resources for each period. This fact is viewed by the component entities as a contract by which the component is obliged to supply the required resources, provided by the component parent level, to its sub entities for each period. The main contribution of the paper is combining:

- *a compositional analysis approach* where the schedulability of a system relies on the recursive schedulability analysis of its individual subsystems.
- *a reconfigurable schedulability framework* where a system structure can be instantiated in different configurations to fit different applications.
- *modeling of concrete task behavior* as a sequence of timed actions requiring CPU and resources.

The rest of the paper is structured as follows: Section 2 introduces related work. Section 3 is an informal description of the main contribution using a running example. The section gives an overview of both modeling hierarchical scheduling systems and how we perform the schedulability analysis in a compositional way. In section 4, we give the UPPAAL model of our framework where we consider concrete behavior of tasks. Moreover, we show how the compositional analysis can be applied on the model using the UPPAAL and UPPAAL SMC verification engines. Section 5 shows the applicability of our framework, where we analyze the schedulability of an avionics system. Finally, section 6 concludes our paper and outlines the future work.

2 Related Work

Hierarchical scheduling systems were introduced in [9, 7]. An analytical compositional framework for hierarchical scheduling systems was presented in [12] as a formal way to elaborate a compositional approach for schedulability analysis of hierarchical scheduling systems [13]. In the same way, the authors of [11] dealt with a hierarchical scheduling framework for multiprocessors based on cluster-based scheduling. They used analytical methods to perform analysis, however both approaches [12, 11] have difficulty in dealing with complicated behavior of tasks.

Recent research within schedulability analysis increasingly uses model-based approaches, because this allows for modeling more complicated behavior of systems. The rest of the related work presented in this section focuses on model-based approaches.

In [2], the authors analyzed the schedulability of hierarchical scheduling systems, using a model-based approach with the TIMES tool [1], and implemented their model in VxWorks [2]. They constructed an abstract task model as well as scheduling algorithms, where the schedulability analysis of a component does not only consider the timing attributes of that component but also the timing attributes of the other components that can preempt the execution of the component under analysis.

In [5], the authors introduced a model-based framework using Uppaal for the schedulability analysis of flat systems. They modeled the concrete task behavior as a sequence of timed actions, each one represents a command that uses processing and system resources and consumes time.

The authors of [3] provided a compositional framework for the verification of hierarchical scheduling systems using a model-based approach. They specified the system behavior in terms of preemptive time Petri nets and analyzed the system schedulability using different scheduling policies.

We combine and extend these approaches [3, 5] by considering hierarchy, resource sharing and concrete task behavior, while analyzing hierarchical scheduling systems in a compositional way. Moreover, our model can easily be reconfigured to fit any specific application. Comparing our model-based approach to analytical ones, our framework enables to describe more complicated and concrete systems.

3 Compositional Scheduling Framework

A hierarchical scheduling system consists of multiple scheduling systems in a hierarchical structure. It can be represented as a tree of nodes, where each node in the system is equipped with a scheduler for scheduling its child components.

In this paper, we structure our system model as a set of hierarchical components. Each component, in turn, is the parallel composition of a set of entities (components or tasks) together with a local scheduler and possible local resources. A parent component treats the real-time interface of each one of its child components as a single task with the given real-time interface. The component supplies its child entities with resource allocation according to their real-time interfaces. Namely, each component is parameterized by a period (prd), a budget (budget) specifying the execution time that the component should be provided by its parent level, and a scheduling policy (s) specifying resource allocations that are provided by the component to its child entities. The analysis of a component (scheduling unit) consists of checking that its child entities can be scheduled within the component budget according to the component scheduling policy. A component can be also parameterized by a set of typed resources (R) which serve as component local resources. An example of a hierarchical scheduling system is depicted in Fig. 1.

Tasks represent the concrete behavior of the system. They are parameterized with period (prd), execution time (e), deadline (d), priority (prio) and preemption (p). The execution time (e) specifies the CPU usage time required by the task execution for each period (prd). Deadline parameter (d) represents the latest point in time that the task execution should be done before. The parameter prio specifies the user priority associated to the task. Finally, p is a Boolean flag stating whether or not the task is preemptive.

The task behavior is a sequence of timed actions consuming CPU time and resources. Moreover, task and component parameters prd, budget and e can be single values or time intervals.

3.1 Motivating Example

In this section and throughout the paper, we present the running example shown in Fig. 1 to illustrate our system model of hierarchical scheduling systems, and show the compositional analysis we claim. For the sake of simplicity, we omit some parameters like priorities and resources and only consider single parameter values instead of time intervals.

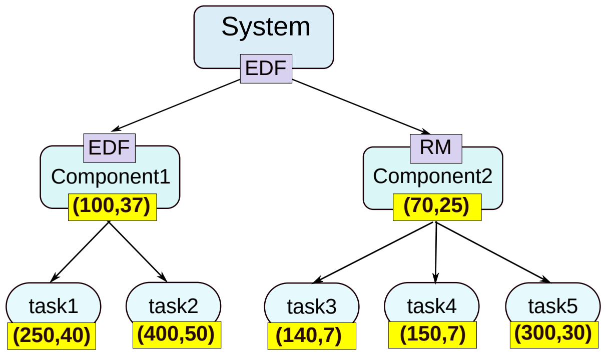


Fig. 1. Example of hierarchical scheduling system.

