Statistical and exact schedulability analysis of hierarchical scheduling systems

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

Published in:
Science of Computer Programming

DOI (link to publication from Publisher):
10.1016/j.scico.2016.05.008

Publication date:
2016

Document Version
Early version, also known as pre-print

Statistical and Exact Schedulability Analysis of Hierarchical Scheduling Systems[☆]

Abdeldjalil Boudjadar

Computer Science, Aalborg University, Denmark

Alexandre David

Computer Science, Aalborg University, Denmark

Jin Hyun Kim

Computer Science, Aalborg University, Denmark

Kim G. Larsen

Computer Science, Aalborg University, Denmark

Marius Mikučionis

Computer Science, Aalborg University, Denmark

Ulrik Nyman

Computer Science, Aalborg University, Denmark

Arne Skou

Computer Science, Aalborg University, Denmark

Abstract

This paper contains two contributions: 1) A development methodology involving two techniques to enhance the resource utilization and 2) a new generic multi-core resource model for hierarchical scheduling systems.

As the first contribution, we propose a two-stage development methodology relying on the adjustment of timing attributes in the detailed models during the design stage. We use a lightweight method (statistical model checking) for design exploration, easily assuring high confidence in the correctness of the models.

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

Email addresses: jalil@cs.aau.dk (Abdeldjalil Boudjadar), adavid@cs.aau.dk (Alexandre David), jin@cs.aau.dk (Jin Hyun Kim), kgl@cs.aau.dk (Kim G. Larsen), marius@cs.aau.dk (Marius Mikučionis), ulrik@cs.aau.dk (Ulrik Nyman), ask@cs.aau.dk (Arne Skou)

Once a satisfactory design has been found, it can be proved schedulable using the computation costly method (symbolic model checking). In order to analyze a hierarchical scheduling system compositionally, we introduce the notion of a stochastic supplier modeling the supply of resources from each component to its child components in the hierarchy. We specifically investigate two different techniques to widen the set of provably schedulable systems: 1) a new supplier model; 2) restricting the potential task offsets. We also provide a way to estimate the minimum resource supply (budget) that a component is required to provide. In contrast to analytical methods, we prove non-schedulable cases via concrete counterexamples. By having richer and more detailed scheduling models this framework, has the potential to prove the schedulability of more systems.

As the second contribution, we introduce a generic resource model for multi-core hierarchical scheduling systems, and show how it can be instantiated for classical resource models: Periodic Resource Models (PRM) and Explicit Deadline Periodic (EDP) resource models. The generic multi-core resource model is presented in the context of a compositional model-based approach for schedulability analysis of hierarchical scheduling systems. The multi-core framework presented in this paper is an extension of the single-core framework used for the analysis in the rest of the paper.

Keywords: Hierarchical Scheduling Systems, Schedulability Analysis, Resource utilization, Embedded Systems, Uppaal, Model Checking, Statistical Model Checking, Hybrid Automata, Stopwatch Automata.

---

1. Introduction

In a hierarchical scheduling systems a number of individual components are integrated into a single system running on one execution platform. Embedded hierarchical scheduling systems have reached a maturity level that enables their actual application on automotive and space systems [34, 19]. A class of analytical methods have been developed for hierarchical scheduling systems [41, 39]. Due to the rigorous nature of analytical methods, they are easy to apply once proven correct. On the other hand proving the correctness of an analytical method is in itself a research endeavor. 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 of needed resources. Model-based methodologies for schedulability analysis [14, 11, 19] allow the modeling of detailed and complicated behavior of individual tasks, relative to analytical methods. Due to the complexity and size of the systems, it is not feasible to analyze the complete system in one model. This leads us to adopt a compositional structure for our models and the applied analysis.

As part of the compositional description of a hierarchical scheduling system the resource supply from parent to child component in the hierarchy is described using separate resource models. Our resource model is specified in the form of a transition system, where in contrast to classical resource models we

have several supply states. To each supplying state we assign a specific supply pattern, thus enabling the description of much more complex supply scenarios. Unlike previous approaches our method enables the description of supply patterns handling multi-core aspects related to typed resources. Typed resources can be used to describe, among others, the types of CPU cores on homogeneous and heterogeneous execution platforms. As an example, two different supply states could specify the supply of respectively 2 and 4 computation cores to the child component. The resource supply could be preemptive and/or urgent in order to model the specific behavior of the execution environment. Our generic resource model can easily be instantiated for classical single-core resource models for hierarchical scheduling systems, such as the Periodic Resource Models (PRM) [42] and Explicit Deadline Periodic (EDP) [24] resource models.

Profiting from the technological advances in model checking, we provide a model-based methodology for the schedulability analysis of hierarchical scheduling systems. We model tasks, resources, schedulers and suppliers as Stopwatch Automata (SWA) [15]. 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). Our approach aims at increasing resource utilization by: 1) providing a new supplier model where the supply of resources is delayed as much as possible according to task requests, 2) adjusting task offsets relative to the component period. Our methodology also has the advantage that it is possible to update the system models such that they fit a specific system. With a dedicated modeling tool it could even be manageable for the system engineers to update the models in order to have a more realistic analysis of the system. In this way, they can utilize detailed knowledge of the system that they are working with; something that cannot be achieved with a classical analytical approach.

An example of a hierarchical scheduling system is depicted in Fig. 1. It includes two top level components Controls and Display and Nav.Ctrl scheduled according to the Earliest Deadline First (EDF) policy. Each component is characterized by timing requirements consisting of period and execution time (e.g. $(10, 6)$ for Nav. Ctrl). The attributes of tasks are similar to the ones of components. Task deadlines are the same as the task periods.

According to the CARTS tool [39], the hierarchical scheduling system of Fig. 1 is not schedulable. However using the specific approaches shown in this paper, this system can be shown to be schedulable using different offset parameters and/or a new resource supplier model.

Symbolic model checking offers absolute certainty that the verified properties are correct with regards to the model. However, it suffers from state space explosion and undecidability of certain properties (for a given specification), thus some models might not be feasible to check and others will take a long time to verify. Statistical model checking provides high confidence but not absolute certainty, and the results are obtained much faster than with symbolic model checking.

This paper is an extension of the conference paper [12], where we presented

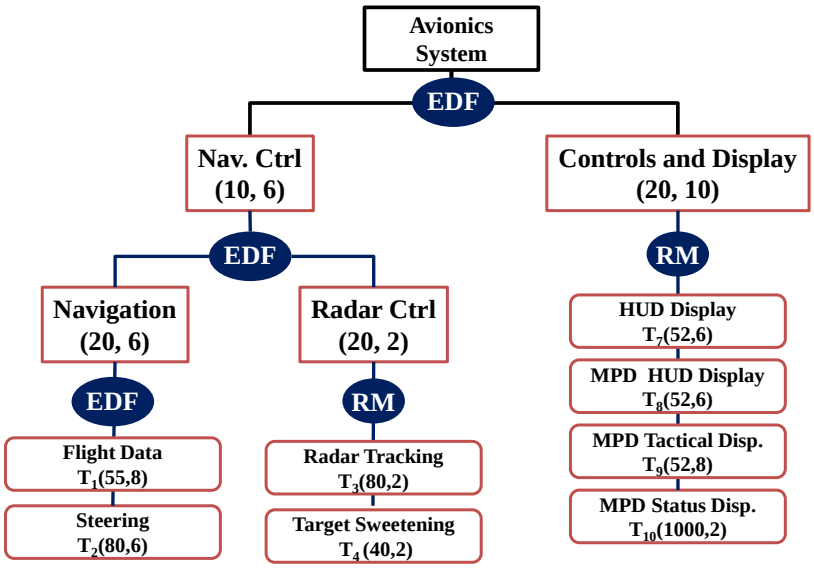


Figure 1: A hierarchical scheduling systems.

a methodology for compositional schedulability analysis for hierarchical scheduling systems, together with two techniques to enhance the resource utilization. Like the conference version we keep using the same methodology, as described in Section 2, to validate the schedulability of hierarchical scheduling systems, however in this paper we have more rich and expressive models that enable to capture more features in terms of resource supply patterns. We also introduce a new generic resource model for describing multi-core systems.

Besides to the introduction of the theoretical basis underlying our model-based framework, the new contribution of this paper includes:

- Revised UPPAAL models which vastly improve the size of scheduling systems that can be handled with symbolic model checking.
- New generic resource model for multi-core hierarchical scheduling systems that can be instantiated for any periodic resource model.

To bridge the gap between these new contributions, the new generic resource model will be used as a component supplier at different levels of the hierarchy; though it can be used as a supplier for the system level (root). Accordingly, since we do not assume dependency between tasks/components, more than one child entity (for example Navigation and Radar Ctrl) can run in parallel when

the supplier of their parent component (Nav.Ctrl) is supplying resource with a parallel pattern. We show how the behavior of such a multi-core resource is captured in UPPAAL, and how to derive an instantiation of this new model for 2 classical resource models: Periodic Resource Model (PRM) and Explicit Deadline Periodic (EDP) resource model. Thereafter, we study the impact of the new resource model, in particular the parallel supply pattern simulating multi-core platforms, on the system schedulability and scalability.

As described in Section 2, the general methodology, in both [12] and this paper, consists of using a low cost statistical method for the design exploration; and a costly but absolute certain symbolic model checking method for the final verification. When the design space exploration is performed using statistical model checking, one can determine optimal system parameters that could be impossible to find using classical analytical methods. Our framework is realized using an extension of Timed Automata (TA) called Stopwatch Automata (SWA) which enables the description and analysis of detailed task behavior and resource supply patterns, something which cannot be achieved using classical analytical methods [42, 25, 24, 43, 3, 14].

Using our framework more systems can be proven to be schedulable, since we are enhancing the resource utilization using the two techniques; 1) synchronous periodic resource model and 2) offset manipulation.

Similarly to [11], we use the notion of a stochastic supplier model in order to enable compositional analysis, such that the schedulability of each component can be analyzed separately. We evaluate our methodology by comparing our results to the ones obtained using the state of the art tool CARTS [39]. Our verification results are consistent with the results obtained from CARTS. When checking the schedulability of a system, our tools can prove the non-schedulability by means of a counterexample.

The rest of this paper is organized as follows: Section 2 highlights the problem that we are solving and differentiates between the different aspects of our proposed solution. Section 3 describes related work. Section 4 presents the formal basis underlying our model-based framework. Section 5 provides high level conceptual models of our framework. In Section 6, we give a new generic resource model for multi-core hierarchical scheduling systems. Section 7 presents our modeling and analysis of hierarchical scheduling systems with respect to both classical and new resource models using UPPAAL and UPPAAL SMC. Section 8 describes two techniques to improve resource utilization as well as an evaluation of the scalability. Section 9 compares our results with a state of the art tool and discusses the scalability and performance of our analysis methods. Section 10 concludes the paper.

2. Methodology and Challenges

The purpose of this section is to describe the methodological contribution that this paper shares with the conference version [12] and the motivation behind this methodology.

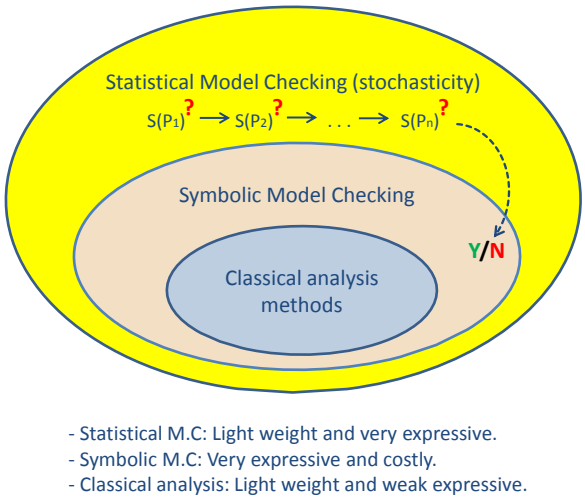


Figure 2: Classes of systems that different methods can prove schedulable.

This paper presents a general methodology, which could be instantiated using any modeling formalism supporting both a lightweight statistical analysis and a more costly formal verification. In this paper, the methodology is instantiated as a specific approach using Stopwatch Automata (SWA) together with the verification suite Uppaal SMC and Uppaal. Once the methodology has been instantiated with a specific model and associated tools we say that we have specific approach.

The paper also presents two concrete techniques for enhancing the resource utilization, which are described in Section 8. These concrete techniques will not be discussed further in this section.

The problem that we intend to solve is the following: a complex hierarchical system is being developed for a safety critical product. It is essential to produce both a safe system and a system which uses as few resources as possible.

The general principle of the methodology is the following: 1) design space exploration is carried out using a lightweight simulation based evaluation method in order to find good candidates for the task division and configuration of the system; 2) when a good candidate for a system configuration has been found, the same models can be reused with a different verification technique to establish with certainty that the system is schedulable.

Fig. 2 shows a graphical conceptual representation of different sets of systems that different methods can show to be schedulable. Systems that are easily proven schedulable using classical analytical approaches can also be proven correct using symbolic model checking. Systems that can be shown, with a high degree of certainty, to be correct using statistical model checking (SMC) cannot always be proven to be correct using symbolic model checking due to state space explosion. In the same way, some complex systems that are analyzable

using model checking cannot be proved correct using analytical approaches [14]. An obvious example is a system having an internal sequencing of tasks due to a dependency relation between tasks. Another trivial example is a scheduling system with different typed resources where the schedulability analysis considers the different resources together (simultaneously).