In this example, the top level System schedules Component1, Component2 with the EDF scheduling algorithm. The components are viewed by the top level System as tasks having timing requirements. Component1, respectively Component2, has the interface (100, 37), respectively (70, 25), as period and execution time. The system shown through this example is schedulable if each component, including the top level, is schedulable. Thus, for the given timing requirements Component1 and Component2 should be schedulable by the top level System according to the EDF scheduling policy. The tasks task1 and task2 should be schedulable, with respect to the timing requirement of Component1 (100, 37), also under the EDF scheduling policy. Similarly, task3, task4 and task5 should be schedulable, with respect to the timing requirements of Component2, under the RM scheduling policy. The next section presents the compositional analysis of the schedulability of our example.

For a given system structure, we can have many different system configurations. A system configuration consists of an instantiation of the model where each parameter has a specific value. Fig. 1 shows one such instantiation.

3.2 Our Analysis Approach

In order to design a framework that scales well for the analysis of larger hierarchical scheduling systems, we have decided to use a compositional approach. Fig. 2 shows how the scheduling system, depicted in Fig. 1, is analyzed using three independent analysis steps. These steps can be performed in any order.

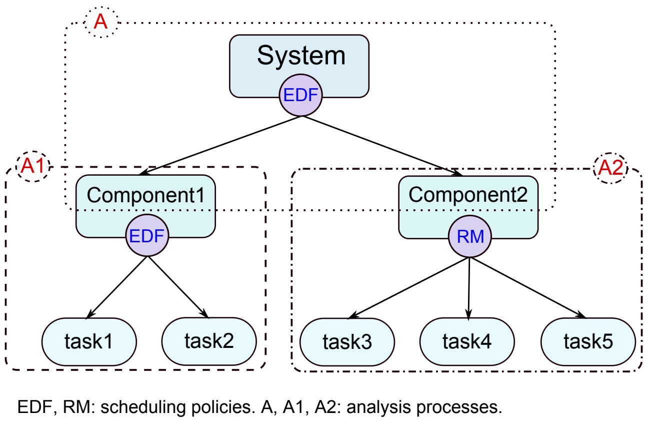


Fig. 2. Compositional analysis

The schedulability of each component, including the top level, is analyzed together with the interface specifications of the level directly below it. Accordingly, we will never analyze the whole hierarchy at once. In Fig. 2, the analysis process A consists of checking whether the two components Component1 and Component2 are schedulable under the scheduling policy EDF. In this analysis step, we only consider the interfaces of components in the form of their execution-time (budget) and period, so that we consider the component as an abstract task when performing the schedulability analysis of the level above it. In this way, we consider the *component-composition* problem similarly to [14] but using a non-deterministic supplier model for the interfaces. When performing an analysis step like A1, the resource supplier is not part of the analysis. In order to handle this, we add a non-deterministic supplier to the model. The supplier will guarantee to provide the amount of execution time, specified in the interface of Component1, before the end of the component period. We check all possible ways in which the resources can be supplied to the subsystem in A1. The supplier of each component provides resources to the child entities of that component in a non-deterministic way. During the analysis of A1, the supplier non-deterministically decides to start or stop supplying, while still guaranteeing to provide the required amount to its sub entities before the end of the period. The analysis A2 is performed in the same way as A1.

Our compositional analysis approach results in an over-approximation i.e. when performing the analysis of a subsystem, we over-approximate the behavior of the rest of the system. This can result in specific hierarchical scheduling systems that could be schedulable if one considers the entire system at once, but that is not schedulable using our compositional approach. We consider this fact as a design choice which ensures separation of concerns, meaning that small changes to one part of the system does not effect the behavior of other components. In this way, the design of the system is more stable which in turn leads to predictable system behavior. This over-approximation, which is used as a design choice, should not be confused with the over-approximation used in the verification algorithm inside the UPPAAL verification engine (Section 4.4). The result can either be *true (false)* or *maybe-not (maybe)*, in the case of *true (false)* the result of the analysis is conclusive and exact.

Thanks to the parameterization of system entities; scheduling policies, preemptiveness, execution times, periods and budgets can all easily be changed. In order to estimate the performance and schedulability of our running example, we have evaluated a number of different configurations of the system. This allows us to choose the best of the evaluated configurations of the system.

4 Modeling and Analysis using UPPAAL

The purpose of modeling and analyzing hierarchical scheduling systems is to check whether the tasks nested in each component are schedulable, with respect to resource constraints given by the component. This means that the minimum budget of a component supplier, for a specific period, should satisfy the timing requirements of the child tasks. For this purpose, we consider a scheduling unit and use symbolic model checking and statistical model checking methods to check the schedulability, and to find out the minimum budgets of components. In fact, a scheduling unit [13] consists of a set of tasks, a supplier and a scheduler, in [13] known by the terms Workloads, Resource model and Scheduling policy.

This section presents our modeling framework that will be used for the schedulability analysis. We revisit the running example shown in Fig. 1, which is built on the instances of four different UPPAAL timed automata templates: 1) non-deterministic supplier 2) periodic task 3) CPU scheduler (EDF, RM), and 4) resource manager. Similarly to [5], we also use broadcast channels where no sender can be blocked when performing a synchronization. We use stop watches, writing $x' == e$ to specify a clock x that can only progress when e evaluates to 1. UPPAAL also allows for clocks to progress with other rates but we only use 0 and 1.

4.1 Non-Deterministic Supplier Model

In this section, we present some arguments for why it makes sense to use a non-deterministic supplier model in our compositional analysis. The hierarchical

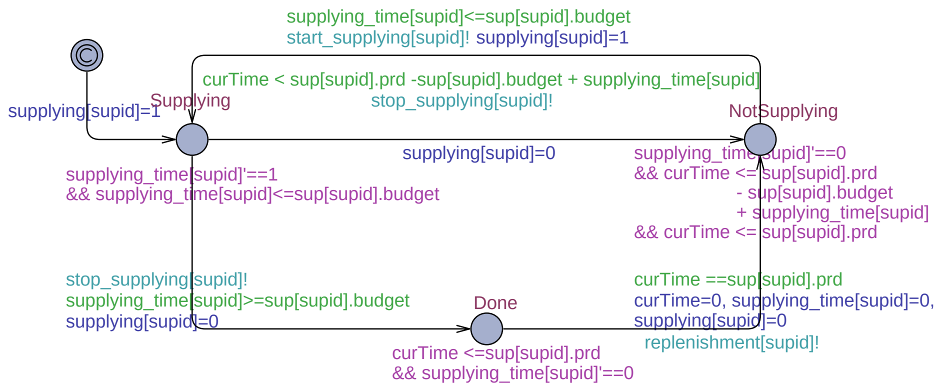

supplying_time[supid]<=sup[supid].budget
start_supplying[supid]! supplying[supid]=1
curTime < sup[supid].prd -sup[supid].budget + supplying_time[supid]
stop_supplying[supid]!
Supplying
NotSupplying
supplying[supid]=1
supplying[supid]=0
supplying_time[supid]'==1
&& supplying_time[supid]<=sup[supid].budget
supplying_time[supid]'==0
&& curTime <= sup[supid].prd
- sup[supid].budget
+ supplying_time[supid]
&& curTime <= sup[supid].prd
stop_supplying[supid]!
supplying_time[supid]>=sup[supid].budget
supplying[supid]=0
Done
curTime ==sup[supid].prd
curTime=0, supplying_time[supid]=0,
supplying[supid]=0
replenishment[supid]!
curTime <=sup[supid].prd
&& supplying_time[supid]'==0

Fig. 3. Non-deterministic supplier template

scheduling system structure is a set of scheduling components, each one includes a single specific scheduling algorithm and a set of entities (tasks or components). To analyze a single component by means of a compositional manner, it is necessary to consider the interrupted behavior of that component by the other concurrent components within the same system. However, it is hard to capture the interrupting behavior of the other components that influence the component under analysis. For this reason, we introduced a non-deterministic supplier to model all scenarios that the component under analysis can run. Such a non-deterministic fact simulates the influence of the other system components on the execution of the component under analysis.

As mentioned earlier, the non-deterministic supplier is a resource model that provides resources to the component. The scheduling policy within the component then allocates the resources to tasks. It also abstracts the possibility that a task from another part of the system (not part of the current analysis step) could preempt the execution of tasks of the current component.

Fig. 3 depicts the UPPAAL template model of the non-deterministic supplier. In fact, the non-deterministic supplier assigns a resource, denoted by rid, to a set of tasks characterized by the timing attributes given in listing 1.1.

Listing 1.1. Component interface

```
typedef struct {
  time_t      prd;
  time_t      budget;
  tid_t       task_arr[tid_t];
} sup_t;
```

A resource rid can represent a processing unit (CPU) or a any other system resource, represented in the model by a semaphore. prd is a period and budget is the amount of resources to be provided. The supplier assigns the budget amount of resources to tasks in task_arr[tid_t]. In this model, supplying_time[supid] (supid