Our methodology consists of exploring system models with different sets of parameters ($S(P_i)$) searching for a realistic configuration that optimally satisfies the requirements. Basically, a configuration includes a set of tasks together with their timing attributes, the scheduling algorithm of each level of the hierarchy and a potential budget (the maximum resource amount to be provided) of each component. The experiments we have done are performed using SMC with a high confidence level. In that way, using SMC one can easily and interactively obtain either a high degree of confidence that the model is correct or a counter-example showing an error trace. When a satisfying final configuration has been found the system can be proven to be schedulable using symbolic model checking. In very rare cases an error could be found at this stage, but this is highly unlikely due to the confidence levels obtained using SMC.

3. Related Work

In an engineering setting, it is very desirable to easily determine parameters that will make a given system configuration schedulable and realizable. In this paper, while we explore the schedulability analysis of hierarchical scheduling systems by profiting from the technological advances made in the area of model checking, we propose a compositional analysis approach to determine and increase the potential configurations making much more hierarchical scheduling systems schedulable.

The concept of hierarchical scheduling systems was first introduced as 2 levels systems in [23], and then generalized as a real-time multi-level system by [35]. An example of the increasing use of hierarchical scheduling systems is the standard ARINC 653 [4] for avionics real-time operating systems. The following sections overview the ideas that our approach relies on.

3.1. Analytical Approaches to Schedulability Analysis

Several compositional analysis techniques [42, 25, 24, 43, 3, 14] have been proposed. Lipari et al [32] 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). We generalize the analysis and such an estimation of the timing requirements to any scheduling mechanism.

Davis and Burns improve their previous work [22] to analyze the schedulability of hierarchical scheduling systems where fixed priority scheduling is used both at the global and the local levels. The authors find that harmonic tasks

linked to the release of their server improve schedulability. We explore thoroughly the formal impact of synchronicity between the release of tasks and the start of the resource supplier, called offset manipulation, in Section. 8.

An analytical compositional framework was presented in [43] as a basis for the schedulability analysis of hierarchical scheduling systems. Such a framework relies on the abstraction and composition of system components, which are given by periodic interfaces. The interfaces state the components timing requirement without any specification of the tasks concrete behavior. The authors of [41] extend their previous work [43] to a hierarchical scheduling framework for multiprocessors based on cluster-based scheduling. Shin et al used analytical methods to perform the analysis. However, in both [43] and [41], the proposed framework is limited to a set of formulas describing an abstraction of the system entities. The system entities are given in terms of periodic interfaces, without any specification of the tasks behavior and interaction. CARTS (Compositional Analysis of Real-Time Systems) [39] is a tool which implements the theory given in [43, 41]. Compared to our approach CARTS is a mature tool that is easy to use. On the other hand, we provide a more detailed modeling and analysis.

3.2. Model-based Approaches to Schedulability Analysis

As common traits, analytical approaches assume computations with deterministic Execution Time usually coincident with the Worst Case Execution Time (WCET), and they provide pessimistic results [14]. Recent research within schedulability analysis gives tremendous attention to model-based approaches, because of their expressiveness which allows for modeling more complicated behavior of systems, and also due to the technological advances made in the area of model-based simulation and analysis tools. Behnam et al [6] analyze the schedulability of hierarchical scheduling systems using the TIMES tool [3, 40], and implement their model-based framework in VxWorks [6]. The authors construct an abstract task model as well as scheduling algorithms focusing on the component under analysis. However, the authors not only consider the timing attributes of the component under analysis but also the timing attributes of the other components that can preempt the execution of the current component. Thus, the proposed approach is not fully compositional. The authors of [14] provide a compositional framework modeled as preemptive Time Petri Nets for the verification of hierarchical scheduling systems using the ORIS tool [37]. Carnevali et al only analyze systems using two specific scheduling algorithms severely restricting the class of systems they can handle.

Sun et al introduce a component-based framework [44] for the analysis of hierarchical scheduling systems encoded using hybrid automata. The authors prove the correctness of their models and study the decidability of the reachability (schedulability) analysis for the case of periodic tasks. Unlike our framework where we restricted guard and update statements so that they depend only on discrete variables, Sun et al exploit the whole expressiveness capability of the Uppaal language. However, making guards and updates depending on continuous variables leads the analysis, using model checking, to be undecidable for

timed automata and pessimistically over-approximating in case of stopwatch automata.

Bøgholm et al introduce a model-based approach for the verification of safety critical hard real-time systems implemented in safety critical Java [9]. This work focuses on modeling the actual behavior of the execution platform, but does not use the concept of a hierarchical scheduling system. The concepts from this work could be combined with the current paper as the lowest level in a compositional approach.

The authors of [19] introduced a model-based framework using Uppaal for the schedulability analysis of single layered scheduling systems, modeling the concrete task behavior as a sequence of timed actions. We have been inspired by the work in [19] but generalizing and lifting it to a compositional approach for hierarchical scheduling systems.

3.3. Resource models for Hierarchical Scheduling Systems

Resource efficiency constitutes one of the most important factors in the performance evaluation of hierarchical scheduling systems. Such resources are often represented by either periodic [42] or explicit deadline periodic [24] resource models. The resource models represent an interface between a component and the rest of the system. In [31], the authors introduce the Dual Periodic Resource Model (DPRM) and present an algorithm for computing the optimal resource interface, reducing the overhead suffered by the classical periodic resource models. The authors of [38] introduce a technique for improving the schedulability of real-time scheduling systems by reducing the resource interferences between tasks.

In this paper, we propose a model-based framework for the modeling of hierarchical scheduling systems with a generic resource model, while we use Uppaal and Uppaal SMC to analyze the schedulability of components in a compositional manner. We show how such a generic resource model can be instantiated for any classical periodic resource model. Moreover, we introduce two novel techniques for improving the resource efficiency, and computing the minimum resource supply of system components. In our model-based framework we can also model the detailed behavior of specific tasks, specific arrival patterns and potential dependencies between tasks.

4. Background

This section presents the formal basis underlying our model-based framework.

4.1. Parameterized Stopwatch Automata

The modeling formalisms used in this paper range from classical timed automata to hybrid automata with algorithmic support from the various branches of the tool Uppaal. The classical version of Uppaal offers support for efficient symbolic verification of timed automata [1] and over-approximate verification

of stopwatch automata (SWA) [15]. The branch UPPAAL CORA extends the symbolic verification engine of UPPAAL to support cost-optimal reachability for priced timed automata [7, 2, 30].

Most recently the branch UPPAAL SMC [20, 21] provides highly scalable verification engine for statistical model checking (SMC) for not only the three formalisms above but stochastic hybrid automata in general. In essence, statistical model checking is based on stochastic semantics allowing for the probability of linear time properties to be estimated (or tested) with arbitrary precision and confidence through simulations.

UPPAAL SMC thus supports the analysis of stochastic hybrid automata (SHA) [18] that are timed automata whose clock rates can be changed to be constants or expressions depending on other clocks, effectively defining Ordinary Differential Equations (ODEs). This generalizes the model used in previous work [20, 21] where only linear priced automata were handled. The release UPPAAL SMC 4.1.18[1] supports fully hybrid automata with ODEs and a few built-in functions (such as `sin`, `cos`, `log`, `exp` and `sqrt`).

4.2. Hybrid Automata

Intuitively, a hybrid automaton $\mathcal{H}$ [27] is a finite-state automaton extended with continuous variables that evolve according to dynamics characterizing each discrete state (called a *location*). 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. 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 ν, $F(d, \nu)$ 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 hybrid automata, we assume a set of actions Σ, which is partitioned into disjoint sets of input and output actions, i.e. $\Sigma = \Sigma_i \uplus \Sigma_o$.

Definition 1. *A Hybrid Automaton (HA) $\mathcal{H}$ is a tuple $\mathcal{H} = (L, \ell_0, X, \Sigma, E, F, I)$, where: (i) L is a finite set of locations, (ii) $\ell_0 \in L$ is an 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 (Σ_i) and outputs (Σ_o), (v) E is a finite set of edges of the form $(\ell, g, a, \phi, \ell')$, where ℓ and ℓ' are locations, g is a predicate on $\mathbb{R}^X$, action label $a \in \Sigma$ and ϕ is a binary relation 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 ℓ.*

The semantics of a HA $\mathcal{H}$ 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 $\nu \models g$

[1] `www.uppaal.org`.

and $\phi(\nu, \nu')$. We write $(\ell, \nu) \rightsquigarrow (\ell', \nu')$ if there is a finite sequence of delay and discrete transitions from (ℓ, ν) to (ℓ', ν').

In the above definition, we have deliberately left open the concrete syntax for the delay function F as well as guards g, update predicate ϕ and invariant I. For *timed automata* (TA) [1], the continuous variables are simple clocks x where the delay update $F(\ell)$ is given by an implicit rate $x' = 1$. For *stopwatch automata* (SWA), the rate in a location ℓ may be either $x' = 1$ or $x' = 0$ (the latter to be annotated explicitly). For both TA and SWA, 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. This restriction ensures decidability and efficiency of model checking in the case of TA and permits efficient over-approximate analysis of SWA.

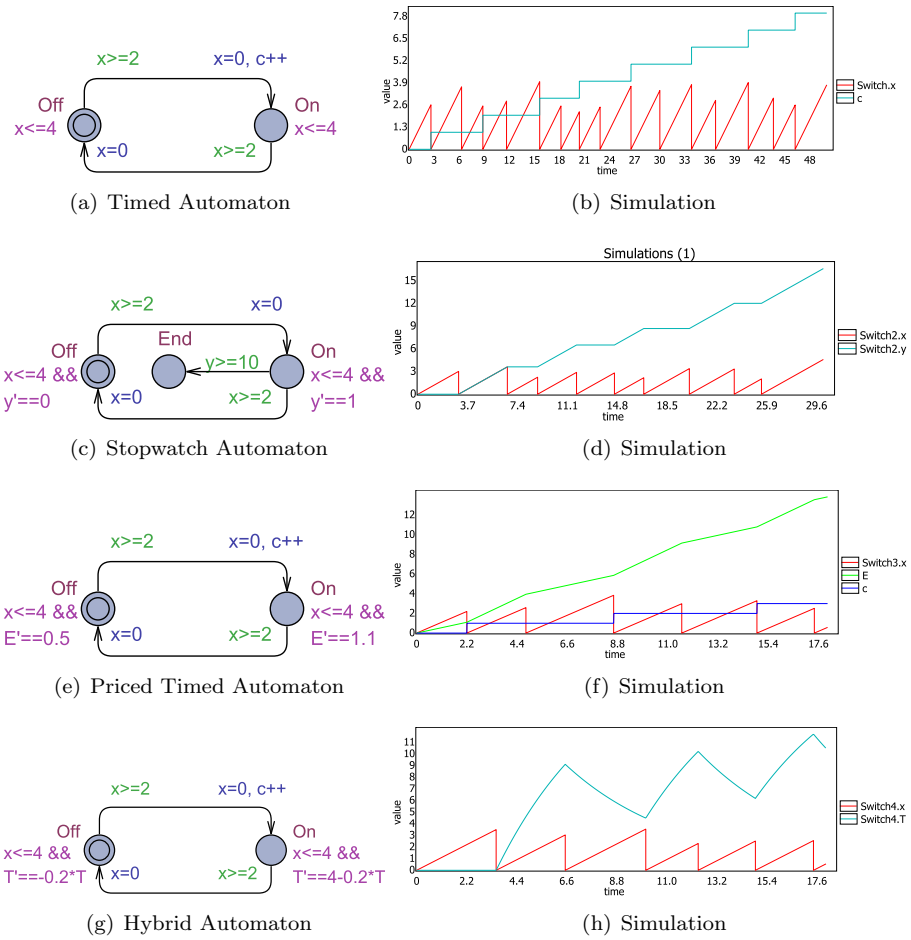
For *priced timed automata* (PTA) [7, 2, 30], the continuous variables are either simple clocks as in TA or cost-variables for which the delay update is given by an explicit rate $x' = e$ appearing in the invariant of ℓ, where e again is an expression only depending on the discrete part of the current state. PTA guards, updates and invariants may only refer to discrete part or simple clocks – thus the cost-variables cannot affect the behavior of the models but are simple observers. Under these restrictions, cost-optimal (minimal or maximal) reachability is decidable and may be computed exactly and efficiently using symbolic techniques [30].

In the most general case of a hybrid automaton (HA), the delay function F may need to solve a set of ODEs. It is important to note that in specifying the delay function F and the invariant I, the full syntax of Uppaal expressions – including user-defined functions – is at the disposal. For this class of models only simulation-based techniques are supported.

The colors used in different figures throughout this paper are the Uppaal patterns as follows: `blue` statements are clocks reset and variables update; `green` statements are transition guards; `pink` statements are location invariants; and `brown` statements are location names.

Example 4.1. *The various extended automata of Fig.3 model various quantitative aspects of a simple* `Switch` *with two modes* `On` *and* `Off`*. Fig. 3(a) is a timed automaton model of the* `Switch` *using a clock* `x` *to enforce that the time-separation between mode-switches is between 2 and 4 time-units. In addition an integer variable* `c` *counts the number of times the* `Switch` *has been in location* `On`*. Using the model checker of* Uppaal *it can be verified that the total time until* `c` *becomes 3 is between 10 and 20 time-units as confirmed by the simulation in Fig. 3(b).*

Fig. 3(c) introduces a stopwatch `y` *which is running only in location* `On`*, thus effectively measuring the accumulated residence-time in* `On`*. Using the over-approximate verification offered by* Uppaal *for stopwatch automata, it can be concluded that within 11 time-units the* `Switch` *cannot have been in* `On` *for more than an accumulated time of 10 time-units. This is confirmed by the simulation in Fig. 3(d).*



(a) Timed Automaton (b) Simulation

(c) Stopwatch Automaton (d) Simulation

(e) Priced Timed Automaton (f) Simulation

(g) Hybrid Automaton (h) Simulation

Figure 3: Timed, Stopwatch, Priced and Hybrid Automata for Switch

Fig. 3(e) is a priced timed automaton model of the `Switch` *with a (single) cost-variable* `E` *measuring the total accumulated energy consumption during the behavior. Here the rate of* `E` *is 0.5 in the* `Off` *location and 1.1 in the* `On` *location. The most energy-efficient way of having the counter variable* `c` *reaching 3 is* 7.4*. This could also have been calculated using the cost-optimal scheduling algorithm of* Uppaal CORA*, but* Uppaal CORA *can only handle integer values on the cost rates so the values would have to be scaled up before the verification. Again this finding is confirmed by the random simulation of Fig. 3(f).*

Finally, Fig. 3(g) is a hybrid automaton model of the `Switch` *with the continuous variable* `T` *modeling the temperature. Here the invariants in the locations* `On` *and* `Off` *are simple linear differential equations describing the evolution of* `T`*. Fig. 3(h) provides a random simulation of the model. For this type of model no*

exact model checking is offered.

4.3. Stochastic Hybrid Automata

The stochastic semantics of HAs refines the non-deterministic choices that may exist with respect to delay, output and next state. For each state $s = (\ell, \nu)$ of a HA $\mathcal{A}$, we shall assume that there exist probability distributions for delays, output as well as next-state:

- the *delay density function*, μ_s over delays in $\mathbb{R}_{\geq 0}$, provides stochastic information for when the component will perform an output, thus $\int \mu_s(t)dt = 1$;
- the *output probability function* γ_s assigns probabilities for resolving what output $o \in \Sigma_o$ to generate, i.e. $\sum_o \gamma_s(o) = 1$;
- the *next-state density function* η_s^a provides stochastic information on the next state $s' = (\ell', \nu') \in \mathbb{R}^X$ given an action a, i.e. $\int_{s'} \eta_s^a(s') = 1$.

For outputs happening deterministically at an exact time point d (or deterministic next states s'), μ_s (η_s^a) becomes a Dirac delta function δ_d ($\delta_{s'}$)[2].

In Uppaal SMC, uniform distributions are applied for states where delay is bounded, and exponential distributions (with location-specified rates) are applied for the cases, where a component can remain indefinitely in a location. Also, Uppaal SMC provides syntax for assigning discrete probabilities to different outputs as well as specifying stochastic distributions on next-states (using the function `random(b)` denoting a uniform distribution on $[0, b]$).

Example 4.2. *Under the above stochastic interpretation of timed automata, all of the extended timed automata models of the* `Switch` *will have the delays in* `Off` *and* `On` *being determined by a uniform distribution on the interval* $[2, 4]$. *The various simulations illustrated are obtained using this stochastic semantics. Now using the statistical model checking engine of* Uppaal *SMC, we may establish a number of interesting performance properties. Using the timed automata model Fig. 3(a) we find that the probability that* `c` *becomes 3 before 15 time units is estimated to be in the confidence interval* $[0.419126, 0.518993]$ *with confidence 0.95 in Fig. 3 after some 402 simulation runs. Using the priced timed automaton model of the* `Switch`*, we may estimate the expected energy consumption before* `c` *becomes 3 to be in the interval* $[11.0389 - 0.34824, 11.0389 + 0.34824]$ *with confidence 0.95 within 36 runs. Finally, using the hybrid automaton model, it may be established that the probability that the temperature drops below 5 degrees after 10 time-units is in the interval* $[0.104583, 0.204489]$.

In general a model comes as a network of HAs. For networks, the stochastic semantics is based on the principle of independence between components under

[2] which should formally be treated as the limit of a sequence of delay density functions with decreasing, non-zero support around d.

the assumption of input-enabledness. Repeatedly, each component decides on its own – based on a given delay density function and output probability function – how much to delay before outputting and what output to broadcast at that moment. Obviously, in such a race between components the outcome will be determined by the component that has chosen to output after the shortest delay: the output is broadcast and all other components may consequently change state.

For more in-depth description of the semantic foundation of Uppaal SMC we refer the reader to [18]. For concrete syntax of models and queries we refer to the home-page of Uppaal.

4.4. Statistical Model Checking

Statistical Model Checking (SMC) is a simulation-based analysis approach used to give a probabilistic estimate of a certain property being satisfied by a given model. SMC [13] is a widely accepted analysis technique in many research areas such as industrial applications in software engineering [5, 33] and systems biology [16].

Uppaal SMC analyses a network of timed automata and a probabilistic property specification, similar with CTL but including a probability quantifier. Differently from Model Checking, SMC returns a probability regarding a property with a specific certainty. Uppaal SMC supports five different analysis methods: *Hypothesis testing*, *Probability evaluation*, *Probability comparison*, *Expected value*, and *Simulations*. Below we use `N` to denote a natural number, `P` to denote a probability, and *expr* to denote an expression.

- **Statistical evaluation**: SMC estimates the probability of the state property being satisfied. For instance, the following query computes a probability confidence interval where simulation time is limited up to `N` time units:

  ```
  Pr[<=N](<> expr)
  ```
  (1)

- **Hypothesis testing**: SMC checks if the property is satisfied within a certain probability. For instance, the query

  ```
  Pr[<=N](<> expr) >= P
  ```
  (2)

  asks whether the probability of meeting the state property "expr" is greater than or equal to given probability value `P` while checking (simulating) the system under analysis up to `N` time units. This type of query yields less information than an estimated confidence interval above, but it is more efficient as it requires fewer simulation runs.

- **Statistical comparison**: SMC compares the satisfaction possibilities over two properties. For instance, the query can be in the form of

  ```
  Pr[<=N_1](<> expr1) >= Pr[<=N_2](<> expr2)
  ```
  (3)

- **Expected value**: SMC computes the maximal or minimal value of a certain variable while checking the system. For instance, the query

  ```
  E[<=N; M](min: expr)
  ```
  (4)

  asks what the average of the minimal values of the variable in "`expr`" is when simulating the system up to `N` time units by `M` rounds.

- **Simulations**: SMC simulates a system multiple times and computes trajectories of specified expressions over time. Query

  ```
  simulate M [<=N] {expr_1, expr_2}
  ```
  (5)

  requires UPPAAL SMC to show the values of "`expr_1`," and "`expr_2`" expressions over time when running `M` simulations up to `N` time units.

In order to estimate the probability of a property, SMC generates a number of stochastic runs and checks the property on each of the runs. The property is checked up to a certain confidence level (using confidence coefficient δ) and with a certain maximum error limit (ϵ distance from the center). Since many natural properties are monotone, the truth at length k of a run implies truth on the entire run [29], therefore we only check runs up to a certain bound of a run. In UPPAAL SMC the length of runs can be specified either as a number of discrete transitions or as a simulation time or cost bound. In our work we use a constant time bound *timeBound*. The confidence level, error limit and run length are all user parameters in our framework.

In theory, the maximum number of runs n required to achieve the needed confidence level δ and precision ϵ can be derived from Hoeffding's inequality $Pr(|\bar{p} - p| > \epsilon) \leq 2e^{-2n\epsilon^2}$[28], which says that the probability of the wrong result (when the estimated $\bar{p}$ probability differs from the real probability p by more than ϵ) is no greater than $2e^{-2n\epsilon^2}$. The probability of the wrong result is called the *level of significance* $\alpha = 1 - \delta$, and hence $n \geq -ln(\alpha/2)/(2\epsilon^2)$ runs is enough. Hoeffding's inequality implies that the number of runs is sublinear in terms of confidence and quadratic in terms of precision. Moreover, the complexity does not depend on the structure of the model, but merely on the simulation performance. Therefore, it is not prohibitively expensive to get a very high degree of confidence even on the models which are prohibitively difficult to solve analytically. In practice, we can exploit the fact that our samples follow binomial distribution and hence the probability estimation is even more efficient by using sequential methods [26], which adapt to the actual probability value and the confidence interval is computed by more precise methods [17].

Besides the statistical check of property satisfaction, UPPAAL SMC can evaluate the modeled process performance by estimating the mean value of an expression over the model variables. In this case we cannot assume any distribution, hence the value estimation is based on the Central Limit Theorem which says that the distribution of means of sufficiently large samples follows Normal distribution, while the small sample means follow Student's t-distribution [36].

Therefore the confidence interval with level δ and significance $\alpha = 1 - \delta$ is estimated using mean and t-distribution with standard error:

$$\frac{\Sigma_i^n x_i}{n} \pm t_{\alpha/2,n-1} \sqrt{\frac{\Sigma_i^n x_i^2 - (\Sigma_i^n x_i)^2/n}{n(n-1)}}$$

where x_i are the measured samples and $t_{\alpha/2,n-1}$ is the $\alpha/2$-quantile of t-distribution with $(n-1)$ degrees of freedom.

In order to estimate the mean of maximum (minimum) value over the run of an expression V, the following syntax is used: `E[time<=TimeBound; RunCount ] (max: V)`.

The size of the estimated interval depends on the variance of the measured samples, therefore there is no generic way to limit the error and hence the user has to specify the number of runs in the query (`RunCount`) while α is still the level of significance and confidence level is $\delta = 1 - \alpha$. The confidence interval can be made arbitrary tight by increasing the number of runs.

In this paper, we use both SMC and classical symbolic model checking techniques to analyze the schedulability of hierarchical scheduling systems. The UPPAAL verification suite provides both symbolic and Statistical model checking. The models which in practice can be analyzed statistically, using the UPPAAL SMC verification engine, are larger and can contain more features.

Meanwhile, SMC provides much faster responses. The speed of such responses depends entirely on the degree of certainty that one wants to obtain. The reason is that SMC consists of running a sufficiently high number of simulations of the system under analysis. The advantage of SMC resides in: 1) SMC provides a quick response in terms of less than a minute. This is also true in the case of non-schedulability were SMC produces counter-example witnesses; 2) SMC enables quantitative performance measurements instead of the Boolean (true, false) evaluation that symbolic model checking techniques provide.

5. Design of Hierarchical Scheduling Systems

In this section, after introducing the formal basis of our model-based framework we provide a conceptual description of our models and show the conformance of our framework with the classical analytical theory for analyzing the schedulability of hierarchical scheduling systems.

5.1. Classical compositional framework

This section provides the formal basis of our model-based compositional analysis approach and show that our theory conforms with the formal basis given in the compositional framework [42] for hierarchical scheduling systems.

A scheduling unit C is defined as a tuple (W, A) where W is a workload, consisting of a set of tasks $T_i = (p_i, e_i)$, and a scheduling policy A. Each task $T_i = (p_i, e_i)$ has timing attributes in the form of a period p_i and an execution

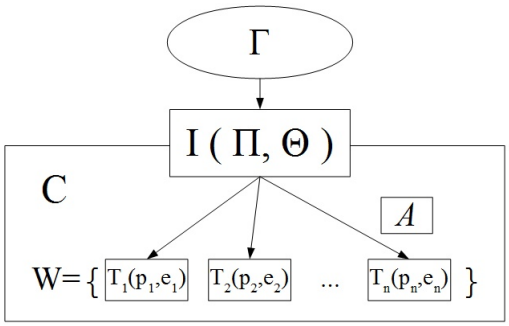


Figure 4: Component and tasks in a compositional framework

time e_i. Task deadlines are the same as periods[3]. The scheduling unit C (Fig. 4) is given a collective timing requirement $I(\Pi, \Theta)$ called interface, where Π is a period and Θ is a budget for the component. The collective timing requirement I is a representative of all timing requirements of tasks constituting the workload W. A task that is responsible for resource supplying of resources is called a resource model (Γ), which is request to satisfy an interface of its child component.

A hierarchical scheduling system is organized in a parent-child relationship in a hierarchical manner. A parent (scheduling) component has one or more tasks, each of which is connected to a child component. A child component in turn is also a scheduling unit and additionally given an interface I, which states the collective timing requirement that is requested by its workload. An interface can be viewed as a contract between a parent component and its child component in that the amount of resource described in the interface is guaranteed by the parent component. Tasks in a child component (resource-demanding) rely on the execution of a task (resource-supplying) at the corresponding parent level component in that they can execute only when that task runs. The execution of a resource-supplying task does not necessarily synchronize with the potential execution of tasks of the resource-demanding component. Hence, the total execution time of the parent task might not always be available to the tasks of the corresponding resource-demanding components.