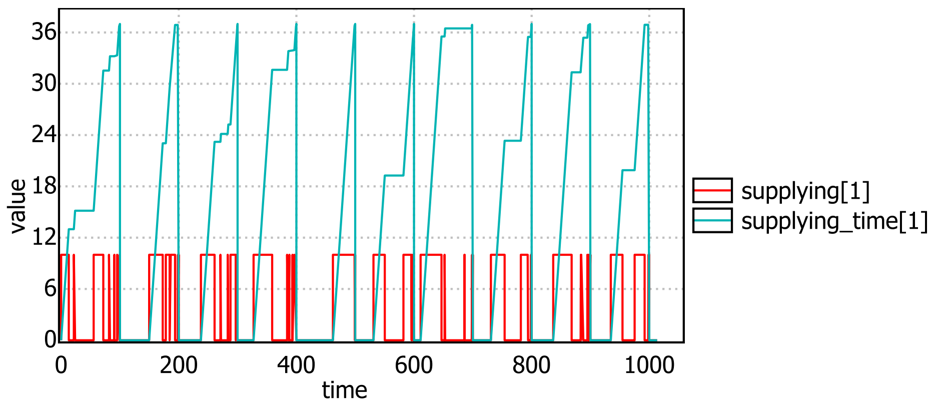


Fig. 4. Supplier's behavior

is the supplier identifier) represents the duration when the supplier provides a resource. start_supplying[supid] and stop_supplying[supid] are broadcast channels that notify tasks of the beginning and completion of the resource supply. curTime denotes the time elapsed since the beginning of the supplier's resource supplying. supplying[supid] contains the supplier's status, 0 (not supplying) or 1 (supplying).

Fig. 4 shows one particular resource supply pattern of Component1. supplying_time[1] is increasing while the supplier is providing resources. supplying_time[1] has a long wait while the supplier provides no resource. supplying[1] indicates whether the supplier provides resources or not. Fig. 4 shows a non-deterministic supply, in which the values of supplying[1] are irregular in behavior. The amount of resource supplied to tasks can be monitored from the supplying_time[1], which for each period supplies the exact amount of resource, 37 time units, given in the timing interface for Component1.

The supplier provides a resource at the location Supplying (Fig. 3). The transitions between Supplying and NotSupplying are non-deterministically taken until the budget is fulfilled ($supplying\_time[supid] <= sup[supid].budget$), or the remaining time is equal to the remaining amount of resource to be provided ($curtime <= sup[supid].prd - sup[supid].budget + supplying\_time[supid]$). The supplier stays at the location Supplying to provide the remaining amount of resource when the budget of the supplier is not fully provided, and the remaining time is equal to the remaining amount of resource to be provided.

4.2 Task model

We only consider a finite set of tasks and refer to them as $T = t_1, t_2, \ldots, t_n$. Each task is defined by the timing attributes given in listing 1.2.

pri is a task priority. initial_offset is an initial offset for the initial release of the task, and offset represents the offset time of each period before the task is released. A task has also best execution time and worst-case execution time.

Listing 1.2. Task data structure

```
typedef struct {
  pri_t        pri;
  time_t       initial_offset;
  time_t       offset;
  time_t       min_period;
  time_t       max_period;
  time_t       deadline;
  time_t       bcet;
  time_t       wcet;
  bool         preemptive;
 } task_t;
```

The timing attributes above are given as a structure associated to a timed automaton template. The task model is given by the template shown in Fig. 5.

Clock exeTime[tid] denotes the execution time in which the task has executed with necessary resources. This clock is a stop-watch and its progress depends on the following condition:

```
int[0,1] isTaskSched() { return rq[rid].element[0] == tid? 1:0;}
```

rq[j].element[0], where j is the resource id, contains the task identifier which is scheduled to use the CPU, and isTaskSched() returns 1 or 0 according to whether the corresponding task is scheduled or not. Thus, exeTime[tid] increases only when isTaskSched() returns 1. A clock tWECT[tid] measures the worst-case execution time for the task. curTime[tid] is the time elapsed since the task arrives. The task is scheduled, according to its priority, by a specific scheduling algorithm. It can execute only when the supplier provides it with resources. That is, the supplier provides a specific resource amount, then a scheduling algorithm assigns the use of that resource to a specific task. Fig. 6 shows the timed behavior of task1 and task2; exeTime[1] and exeTime[2] are increasing according to the resource supply from the supplier. They stop increasing when the supplier stops supplying the resource, or their corresponding tasks complete executing within their periods. Clock exeTime[2] starts increasing after exeTime[1] finishes its execution during its period because task2 has a lower priority than task1. running[1] and running[2] indicate whether the tasks are running or not.

4.3 Resource Model and Scheduling

Fig 7 shows both the resource manager template and one scheduling algorithm template. These two templates behave like a function. They process and return data instantaneously after they receive processing requests. Listing 1.3 depicts the structure (a queue) used by the resource manager.