The schedulability test for an HSS may be performed in a compositional way. Basically, a parent component is checked to see whether its tasks always provide the required amount of resources of the interface (Γ) to the child components, and each of the associated child components is checked to see whether each of its tasks always meets the deadline. In the classic compositional framework[42, 41], the schedulability is checked using both the *demand bound function* (`dbf`) and

[3]In this paper, we use the implicit deadline that is the same as the period for a given task unless a specific deadline d_i is specified for T_i.

the *supply bound function* (sbf) as follows:

$$\forall 0 < t \leq 2 * LCM_W,\ \mathtt{dbf}_A(W,t) \leq \mathtt{sbf}_\Gamma(t) \tag{6}$$

where t is a time interval and LCM_W is the least common multiplier of the periods of all the tasks.

An interface I of a resource-demanding component is characterized by the demand bound function $\mathtt{dbf}_A(W,t)$. A resource model Γ, an instance of the interface instantiated by a resource-supplying task, is characterized by the supply bound function $\mathtt{sbf}_\Gamma(t)$. For the scheduling policies EDF (Earliest Deadline First) and RM (Rate Monotonic), the demand bound functions are respectively defined by:

$$\mathtt{dbf}_{EDF}(W,t) = \sum_{T_i \in W} \left\lfloor \frac{t}{p_i} \right\rfloor \cdot e_i \tag{7}$$

$$\mathtt{dbf}_{RM}(W,t,i) = e_i + \sum_{T_k \in HP_W(i)} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k \tag{8}$$

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

As a resource model, we use Periodic Resource Model (PRM) proposed by Shin et al. [42, 41]. The PRM assigns a required amount of resources every specific period. It consists of two components, Π and Θ, which are a period and a budget, respectively. It provides child tasks with the amount Θ of resources every Π time-unit. The beginning time of its supplying resource is not prior determined, thus does not synchronize with child tasks.

For a given time interval t, the supply bound function $\mathtt{sbf}_{PRM}(t)$ of PRM is formulated as:

$$\mathtt{sbf}_{PRM}(t) = \left\lfloor \frac{t - (\Pi - \Theta)}{\Pi} \right\rfloor \cdot \Theta + \epsilon_s \tag{9}$$

$$\epsilon_s = \max\left(t - 2(\Pi - \Theta) - \Pi \left\lfloor \frac{t - (\Pi - \Theta)}{\Pi} \right\rfloor, 0\right) \tag{10}$$

where ϵ_s is the amount of resources that can gained by the last resource supply overlapped with the window of a given time interval t.

5.2. Conceptual Models of our Approach

In our model-based approach, we realize the compositional framework in the form of SWA models. We implemented the **dbf** as a set of tasks together with a scheduling algorithm, while the **sbf** is implemented by a supplier model (resource model). Our supplier model (R_{SWA}) is an implementation of the conceptual resource model Γ. The schedule when workload can use resources follows a scheduling algorithm, and a task is scheduled to use a resource and is constrained by the supplier model.

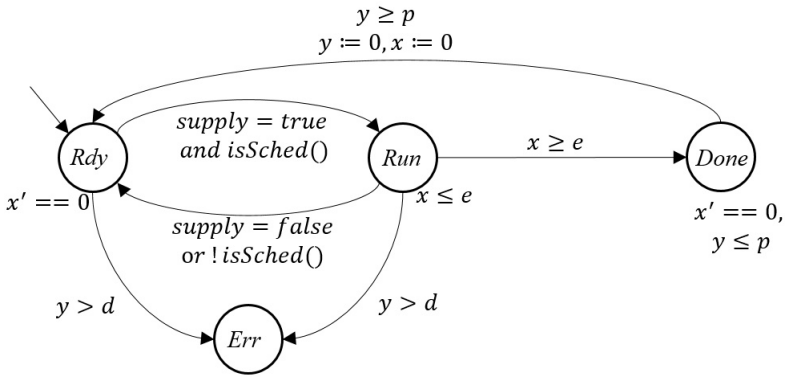


Figure 5: Task model in SWA

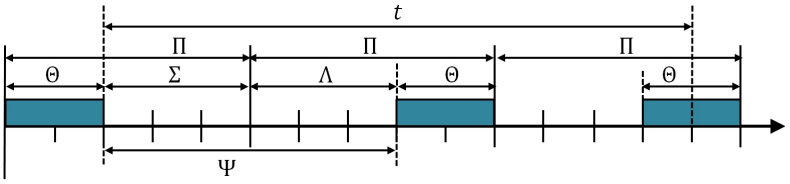


Figure 6: Resource allocations of Periodic Resource Model

SWA supports stopwatches, which are clocks that can be stopped and resumed without a reset. The modeling formalism allows for having different rates of progression for stopwatches, but we only utilize the values 1 (running) and 0 (stopped). In our SWA models, the stopwatch is used to express the preemption of a task's execution. The execution of a task is preempted, i.e. the associated clock stops, in two cases: when it is preempted by a higher priority task or when any of the needed resources is not provided by the supplier.

Fig. 5 is a conceptual model of a task, which we will realize using SWA in Section 7. The clock x stops progressing in the locations where its derivative x' is set to 0. The clock x keeps progressing at other locations. The task starts at the initial location *Rdy* and moves to *Run* when the two following conditions hold: the task is scheduled to use a resource *pid*, ($isSched(pid)$) and there is a supply of necessary resources ($supply = true$). The clock x measures the execution time of the task while it is in the location *Run*. If either of the two conditions is false at the location *Run*, the task moves back to the location *Rdy*. The task stays at location *Run* until the stopwatch x reaches the execution time e, and then jumps to location *Done* delaying until the next period. A task joins the error location *Err* when its deadline d is missed ($y > d$). Throughout this paper we keep the assumption that $e \leq d \leq p$.

In the following, we relate the analytical view of the supply bound function bound to the PRM, a resource model, with the way they are implemented as a supplier model in our approach. Fig. 6 shows the of the resource allocations of

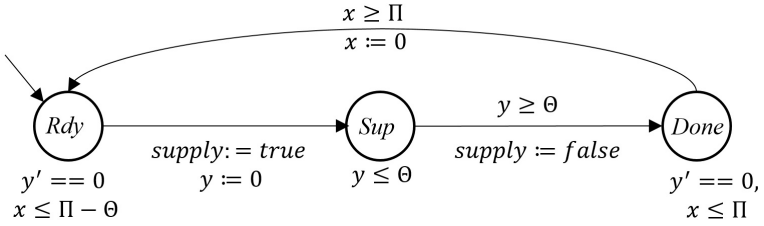


Figure 7: Conceptual PRM model in SWA notation

the PRM which guarantees the resource requirement $I(\Pi, \Theta)$ where Π is 5 and Θ is 2.

One can remark that our resource model supplies the whole budget non-preemptively in one chunk, however according to [42] if one considers only worst cases, both preemptive and non-preemptive resource models provide the same analysis results. Thus we will use a non-preemptive supplier model (Fig. 7) both in this conceptual description as well as in the computation models. Fig. 7 shows the conceptual model of SWA resource model. In this model, the variable *supply* represents the resource allocation, which is a shared variable with the task model. Thus the supply is only enabled for Θ time units within the period Π. The location *Rdy* of R_{SWA} corresponds to the delay Λ in PRM of Fig. 6. the delay Λ is a delay where a new period has started but the resource allocation has not. The location *Sup* corresponds to Θ where the resource is being allocated, and *Done* corresponds to Σ where the resource model waits for the next period.

In order to realize a compositional approach, our resource model R_{SWA} of PRM does not synchronize with the execution of tasks similarly to the resource allocation of PRM either. Thus the resource model can stay at the location *Rdy* up to $\Pi - \Theta$ or immediately move to the location *Sup*. This resource model is designed to generate all possible resource allocations including the maximum duration of no resource allocation Ψ.

6. Generic Resource Model for Multi-core Hierarchical systems

In this section, we first characterize what traits a general resource model concept would need in order to be able to specify any particular resource model, namely: urgency, preemptiveness and single/multi-core supply patterns. After introducing the different characteristics that a resource model can be specified with, we formally define the class of potential resource models that we believe can be instantiated for any relevant resource model.

In any hierarchical scheduling context, the behavior of the resource model does not depend at all on the resource demanding component, i.e. child component served by the resource model. Any resource model behaves in the way that it supplies resource for an amount of time then stops supplying for a given time interval. In that way, we define the behavior of any resource model by a transition system consisting of a sequence of transitions over a set of states, that may have different supply patterns, and at least one of the states is non-supplying.

We use resource models to describe the interface between different levels of the hierarchy, in such a way that the system can be analyzed compositionally. Accordingly, a resource model abstracts the scheduling behavior of the parent task. It describes all potential ways in which resources can be supplied to the level below it. For exactly this reason non-determinism is needed to model a concrete resource model. Classical examples of a resource model that can be instantiated from our resource model concept are Periodic Resource Model (PRM) [42] and Explicit Deadline Periodic (EDP) [24], which guarantee a certain amount of the computation time per period.

Generally, a resource of a scheduling system is characterized by the following properties:

- **Regularity**: a resource allocation may be given according to a strict period or a loosen (quasi) period.
- **Time-wise**: a resource allocation may be given according to a time schedule.
- **Event-triggered**: a resource allocation can be initiated by an event.
- **Availability**: the availability of resource at the moment may be interesting.
- **Amount**: the amount of resource to be assigned within a time bound may be interesting.

In order to make our schedulability analysis technique compositional, we add non-determinism on the assignment of resources as follows:

- **Non-deterministic preemption**: The resource assignment can be preempted at any time as long as it is not accomplished according to a resource assignment contract.

Roughly speaking, our resource model can be seen as a specialization of timed automata. We identify four different types of locations that can be part of a resource model. Each of these types has distinctively different semantic interpretation.

- **Non-urgent (non-deterministic) and preemptible resource supply (NP)**. The resource allocation can be delayed and preempted.
- **Urgent (deterministic) and non-preemptible resource supply (UN)**. The resource allocation must start immediately as soon as the corresponding state is reached. Since urgent supply cannot delay, we assume that it cannot be preempted.
- **Non-urgent and non-preemptible supply (NN)**. The start of resource allocation can be delayed, but cannot be preempted once it begins.
- **Non-supply (N)**. No resource is allocated.

In Table 1, we summarize the three supplying location types. Notice that an urgent supply location cannot be preemptible because the whole supply must be done without delay.

Table 1: Supplying location types in resource models.

	Preemptible	Non-preemptible
Urgent	-	UN
Non-urgent	NP	NN

As the resource model is quite independent from the resource demanding (child) component, one can explore the state space and show the different behaviors of the resource model regardless of the scheduling system.

Given a set of resources $\mathcal{R}$, a buffer $\mathcal{B} : \mathcal{R} \to \mathbb{N}$ is a function that specifies the amount of resource that a given resource model guarantees to provide at each supply, i.e. from each supplying state. The guaranteed resource amount will be supplied according to a supply pattern $sPattern$. In fact, the supply pattern states how the different resource units, of the resource model, collaborate to provide the whole amount of the resource guaranteed for each individual supply. For example, if only one resource unit is used then the supply time of that unit must be equal to the resource amount guaranteed in $\mathcal{B}$. Whereas if two resource units supply the resource in parallel, the supply time of the resource model could be half of the resource usage time specified in $\mathcal{B}$ because each unit provides half of that amount. Formally, we specify the supply patterns of each resource model by the following grammar:

$$\alpha ::= \alpha \| \alpha \mid \alpha + \alpha \mid r$$

whereas $r \in \mathcal{R}$ is a resource unit, that could be for example a core of a multi-core execution platform. Resource units can be used in a strict parallel mode ($\alpha \| \alpha$) and choice mode ($\alpha + \alpha$). Using the choice pattern, only one resource unit is non-deterministically selected to supply the resource. The strict parallel pattern states that the resource units are used to supply resource simultaneously. In the individual resource pattern (r), only one individual resource unit is used to supply the resource. The resource units could be heterogeneous in case of any supply pattern, except for the choice pattern where we assume that both units are homogeneous.

The resource model we are introducing is formally specified by the following:

Definition 2 (Resource model (R)). *A resource model R is a tuple $(L, l_0, X, I, \mathcal{R}, locType, \mathcal{B}, sPattern, \to)$ where:*

- *L, l_0, X and I are the same as for hybrid automata given in Section. 4.2,*
- *$\mathcal{R}$ is a set of resource units,*
- *$locType : L \to \{NP, UN, NN, N\}$ gives the type of each location.*

- $\mathcal{B} : L \setminus \{l \mid locType(l) = N\} \rightarrow \mathbb{N}$ *is a buffer function that associates to each supplying location the current amount of resource to be supplied at that location.*
- $sPattern : L \setminus \{l \mid locType(l) = N\} \rightarrow \alpha$ *states the supply patterns used at each supplying location,*
- $\rightarrow$: $L \times \mathcal{G}(X) \times \Lambda \times A \times L$ *is the transition relation where* $\mathcal{G}(X)$ *is the set of guard predicates over* X*,* Λ *is a set of events and* A *is a set of actions.*

Given a time duration x and a supply pattern α of a supplying location, we characterize the amount of resource that minimally will be provided according to the supply pattern α during x time units by $x \otimes \alpha$ given recursively as follows:

$$x \otimes \alpha = \begin{cases} x & if\ \alpha ::= r \\ x \otimes \alpha_1 + x \otimes \alpha_2 & if\ \alpha ::= \alpha_1 \| \alpha_2 \\ min(x \otimes \alpha_1, x \otimes \alpha_2) & if\ \alpha ::= \alpha_1 + \alpha_2 \end{cases}$$

So that for a time duration x, the resource amount provided by a single resource unit r is x, whereas the amount provided by two strictly parallel units $\alpha_1 \| \alpha_2$ is the sum of the resource amounts provided by both units. The resource amount for the choice pattern $\alpha_1 + \alpha_2$ is given by the minimum of the resource amounts that can be provided by both resource units over the same time duration x.

The *interface requirement* (Γ) is a collective timing requirement that all tasks in C_{demand} require to be satisfied. Meanwhile, a resource model is viewed as a specific design that satisfies a requirement. Thus, the interface requires a resource model to supply a resource no matter how it will be allocated. For instance, if an interface requirement adopting the parameters of PRM, (period, executionTime), is (10,3) meaning that 3 time units of resources are required every 10 time units by the demander component, all the resource models in Fig. 10 satisfies this requirement.

6.1. Graphical Representation of the Generic Resource Model

Fig. 8 shows the graphical notations we use to represent the different types of locations. Each of the locations has been distinguished by a specific shape and provided with a label. The resource model is then obtained by linking different locations to each other via transitions. Locations as well as transitions can be associated with timing constraints.

Fig. 9 shows an example of our resource model designed as a transition system. Such an example consists of 3 supplying states, having different supply patterns, and 1 non-supplying locations. Basically, the example is a resource model that provides 2 computation CPUs. The resource model starts at the location S_2, where $cpu1$ is available for 15 time units within 25 time units. $cpu1$ can be preempted at any time, but it is guaranteed that $cpu1$ can be usable for 15 time units within 25 time units. The location S_3 guarantees that the two resources $cpu1$ and $cpu2$ are available for 10 time units within 15 time

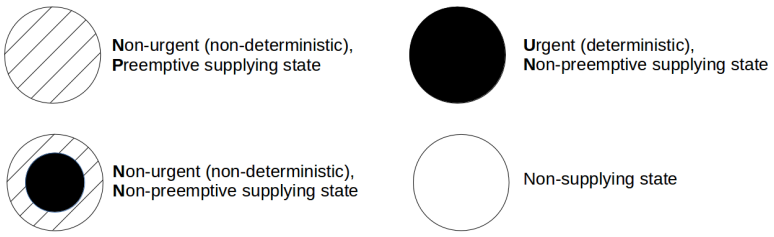


Figure 8: Graphical representation of the typed locations

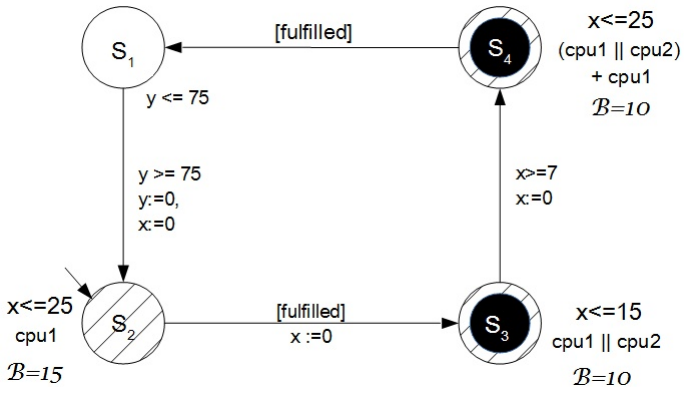


Figure 9: Example of a resource model

units, where the resources can never preempted once they are occupied by tasks relevant to this resource model. At the location S_4, there are two possibilities for tasks: to use two resources $cpu1$ and $cpu2$ or to use one resource $cpu1$. The location S_4 specifies that either the two resources ($cpu1$ and $cpu2$)or the one resource ($cpu1$) are available for 10 time units within 25 time units. Similar with the location S_3, the resources at the location S_4 are never preempted once they have been occupied.

6.2. Instantiation for Classical Resource Models

In this section, we show how classical resource models, PRM and EDP, can be instantiated form our generic resource description. The Periodic Resource Model (PRM) [42] can simply be obtained from our generic resource model by just providing only one supplying location (NP, UN or NN) having a single supply pattern ($\alpha ::= r$) together with a non-supply location. The time spent at the supplying location each period is constrained by the budget of the PRM, specified as a location invariant. Examples of the instantiation for PRM are depicted in Fig. 10(a) and Fig. 10(b).

For the Explicit Deadline Periodic (EDP) resource model [24], which is a PRM with deadline, the invariant of the supplying location must be constrained

with the given deadline as depicted in Fig. 10(c). An instantiation for PRM with an urgent non-preemptible supply is illustrated in Fig. 10(d).

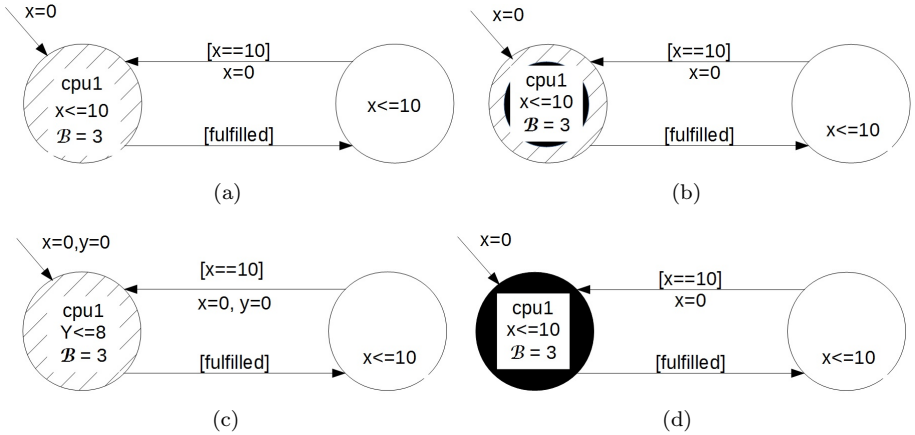


Figure 10: Instantiation for periodic resource models PRM and EDP.

Fig. 10(a) specifies that `cpu1` is available for 3 time units every 10 time units and the assignment is possible at any time before the period of 10 time units expires. In the same way, Fig. 10(b) specifies that the same behavior as Fig. 10(a) but the assignment is non-preemptive once it begins. Fig. 10(c) specifies a deadline of the resource assignment. Once the resource model begins the assignment of `cpu1`, the assignment of 3 time units should be over no later than 8 time units. Finally, Fig. 10(d) states that the assignment of the resource must start as soon as a new period begins and it is non-preemptive.

7. Model-based Compositional Analysis Framework

Our compositional analysis framework accomplishes two primary purposes: budget estimation and schedulability proof. The budget estimation is to guess the minimum budget of the interface that makes a component schedulable for a given period and a set of real-time tasks. Schedulability proof is to verify that the component is schedulable w.r.t. the estimated budget. The compositional framework, adopting a model-based approach as shown in Figure 11, consists of three types of concurrent process models: resource (supplier) model, task model, and scheduler model.

The task model together with the scheduler model constitute a general scheduling system model. The resource (supplier) model simulates the behavior of the interface of a component in hierarchical scheduling systems: it stops and resumes the running of scheduled tasks according to the availability of resources, that can be preempted by other higher priority components. In other words, a task in a component can stop running even though it is scheduled to use the resource by a local scheduler because the resource is not available by intervening

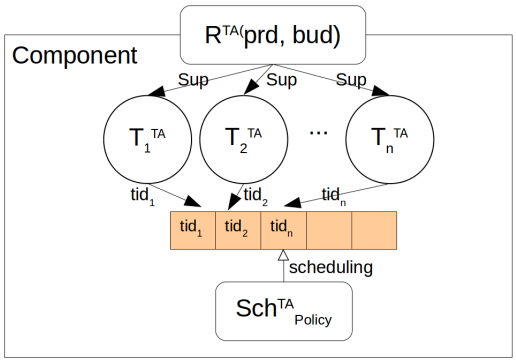


Figure 11: Model-based compositional framework

of higher priority components. The resource supplier model stops and resumes the running of a task according to the interface requirements. We assume that the supplier process and task process are not synchronous so as to check the schedulability of the task under the most pessimistic resource supply scenario [42]. How to estimate the minimal budget of the interface and verifying the budget for a component is described later in this section.

7.1. Scheduling System Model

In this section, we present the UPPAAL models we use to describe the scheduling system entities: tasks, schedulers and resource models. We also provide a way to estimate the minimum budget of each component.

7.1.1. Task Model

The task model in this paper has various execution attributes, such as best case execution time, worst case execution time, deadline, initial offset and regular offset. Thus, our framework can easily be used to describe complicated hierarchical scheduling systems. Formally, a workload W consists of a set of tasks $\{T_1, T_2, ..., T_n\}$, and individual task T_i is characterized by

- `pri`: Task priority.
- `ioffset`: The offset of the initial period of the task.
- `poffset`: The offset from the beginning of each period until the task release.
- `bcet`: Best-case execution time.
- `wcet`: Worst-case execution time.
- `preemptable`: Whether a task is preemptible.
- `tid`: Task identifier.

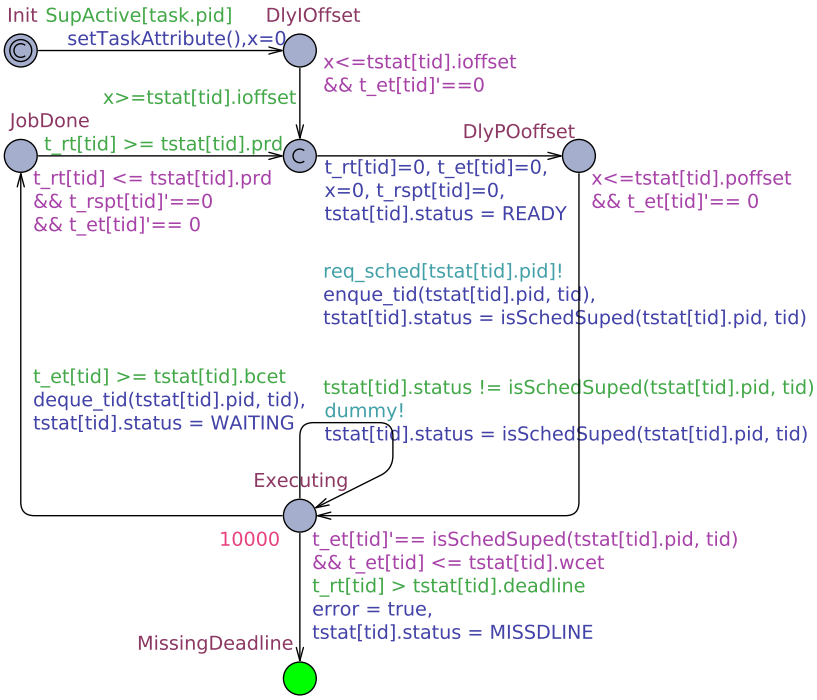


Figure 12: Task model

Listing 1: isSchedSuped()

```
bool isSchedSuped(pid_t pid, tid_arr tid) {
      return (rq[pid].qmem[1]==tid && isSupply[pid]);
}
```

Figure 12 shows the SWA task template. It begins by waiting, non-deterministically, for an amount of time up to the initial offset (`ioffset`). Using this parameter, we can adjust the synchronicity of the task execution with the supplier. This will be further explained in Section 8.

The clock `t_et` is a stopwatch that represents the execution time of a task utilizing resources. Thus, `t_et` runs and resumes according to some conditions. Using this stopwatch mechanism, we model the preemption of tasks in SWA formalism. Meanwhile, the variable `t_rt` is a typical clock that keeps running in order to keep track of the elapsed time since the task begins the job. The stopwatch `t_rspt` is used to measure the worst-case response time of the task. The behavior of the task model consists mainly in checking whether a resource is available or not, by checking the supplier status `isSupply[pid]`, which is done inside the function `isSchedSuped()`. This function is represented by the following description:

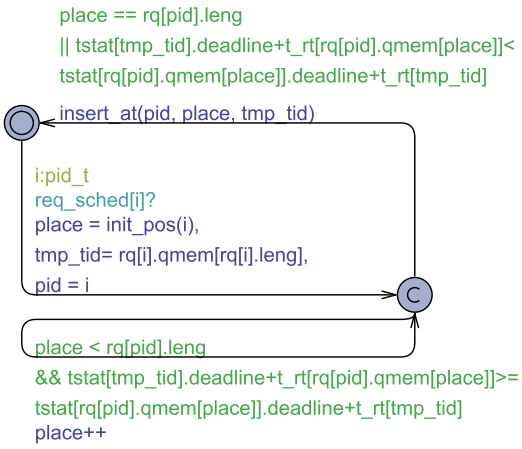


Figure 13: EDF scheduler

Notice that the function `isSchedSuped()`, shown in Listing 1, returns true if the task id `tid` is at the first element of the queue, i.e. the task is scheduled, and the corresponding supplier is currently active.