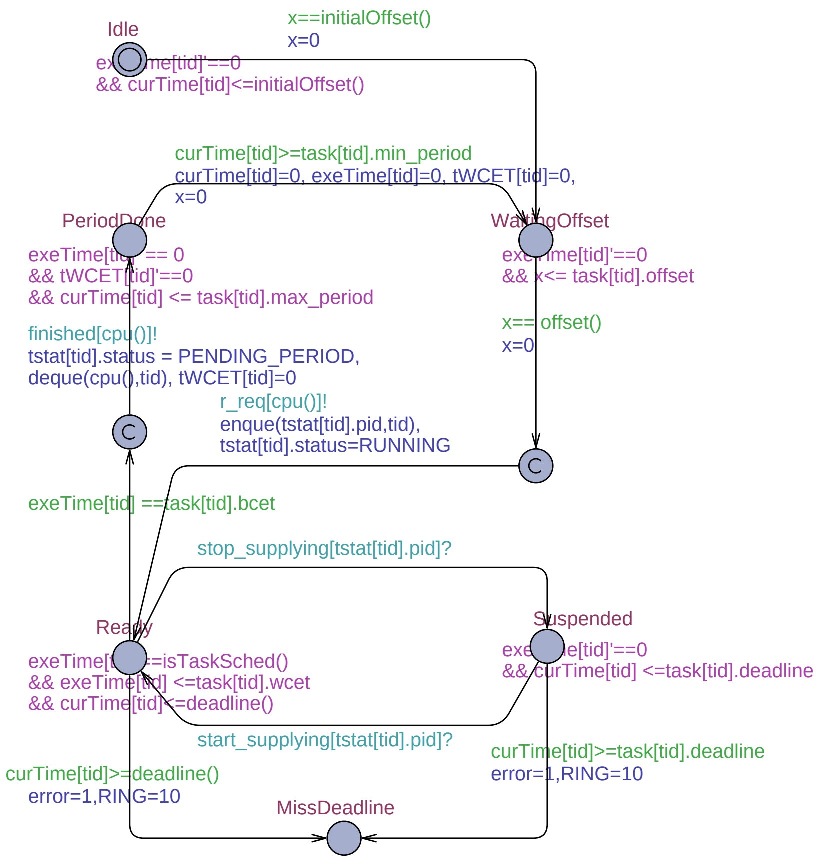

Idle
x==initialOffset()
x=0
exeTime[tid]'==0
&& curTime[tid]<=initialOffset()
curTime[tid]>=task[tid].min_period
curTime[tid]=0, exeTime[tid]=0, tWCET[tid]=0,
x=0
PeriodDone
WaitingOffset
exeTime[tid] == 0
&& tWCET[tid]'==0
&& curTime[tid] <= task[tid].max_period
exeTime[tid]'==0
&& x<= task[tid].offset
x== offset()
x=0
finished[cpu()]!
tstat[tid].status = PENDING_PERIOD,
deque(cpu(),tid), tWCET[tid]=0
r_req[cpu()]!
enque(tstat[tid].pid,tid),
tstat[tid].status=RUNNING
C
C
exeTime[tid] ==task[tid].bcet
stop_supplying[tstat[tid].pid]?
Suspended
Ready
exeTime[tid]'==0
&& curTime[tid] <=task[tid].deadline
exeTime[tid]'==isTaskSched()
&& exeTime[tid] <=task[tid].wcet
&& curTime[tid]<=deadline()
start_supplying[tstat[tid].pid]?
curTime[tid]>=task[tid].deadline
error=1,RING=10
curTime[tid]>=deadline()
error=1,RING=10
MissDeadline

Fig. 5. Task template

In fact, the resource manager shown in Fig 7(a) receives a scheduling request from a task, and requests a scheduling algorithm to select the highest priority task. The scheduling model of Fig 7(b) selects the highest priority task and places it at the first element of the ready queue. The scheduling model acknowledges the resource manager after the selection of a task. At this time, the resource manager notifies the selected task that it is scheduled in order to let it start its execution.

Thanks to the Uppaal instantiation mechanism, our system structure can easly be reconfigured. As early mentioned, we have modeled each system entity (task, resource, supplier, scheduling policy) by a template so that if, for example, we need to use a scheduling policy instead of another one, we just replace the scheduling policy name in the system instantiation.

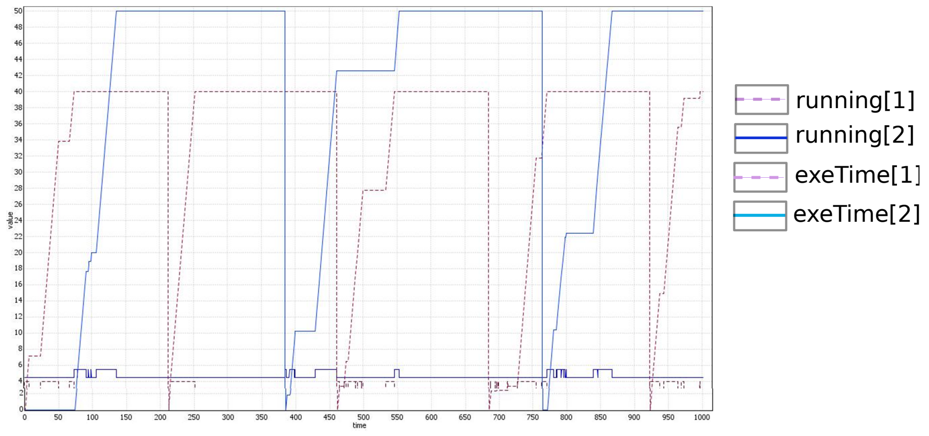


Fig. 6. Task behavior

Listing 1.3. Resource manager data structure

```
typedef struct {
  int[0,tid_n]      length;
  int[0,LastTid]  element[tid_n+1];
} queue_t;
```

4.4 Symbolic Model Checking

In this section, we explain how to check the schedulability using the symbolic reachability engine of Uppaal. We consider the system with various configurations in terms of preemptiveness, scheduling policy, etc.