The execution of a task can always be suspended whenever the supplier stops providing the task's requested resource. The task execution can also be preempted by another task in the same component. The location `Executing` represents both *Running* and *Ready* status because the clock `t_et` can stop and resume according to both conditions concerning the availability of the resource supply and whether the task is currently scheduled or not. If the conditions hold, the clock `t_et` runs in order to measure the execution time elapsed; otherwise, the clock `t_et` stops but `t_rt` and `t_rspt` still are running to measure the elapsed time since the task has began the job.

7.1.2. EDF Scheduler

The EDF scheduler is formulated as shown in Figure 13. Basically, the EDF scheduler maintains tasks identities in its ready queue according to the ascending order of the slack times, the remaining time of the ready tasks to invidious deadlines. In Figure 13, the comparison of the stack times of a newly inserted tasks and the remaining tasks in the ready queue is operated on the recursive transition. The deadline and running time of a new ready task denoted by `tmp_tid` are referred as to `tstat[tmp_tid].deadline` and `tstat[tmp_tid].t_rt`, respectively. The deduction of `tstat[tmp_tid].t_rt` from `tstat[tmp_tid].deadline` is compared with that of `tstat[rp[pid].qmem[place].t_rt` from `tstat[rp[pid].qmem[place].deadline` of the other ready tasks (UPPAAL does not allow the deduction between clocks, thus the deduction is expressed as the plus by the inverse of the inequality).

Listing 2: Structure of resource ready queue

```
typedef struct {
  int[0,tid_n+1]   leng;
  tid_arr          qmem[tid_t];
} queue_t;
```

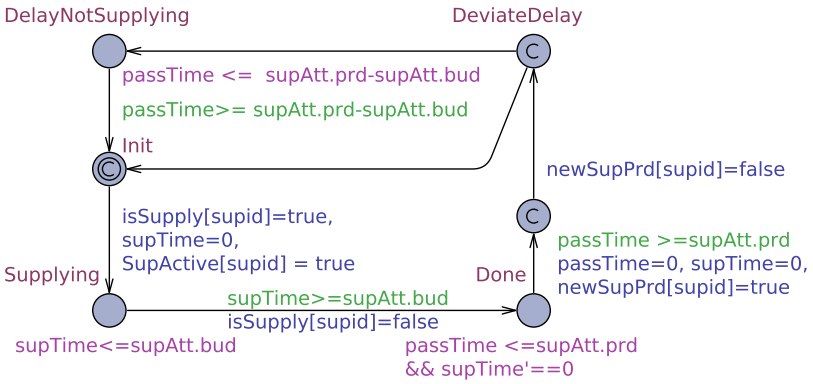


Figure 14: Periodic Resource Model

7.2. Resource Model

When a task is ready it inserts its id into the ready queue of a resource. A scheduler prioritizes the tasks in the ready queue of each resource according to a scheduling policy. The ready queue is implemented by the following structure:

The resource scheduler in our model manipulates the order of task's ids in the ready queue in order that the ids become in the same order task priorities that are determined according to a scheduling policy. Whenever a task requires the scheduler to assign a resource, the scheduler finds a position using a scheduling algorithm where the task is to be placed.

The models used are available at:
`http://people.cs.aau.dk/~ulrik/submissions/881641/SCP2014.zip`. The top level system is formed by a parallel composition of component suppliers together with a scheduling policy. The schedulability of the top level system is performed according to [11]. The SWA models of scheduling algorithms are not included in the paper because their behaviors are trivial, but they are provided in the above link.

The resource model that we use in the followings follows the Period Resource Model (PRM) [43], which is represented by two parameters: period and budget. The PRM model can be modeled as shown Figure 10(a) in our graphical notation for resource models.

The authors of [43] present a framework where the resource allocation by the supplier does not necessarily synchronize with the periods of tasks. That is, if a workload (set of tasks) starts at time t_w and the resource allocation begins at time t_r the authors assume that t_w is not necessarily equal to t_r. This

assumption leads to a resource supplier model where the supply of resources is non-deterministic within the resource period. Thus, we assume that the supply of budget is fulfilled *at any time* before the resource period elapses.

Figure 14 shows the SWA template of the resource model adopting PRM. The resource model communicates with the other templates through a shared variable (`isSuply`), which is also used in the template `Task` to keep track of the resource supply. A clock `passTime` keeps track of the time since the resource model has begun its new resource supplying period. A clock `supTime` keeps the resource-supplying time.

Our resource model is assumed to be non-preemptible, thus it supplies the full amount of resource at once. It is consistent with the approach of the PRM[43] that assumes the worst-case of resource supplying by resource model to pursue the safety of schedulability.

The initial location `Init` of the template is marked with double circles. Such a location is also marked with a "`c`" indicating that it is a committed location, this leads the supplier to move instantaneously to the next location (`Supplying`). Thus, the resource model begins supplying the resource as soon as it starts, by setting `isSupply[supid]` to true, and stays at location `Supplying` until that the resource supplied time (`supTime`) becomes equal to the budget (`supAtt.bud`). Then, it joins the location `Done` and waits for next period. The resource model can delay the supply up to the slack time, which is the maximum amount of time that can elapse before the supplier starts to supply resources. The slack time is written as `supAtt.prd−supAtt.bud` in model and specified at location `DelayNotSupplying` and as a guard on the edge leaving that location. The resource model at location `DeviateDelay` can choose to delay up to slack time or immediately re-start the resource supply as soon as a new period begins. The most pessimistic case of the resource supply occurs, as Ψ in Figure 6, where a period begins the resource supply simultaneously at its beginning and the following period delays the supply until its slack time, i.e. the supply finishes at the end of the period.

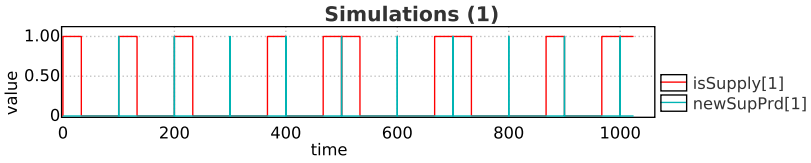


Figure 15: Behavior of PRM

Figure 15 shows a resource supplying pattern of our SWA PRM model. The separation between the third and fourth periods is the most pessimistic case of resource supplying where tasks can most possibly miss the deadline.

7.3. Automated computation of the resource budget for interface

We have automated a technique for directly estimating the supplier budget that makes the corresponding component schedulable. Such an automation is

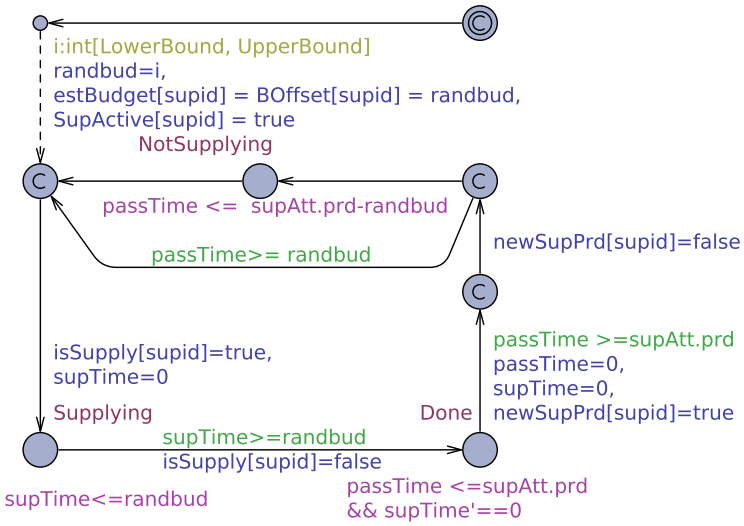


Figure 16: Modified resource model

realized by adding a plot generator template to the system and exploiting the expressiveness of the UPPAAL SMC query language, which collaborates with a plot generator template which stores randomly selected budgets in a stopwatch clock variable. The plot generator template helps UPPAAL SMC display the random budgets that lead the system to miss its deadline to identify the minimum budget that leads the system to be schedulable.

Figure 16 shows the modified initial location of the `Supplier` template depicted in Figure 14. From the initial location to the `Supplying` location, we add a new commit location. On the incoming edge to the location, a temporary budget randomly picked up from between two constants `LowerBound` and `UpperBound` according to the uniform distribution is set to `randbud`, which is noticed by other components via `estBudget[supid]` and `BOffset[supid]`. the system checks the schedulability of a scheduling system with the random budget of `randbud`. The minimal budget for a given component can be found by repeatedly checking the schedulability with the given random budgets and discarding the random budget values which lead the system to miss a deadline. To this end, we use the following query:

```
Pr[estBudget[1] <=runTime] (<> globalTime>=runTime and error)
```
(11)

where `estBudget[1]` is the budget candidate for the supplier in a given run, `randbud` is a constant value that is larger than any of the potential budgets, and `globalTime` is the current simulation time (clock). In the helper template, `estBudget[1]` is assigned a value larger than `randbud` when the simulation has executed for `runTime` time units. Thus, this query finds every number between 0 and the supplier's budget for which UPPAAL SMC finds a run where a task misses its deadline before the expiry of `runTime`, i.e. `globalTime>=runTime`.

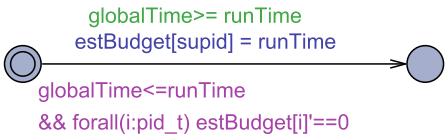


Figure 17: Plot generator

The above formula collaborates with a plot generator of Figure 17 for Uppaal SMC to produce a plot that displays all the random budgets that lead the system to miss its the deadline.

The plot generator of Figure 17 runs up to `runTime`, and the stopwatch clock `estBudget[supid]` keeps track of `estBudget[supid]` which records the values of `randbud` of Figure 16. Notice that the above formula and the plot generator of Figure 17 share the stopwatch clock `estBudget[supid]`, and its values are displayed on the x-axis of Figure 18(a) and 18(b).

Figure 18(a) and 18(b) show the probability distributions of budgets that Uppaal SMC produces after checking the system using the query (11). Figure 18(a) shows that for every potential budget between 25 and 46, as indicated by Span of displayed sample [25, 46] in Figure 18(a), runs where a deadline has been missed were found out of 214 simulations. In other words, a budget greater than 47 *can* make the system schedulable. The X-axis values are the budget candidates whereas the Y-axis data are the probabilities of component S_2 to be unschedulable under the corresponding budgets. Figure 18(b) shows the case of using FP scheduling algorithm, where the runs missing deadlines are shown over within [25, 47] and that indicates that 48 is the minimum budget that might lead the system to be schedulable.

Notice that we set 25 to the lower bound of budgets below which all budgets are assumed to lead the system to be unsatisfiable.

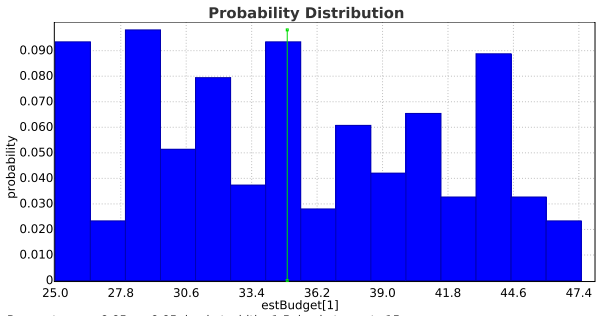
Given a budget, the schedulability of a component workload can be checked using the following query:

```
Pr[<=runTime] (<> error)                                    (12)
```

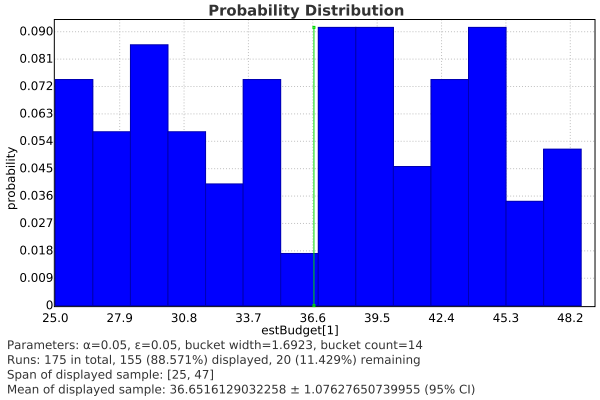
Such a query computes the probability of a component to finally (`<>`) reach an error, where `runTime` is a simulation time and `error` is a global variable indicating whether a task has missed its deadline or not.

Table 2 shows the budgets we have computed for different component configurations as well the time spent when validating our results using the Uppaal model checker.

By using the budget found via query (11) as a parameter value for component S_2 of Table 2 when checking the query (12), we can see that this indeed makes component S_2 schedulable under EDF. Figure 18(b) is the estimation results of the same component under RM policy, showing that a supplier budget greater than 47 can make component S_2 schedulable. By using query (11), we can obtain a very good estimate for the minimal budget. In practice, one might



(a) EDF



(b) RM

Figure 18: Probability distribution of supplier's budgets that make component S_2 of Table 2 non schedulable under EDF and RM

still need to check two or three values using query (12) after having applied query (11). Once a candidate budget is strongly determined, we apply symbolic model checking to be absolutely certain.

7.4. Schedulability Analysis Using the Generic Multicore Resource

A scheduling algorithm for multi-core systems determines multiple highest priority tasks that can occupy cores, and a resource model determines the availability of cores over time according to a resource supply contract or plan.

Figure 9 shows a resource model that manipulates multiple processors whereas Figure 19 provides the UPPAAL template that instantiates this resource model. Each of the locations S_2 and S_3 of Figure 9 is implemented by two locations: `S21` and `S22`, respectively `S31` and `S32` in Figure 19.The availability of processors is given by variables `isSupply[pid]` where `pid` is a CPU Id. If a variable is set to 1, the CPU corresponding to `pid` is available. The clock `x` of Figure 9

Table 2: Model checking performance (second). The budget inside () is for RM. I represents an interface, p is a period, b is a budget, T is a task, and e is an execution time.

$Comp.id$	$I(p, b)$	$T(p, e)$	EDF	RM
C_1	(100, 33 (43)))	T_1(250,40) T_2(400,50)	0.59	0.04
C_2	(150, 48 (48))	T_1(250,40) T_2(750,50)	0.89	0.01
C_3	(100, 34 (39))	T_1(250,40) T_2(450,50) T_2(750,35)	7.53	0.01

corresponds to the clock x in Figure 19. The clock `w` is the resource supplying time at the supplying locations, which comes along with $\mathcal{B}$ in Figure 9.

The two locations `S21` and `S22`, modeling the non-urgent and preemptive location S_2 of Figure 9, are toggled denoting that $cpu1$ is usable at location `S21` but it is not usable at location `S22`. The availability of $cpu1$ is communicated by manipulating the global variable `isSupply[1]`. When the CPU is available, clock `w` keeps running in order to measure the elapsed time. If the location `S21` is being visited for 15 time units, i.e. the clock `w` reaches 15 time units, the resource model moves to location `S31`.

The locations `S31` and `S32` correspond to the location S_3 of Figure 9, which is a non-urgent and non-preemptive resource supplying state. In these locations, 10 time units of CPU resource is guaranteed to be available within an interval of 15 time units. Particularly, the resource supply cannot be preempted once it begins. The Uppaal resource model in Figure 19 captures a non-urgent and non-preemptive resource supplying state in the following way: the corresponding resource model may stay at location `S31` up to 5 time units, but it can leave at any time before the expiry of such a duration (5 time units). When it joins `S32`, the resource model makes two resources $cpu1$ and $cpu2$ available by setting `isSupply[1]` and `isSupply[2]` to 1. The resource model keeps both CPUs available until the guaranteed resource amount is fully provided, i.e. clock `w` reaches 10 time units.

As an experiment of our multi-core resource model, implemented in Figure 19, we check the schedulability of the following task set:

$$T_1(170, 15),\ T_2(350, 20),\ T_3(470, 45),\ T_4(650, 57)$$

To simplify, it is assumed that the resource model applies a global fixed priority scheduling algorithm to assign tasks to the available cores. First, the light-weight analysis, Uppaal SMC, using the query 12 returns the following result:

```
(149 runs) Pr(<> ...) in [0,0.0199048] with confidence 0.95.
```

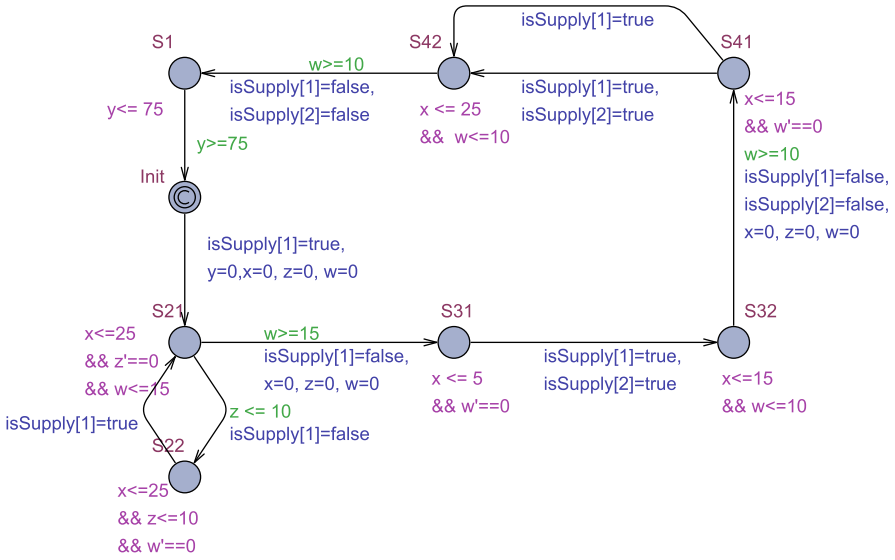


Figure 19: UPPAAL template of Figure 9

This means that, with 95% confidence, there is no deadline miss within the simulation time. To acquire 100% certainty of the schedulability property, we apply UPPAAL MC technique with the following query:

```
A[] not error                                                    (13)
```

The result returned by UPPAAL MC, the following, confirms the response issued by UPPAAL SMC:

```
A[]( not error )
Verification/kernel/elapsed time used: 30,508.266s / 10.11s / 45.216s.
...
Property is satisfied.
```

Therefore, scalability would not be a challenge if the SMC findings are positive, and only very few cases (feasible configurations) from these findings will be analyzed using symbolic model checking. Further results regarding the scalability are discussed in Section 8.4.

8. Enhancement of Resource Utilization

This section presents two techniques for enhancing the utilization of a resource: 1) introducing a new supplier model; 2) making tasks more synchronous with their suppliers by adjusting tasks initial offset.

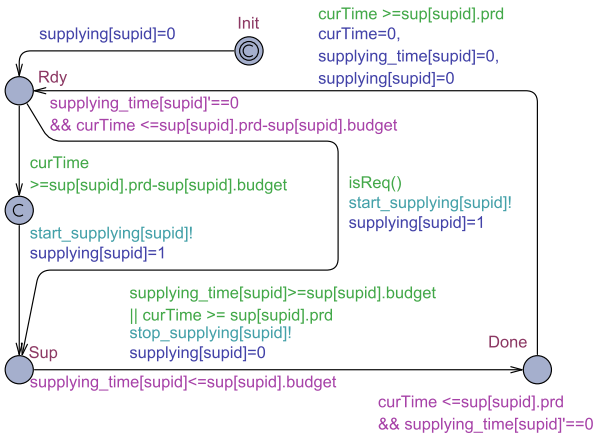


Figure 20: Synchronous Periodic Resource Model in SWA

8.1. Synchronous Periodic Resource Model

In order to increase the resource utilization by trying to avoid supplying resource when it is not needed, i.e. no waiting task, we introduce a new supplier model. The new supplier relies on delaying the resource supply, while no task is requesting resource, until a task request is received. Such a delay is up to the component slack time (period-budget).

Fig. 20 depicts the SWA template that implements our new supplier model. Once started, the supplier joins location `Rdy` and keeps waiting while the slack time is not expired. Such a constraint is implemented by the location invariant `curTime <= sup[supid].prd−sup[supid].budget`, where `prd` and `budget` are respectively the period and budget of the supplier. One can remark that, at location `Rdy`, the stopwatch measuring the resource supply is not progressing (`supplying_time[supid]'==0`). Non deterministically, the supplier moves from location `Rdy` to the location `Sup` by either receiving a task request (guard `isReq()` over the crooked edge), or once the slack time is expired (guard `curTime >=sup[supid].prd−sup[supid].budget` over the vertical edge).

At location `Sup`, the supplier keeps supplying resource before moving to the location `Done`. Such a location can be reached once the whole budget is supplied. From location `Done` and once the period is expired, the supplier joins location `Rdy` to start new period and resets its clocks.

Table 3 shows the gain in resource utilization obtained when applying the new supplier model. At the first stage, using the periodic resource model PRM, we compute the component budgets of the avionics system we mentioned earlier. The component budgets obtained via CARTS (2^{nd} column) and UPPAAL SMC (3^{rd} column) are identical; (10,6) (20,6) (20,2) (20,10). By replacing PRM with our new supplier model, we recompute the minimum budgets making the avionics components schedulable using UPPAAL SMC (4^{th} column). For components Nav.Radar Ctrl and Navigation, the budgets are decreased to 5 in each case, with a

Table 3: Resource utilization comparison

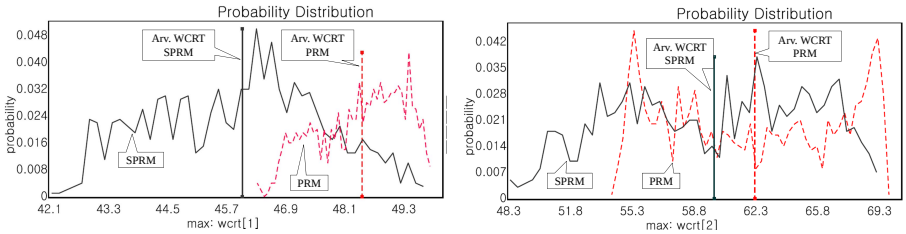
Analysis Tool Resource Model	CARTS PRM	SMC PRM	SMC & MC Synchronous PRM
Nav. Radar Ctrl	(10, 6)	(10, 6)	**(10, 5)**
Navigation	(20, 6)	(20, 6)	**(20, 5)**
Radar Ctrl	(20, 2)	(20, 2)	(20, 2)
Control & Display	(20, 10)	(20, 10)	(20, 10)

gain of 17% thanks to our new supplier model. We have checked and confirmed such new budgets using the UPPAAL symbolic model checking (MC).

In order to evaluate the effects of introducing a new supplier model, we have made a statistical experiment using the two tasks in the component Navigation. Fig. 21 shows the probability distributions of WCRT using two different resource models. This shows that the average WCRT is enhanced using the new supplier model SPRM. There is no significant difference in the actual WCRT. By using these plots we can see how much a hierarchical system can be improved by using different system settings. The fact that we can easily generate such plots also shows the versatility of a model and simulation based approach.

8.2. Offset Manipulation

Our second technique consists in limiting the initial offset for the arrival of all tasks. Explicitly including offsets in the schedulability analysis was initiated in [45]. By limiting this initial offset to a certain percentage of the supplier period, the given component can be schedulable with a lower budget. This is an assumption that we are making about the system. It is the responsibility of the system engineers to confirm that the offset that they chose actually conforms with the real system. Thus we are not computing optimal offsets making the system schedulable, but investigating the impact of different offsets on the individual component resource requirements. As shown in Figure 25, the smallest supplier budget can be obtained if all tasks arrive exactly synchronously



(a) Flight data (b) Steering

Figure 21: Probability distributions of WCRT of flight data and steering tasks. The queries `E[<=100000;1000](max:wcrt[1])` and `E[<=100000;1000](max:wcrt[2])` are used to generate the probability distributions using UPPAAL SMC.

with the start of the supplier period. This could be hard to achieve in practice. On the other hand, we think that it is indeed very possible to make the tasks synchronized with the supplier period such that all tasks arrive within either the first 20% or 50% of the supplier period. For these realistic values, we still obtain significant savings in the budget that a given component needs in order to be schedulable (See columns 6 to 9 of the table depicted in Figure 25). The percentage value is a parameter that can be easily changed in our setting when checking the schedulability. Similarly to Table 5, all statistical results in Figure 25 are found using a confidence level of 0.95.

Another observation that we have made is that, the length of the period of the supplier can have a great impact on the budget that a component needs in order to be schedulable. This can be seen in Figure 25 for component S_5. We have analyzed the same component with two different supplier periods. The first period is not a common divisor of the task periods (50000), while the second supplier period (10000) is a common divisor of the task periods. For the first experiment, the component can be schedulable with 30% of the complete system resources, while in the second case it can be schedulable using only 18.8% of the system resources (see column 4) under EDF. In fact, this observation is an experimental result that can be found using both our approach and the CARTS tool (see Table 5 for component S_4).

8.3. Confirming Uppaal *SMC results with model checking*

In order to give absolute responses about the schedulability analysis performed using Uppaal SMC, we have verified some of the Uppaal SMC results by means of symbolic model checking. These are marked in Figure 25 by a gray background color in the cells. The reason for only verifying some of our results, but not all, is that for some of the models the verification time is as much as a couple of days.

According to our experience, statistical model checking is a good way to deal with the undecidability challenge of symbolic model checking in schedulability analysis, but does not represent an alternative.