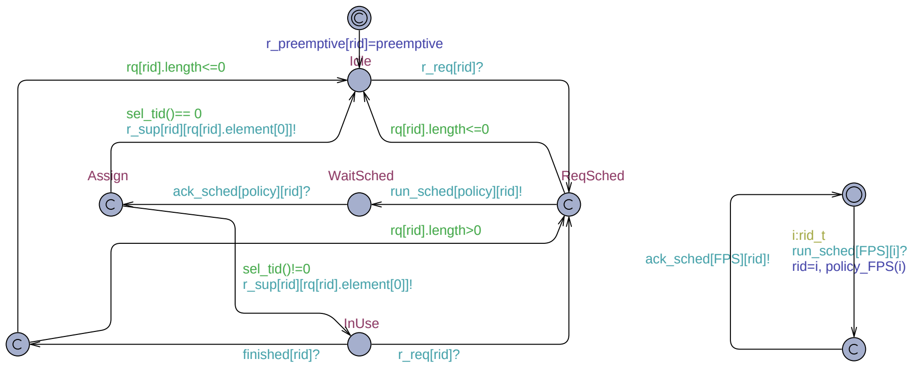
Let us start with an illustration of the schedulability analysis of Component1, depicted in Fig. 1. The components are verified with respect to the following safety property:

A[] error !=1

Here, error is a Boolean variable that will be updated to 1 (true) whenever a task misses its deadline. Thus, this property expresses the absence of deadline violation (i.e. all tasks are schedulable). For a given supplier with a timing requirement (100, 37), the verification results of the component including task1 (250, 40) and task2 (400, 50) are stated below:

Table 1. Budget evaluation based on scheduling policy and preemptiveness.

Component1	(100, 31)	(100, 37)		(100, 44)
	Preemptive	Preemptive	Non-preemptive	Preemptive
EDF	maybe not	Safe	maybe not	Safe
RM	maybe not	maybe not	maybe not	Safe



(a) Resource manager model template (b) FP scheduler

Fig. 7. Resource and scheduling algorithm templates

For the same task set under the EDF scheduling policy, the minimal budget in our verification framework can be greater than the optimal budget of the supplier given in [13]. One of the reasons is that the supplier behaves non-deterministically. The fact that UPPAAL uses an over-approximation technique to analyze models containing stop-watches leads to our framework also being an over-approximation. This results in the answer *maybe-not* to some of our verification attempts. We use the same task set as in [13] where the authors report that the optimal budget is 31 for the EDF scheduling policy, while the minimal budget we have computed to satisfy the same task set by symbolic model checking is 37. The minimal budgets we have computed, for RM scheduling and the same task set, are the same as the budgets presented in [13].

In order to obtain the upper bound on the WCETs of tasks, with respect to the EDF policy and a preemptive resource model, we check the following property:

```
sup: tWCET[1], tWCET[2]
```

where the tWCET[1] and tWCET[2] are stopwatches that are increasing while the corresponding tasks are running. sup is a UPPAAL keyword that refers to a function returning the supprima of the expressions (maximal values in case of integers; upper bounds, strict or not, for clocks). The verification results in $\mathsf{tWCET[1]} \leq 196$ and $\mathsf{tWCET[2]} \leq 196$, signifying that the WCETs of each task is less than or equal to 196. So none of the tasks miss their deadline.

4.5 Statistical Model Checking

As stated in [5], the use of stop-watches in UPPAAL leads to an over-approximation which guarantees that safety properties are valid but reachability properties could be spurious. Thus, symbolic model checking cannot disprove whether tasks are schedulable but only prove when they are schedulable. For that reason, we apply statistical model checking (SMC) to disapprove the schedulability and estimate the minimum budget of the supplier with respect to a specific period.

SMC is a simulation-based approach which estimates the probability for a system to satisfy a property by simulating and observing some of its executions, and then applies statistical algorithms to obtain the result [6]. In this section, we will show a way not only of checking schedulability but also to reason on the execution of tasks.

Table 2. Probability of error estimation with 1% level of significance.

Component1: EDF	(100, 31)	(100,32)	(100, 33)	(100, 34)	(100,35)
Pr[<=10000](<>error)	[0.249,0.349]	[0.0142,0.114]	[0,0.0987]	[0,0.0987]	[0,0.0987]

For our running example, Table 2 shows the query used to evaluate the probability of violating a deadline for runs bounded by 10000 time units regarding different budgets of the supplier. The SMC computed the mentioned results with certain level of confidence and precision, i.e. each result is given as an interval. However, if the lower bound is strictly positive, it guarantees that the checker found at least one witness trace where a task missed its deadline [5]. One may remark that the probability of tasks missing their deadline is much higher when the supplier budget is too small. Note that the possibility that tasks will miss their deadline is between 0.249 and 0.349 for the supplier timing requirement (100,31) of our example.

To visualize a witness of the deadline violation, we can request the checker to generate random simulation runs and show the value of a collection of expressions. For example, run the following query on the system:

```
simulate 100 [<=3000]{3+running[1], exeTime[1],
        4.5+running[2], exeTime[2]}: 1: error==1
```

This query asks the checker to simulate randomly the system execution until the condition error == 1 becomes satisfied, and to generate the task status and the accumulated amount of the resource used by the two tasks.

Fig. 8 shows a case where task1 misses the deadline, visualizing the running status of tasks (running[1] and running[2]) in a Gantt chart and the accumulated amount of the resource used by tasks in a period (exeTime[1] and exeTime[2]). Notice that the flat line at the end of the execution of task1 is one value lower than all the previous tops of task1, indicating that this task misses the deadline because of the lack of 1 time unit at time 1,250.

We apply the following queries for different supplier requirements to generate the probability distribution of the worst-case execution time of tasks:

```
E[globalTime <= 100000;100] (max: tWCET[1])
E[globalTime <= 100000;100] (max: tWCET[2])
```

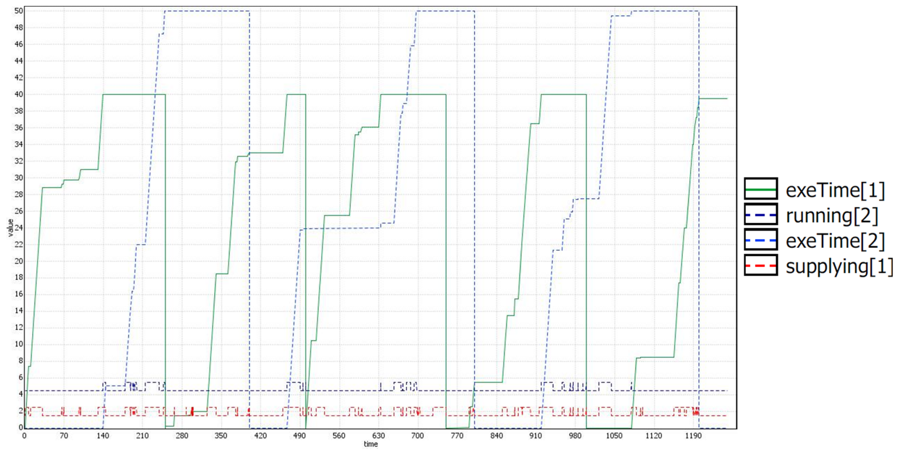
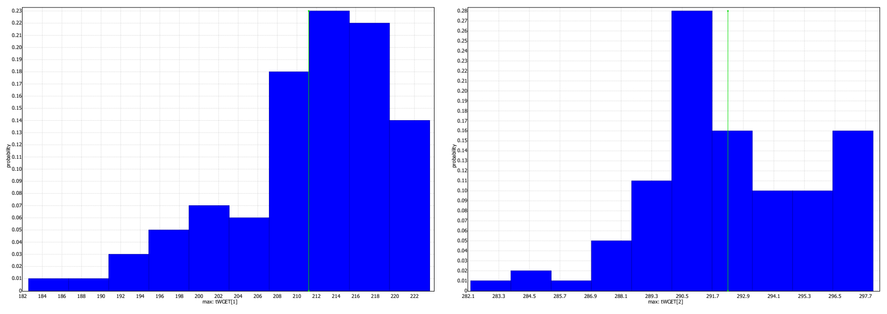


Fig. 8. Unschedulable tasks: task1 misses its deadline at time 1,250.



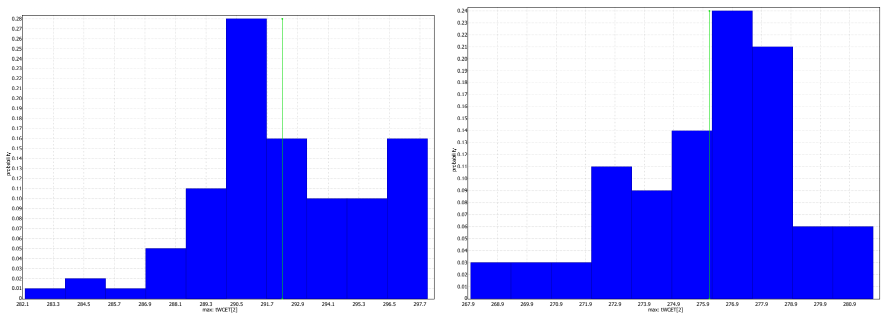
(a) task1 (b) task2

Fig. 9. Probability distribution of the WCET of tasks for the supplier (100, 33)

The results are shown in Fig 9 and Fig 10. In fact, Fig 9 shows the probability distribution of the worst-case execution time of tasks for the supplier timing requirement (100, 33), where task1 and task2 have 210.026 and 292.126 as worst-case execution times. For the supplier timing requirement (100, 37), as shown in Fig 10, task1 and task2 have 181.304 and 276.121 as worst-case execution times. By means of this reasoning, it can be checked that both cases for the supplier satisfies the task resource requirements and make them schedulable.

5 Case Study

To show the applicability of our compositional framework, we have modeled the avionics system introduced in [8, 3], and analyzed its schedulability. The application is a flat composition of 15 tasks declared with different priorities and timing requirements. Depending on the features of tasks, we have structured



(a) task1 (b) task2

Fig. 10. Probability distribution of the WCET of tasks for the supplier (100, 37)

this application in 3 components. Component 1 includes 5 tasks concerning the fire system like bomb button, weapon release, target tracking, etc. Component 2 encapsulates 5 tasks concerning the navigation system whereas component 3 includes the basic 5 tasks like display and auto-toggle. Each of the component has a scheduling policy and timing requirements (period, budget). We have also considered shared resources to perform input/output. In fact, the shared bus for input/output communication is modeled as a particular instance of the processor model. This can easily be extended to model multi-core platforms.

Following the analysis method described in section 3, we associate to each component a non-deterministic supplier. By holding the same timing requirements of tasks as [8], our compositional analysis shows that all components, except the top level one, are schedulable under different scheduling policies with or without preemptiveness. Component 1 is schedulable with at least 89/100 of the system resources. Component 2 could be schedulable if we provide 90/100 of the system resources, i.e. supplier interface (100,90), whereas Component 3 needs at least 67/100 of system resources to being schedulable.

In the top level analysis process (A in Fig. 2), the top level component which consists of a scheduling policy together with the interfaces of the 3 components cannot be scheduled because the sum of the 3 component supplier budgets exceeds 100% of the resource utilization. This non-schedulability is probably due to the existence of tasks having longer execution time than the deadline of the lowest priority task. Thus, according to our compositional analysis this avionics system is not schedulable. Our schedulability result of this avionics system matches perfectly with the schedulability result obtained in a non-compositional way in [8].

A challenge encountered during this application is the estimation of both period and budget of each supplier such that:

- each supplier provides enough resources to its child tasks.
- the parallel composition of all suppliers is schedulable according to the system level scheduling policy.

We have used a binary search approach to estimate the supplier budgets. In fact, we check the schedulability of a component by giving a supplier budget, and if the schedulability property is not satisfied we increase the budget value and rerun the verification process. A perspective of this work is to study the estimation of time requirements (periods, budgets) of the system intermediate levels in an automatic way, making then the checking process much faster.

6 Conclusions

We have defined a compositional framework for the modeling and schedulability analysis of hierarchical real-time systems. The framework has been instantiated as reusable models given in terms of timed automata which we analyzed using UPPAAL and UPPAAL SMC. The reusable models ensure that when modeling a hierarchical scheduling application, only the concrete task behavior and the hierarchical structure need to be specified by the system engineer. The framework also allows for instant changes of the scheduling policy at each given level in the hierarchy. Comparing our model-based approach to analytical ones, our framework enables the modeling of more complicated and concrete systems. We have successfully applied our compositional framework to model an avionics system and analyze its schedulability. As future work, we plan to study how to estimate the optimal timing requirements of suppliers in an automatic way. We also plan to consider multi-core platforms as well as energy efficiency.

References

1. T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Times: A tool for schedulability analysis and code generation of real-time systems. In K. G. Larsen and P. Niebert, editors, *FORMATS*, volume 2791 of *LNCS*, pages 60–72. Springer, 2003.
2. M. Behnam, T. Nolte, I. Shin, M. Åsberg, and R. Bril. Towards hierarchical scheduling in VxWorks. In *OSPERT 2008*, pages 63–72.
3. 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.
4. E. M. Clarke, D. E. Long, and K. L. Mcmillan. Compositional model checking. MIT Press, 1999.
5. 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.
6. A. David, K. G. Larsen, A. Legay, M. Mikucionis, D. B. Poulsen, J. van Vliet, and Z. Wang. Statistical model checking for networks of priced timed automata. In U. Fahrenberg and S. Tripakis, editors, *FORMATS*, volume 6919 of *Lecture Notes in Computer Science*, pages 80–96. Springer, 2011.
7. Z. Deng and J. W.-S. Liu. Scheduling real-time applications in an open environment. In *RTSS*, pages 308–319. IEEE Computer Society, 1997.
8. R. Dodd. Coloured petri net modelling of a generic avionics missions computer. Technical report, 2006.

9. X. A. Feng and A. K. Mok. A model of hierarchical real-time virtual resources. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium*, RTSS '02, pages 26–, Washington, DC, USA, 2002. IEEE Computer Society.
10. J. Lind-Nielsen, H. R. Andersen, H. Hulgaard, G. Behrmann, K. J. Kristoffersen, and K. G. Larsen. Verification of large state/event systems using compositionality and dependency analysis. *Formal Methods in System Design*, 18(1):5–23, 2001.
11. I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *ECRTS*, pages 181–190. IEEE Computer Society, 2008.
12. I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *RTSS*, pages 2–13. IEEE Computer Society, 2003.
13. I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embedded Comput. Syst.*, 7(3), 2008.
14. I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embed. Comput. Syst.*, 7(3):30:1–30:39, May 2008.