8.4. Scalability Evaluation

In order to evaluate how our approach scales we run a number of model-checking queries on a safe task model and an unsafe one (with deadline violation) using a varying number of tasks. The safe models use the same periodic CPU supplier of 100ms with a period of 150ms and the task model becomes unsafe if the supply is reduced to 90ms (using the same period). The experiment starts with two tasks: 1) 250ms period and deadline and 80ms of execution time; 2) 750ms period and deadline and 160ms of execution time. The model with three tasks is created by splitting the first task into two tasks with halved execution time – this way we know that the task model can still be scheduled, because the new task can simply be scheduled after its sister task has finished. The four-task model is created by splitting another task and so on until we have all models up to 6 tasks. The model is then evaluated using the following queries:

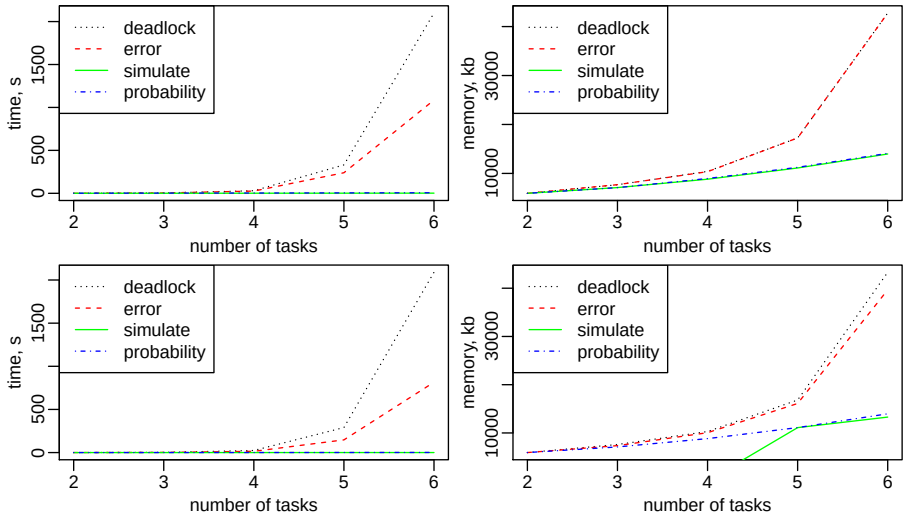


Figure 22: UPPAAL performance: safe system at the top and unsafe system at the bottom.

Deadlock validates that the task model is free of deadlocks: `A[] not deadlock`. All task models satisfy this query.

Error verifies that the task model is schedulable by ensuring that none of the deadlines are violated: `A[] not error`. The safe models satisfy this query, but the unsafe models yield a symbolic counter-example trace (which might be spurious due to stop-watch over-approximation).

Simulate validates that the error can indeed be reachable by a concrete trace by running 100 stochastic simulations until an error is found: `simulate 100 [<=runTime] { error } : 1 : error`. The safe models fail to produce any trace and the unsafe models yield a concrete trace to error as a definite proof that the system fails to fulfill its deadline.

Probability estimates the probability of a deadline violation within a time bound: `Pr[<=10000](<> error)`. The safe models fail to find a single trace and hence yield that the probability is less than 1%, whereas the unsafe models hit an error every time and hence conclude that the probability is greater than 99%.

The time and memory consumed by the checking queries are shown in Fig. 22. The plots show that symbolic model-checking requires exponential amount of resources in terms of number of tasks, whereas statistical queries take fixed amount of time and the memory consumed is proportional to the number of tasks. Both deadlock and error checks require entire state space exploration hence they are the most expensive ones, and the error check is slightly cheaper as it is faster to check the variable value than analyze that the state has no successors. The unsafe system checks are cheaper because the check terminates earlier when it finds an error and the simulation queries are cheap in particular.

Table 4: Supplier and task periods and their least common multiple (LCM)

Scale	Case	Supplier	T1	T2	T3	T4	T5	T6	LCM
Linear	1	100	100	100	100	100	100	100	100
	2	100	100	100	100	200	200	200	200
	3	100	100	100	150	150	300	300	300
	4	80	80	100	100	200	200	400	400
	5	100	50	100	125	250	250	500	500
	6	100	50	100	120	150	300	600	600
Double	1	100	100	100	100	100	100	100	100
	2	40	40	40	40	50	50	50	200
	3	40	50	50	80	80	100	100	400
	4	40	40	50	50	100	100	160	800
	5	50	50	64	64	80	100	160	1600
	6	50	50	64	80	100	128	160	3200

The state space size of a periodic system also depends on the size of its hyper-period, thus the verification effort is sensitive to concrete task periods. The system hyper-period is at least as large as the least common multiple (LCM) of the individual task periods. The following experiment examines the verification effort by increasing the LCM of the task periods while keeping the individual periods in a similar range. Table 4 shows six-task system instances where LCM is increased linearly and doubly. The task execution times are picked at random, wheres the supplier budget is kept tight but still enough for the system to be schedulable using earliest deadline first scheduler.

Figure 23 shows the time and memory used by Uppaal to verify instances described in Table 4. Unsurprisingly the SMC queries (simulation and probability estimation) still take a fixed amount of effort across all instances. The validation and schedulability (deadlock and error) checking requires a similar amount of resources when the LCM is increased linearly and the CPU time usage is increased at most twice when the LCM is being doubled. Thus the verification resources are surprisingly manageable even if the LCM is being doubled. However schedulability checking still suffers from explosion if prime numbers are chosen as task periods (LCM becomes exponential in terms of distinct periods). For example, it takes 26.7 hours to model-check Case 1 if periods are increased from 100 to mere 101 (with LCM being 10100) – that is more than tenfold more amount of time than the successful effort, whereas the LCM has increased only by 3 times.

9. Evaluation and Comparison

In order to evaluate the correctness of our model-based approach, we compare the component budgets from our estimation to the budgets obtained by the CARTS tool [39] for the same hierarchical system configurations. The comparison is depicted in Table 5, where bold data state a difference between our results and that obtained using CARTS.

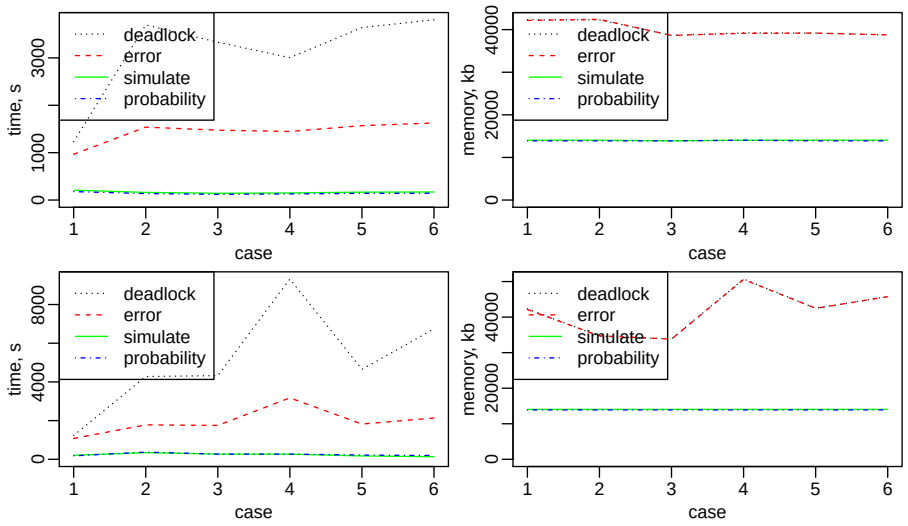


Figure 23: UPPAAL performance in terms of linear LCM (top) and double LCM (bottom).

Table 5: Comparison of the estimated budgets of CARTS and UPPAAL SMC

Comp	Task	P, WCET	CARTS		SMC & MC	
			EDF	RM	EDF	RM
S_1	T_1 T_2	500, 30 500, 100	100, 32.5	100, 32.5	**100, 33**	**100, 33**
S_2	T_1 T_2	170, 30 500, 100	100, 46.67	100, 47.5	**100, 47**	**100, 48**
S_3	T_1 T_2	250, 40 750, 50	150, 42.5	150, 42.5	**150, 45**	**150, 45**
S_4	T_1 T_2 T_3 T_4	80000, 6890 100000, 8192 200000, 2644 1000000, 5874	50000, 15082 10000, 1880	50000, 15082 10000, 2155.6	50000, 15082 **10000, 1875**	50000, 15082 **10000, 2155**

All the results presented in Table 5 are obtained with a confidence 0.95. When UPPAAL SMC returns a result where the estimated probability of missing a deadline is an interval from zero to some low value ϵ (e.g. [0,0.0973938]), this means that UPPAAL SMC did not find any trace in which a deadline was missed, i.e. with 95% confidence a deadline will not be missed with the given budget and probability distribution. If a higher confidence is needed, the confidence value can be increased and the query can be rerun.

Table 5 shows the comparison we have done with the CARTS tool. Column 1 (Comp) contains 4 different components on which we have performed the experiment. The workload of each component is stated on the second column (Tasks). In fact, each of component S_1, S_2 and S_3 is a parallel composition of 2 tasks (T_1, T_2), while S_4 contains 4 tasks (T_1,..., T_4). The third column specifies the period and the worst case execution time for each task. In order to perform a

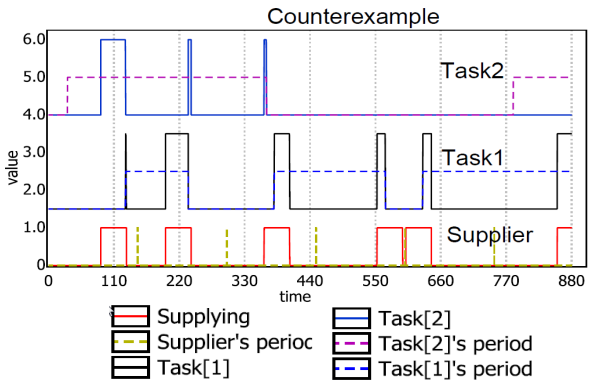


Figure 24: Counterexample for the deadline missing of T_1 in S_3 with the budget 43 under RM in Table 5

more thorough comparison, we have considered two different scheduling policies; EDF and RM. According to the CARTS tool, the minimum budget that the resource supplier should provide each 100 time units, for which the component S_1 is schedulable under EDF and RM, is **32.5**. For the same parameters, the minimum budget we have computed in our framework using UPPAAL SMC is **33**, which is very close to that obtained by CARTS. The two tools produce almost identical results. CARTS has the advantage of being an extremely fast method, while our approach is extremely flexible and configurable.

9.1. UPPAAL *SMC counterexample for one CARTS result*

During the schedulability analysis of a specific component configuration, we obtained a result from CARTS that was in conflict with our own results.

This was for the specific case (bold gray numbers) of component S_3 in Table 5. According to CARTS's computations, the minimal necessary budget for S_3 to be schedulable under EDF and RM is **42.5**. With the use of UPPAAL SMC, we first estimated the minimal budget to be 45, which has a considerable difference with the results from CARTS.

Our estimation using UPPAAL SMC immediately produced a counterexample trace showing that task (T_1) can miss its deadline with a supplier budget of 43.

The counterexample is depicted in Fig. 24 in terms of a plot that was also produced by UPPAAL SMC. The bottom of the plot shows the supplier; the dashed spikes represent the length of the supplier period and the solid line illustrates when the supplier is supplying. Each of the other two groups illustrates the behavior of a task. The solid line shows when the task is executing, and the dashed line goes up when the task is released and down when the task has finished its computation. Approximately at time 880, `Task1` is executing on its third period but fails to complete before its deadline. In order to confirm our findings, we also calculated the minimum supplier budget according to the theory underlying the CARTS tool. We calculated this both using the equations from [43] and equations from [42]. The results of such calculations confirmed

S_i	T_i	P, WCET	Δ_{init} of supplier's period							
			Δ_{init} = 100%		Δ_{init} = 50%		Δ_{init} = 20%		Δ_{init} = 0%	
			EDF	RM	EDF	RM	EDF	RM	EDF	RM
1	T_1	500, 30	100, 33	100, 33	100, 33	100, 33	**100, 29**	**100, 29**	**100, 26**	**100, 26**
	T_2	500, 100	(33%)	(33%)	(33%)	(33%)	(**29%**)	(**29%**)	(**26%**)	(**26%**)
2	T_1	170, 30	100, 47	100, 48	100, 47	100, 48	**100, 43**	**100, 45**	**100, 39**	**100, 45**
	T_2	500, 100	(46 %)	(48 %)	(47 %)	(48 %)	(**43 %**)	(**45 %**)	(**39 %**)	(**45 %**)
3	T_1	250, 40	150, 48	150, 48	150, 45	150, 45	150, 45	150, 45	**150, 41**	**150, 41**
	T_2	750, 50	(32 %)	(32 %)	(30 %)	(30 %)	(30 %)	(30 %)	(**27 %**)	(**27 %**)
4	T_1	10000, 1406	5000, 1406	5000, 1406	5000, 1407	5000, 1407	**5000, 1204**	5000, 1204	**5000, 1057**	**5000, 1057**
	T_2	40000, 2826	(28 %)	(28 %)	(28 %)	(28 %)	(**26 %**)	(28 %)	(**21 %**)	(**21 %**)
5	T_1	80000, 6890	50000, 15082	50000, 15082	50000, 15082	50000, 15082	**50000, 12531**	**50000, 14948**	**50000, 12400**	**50000, 13261**
	T_2	100000, 8192	(30 %)	(30 %)	(30 %)	(30 %)	(**25 %**)	(30 %)	(**29 %**)	(**27 %**)
	T_3	200000, 2644	10000, 1875	10000, 2155	10000, 1875	10000, 2155	10000, 1875	**10000,2060**	**10000,1872**	10000,1985
	T_4	1000000, 5874	(18.8 %)	(21.6 %)	(18.8 %)	(21.6 %)	(18.8 %)	(20.6 %)	(**18.7 %**)	(**19.9 %**)

Figure 25: Enhancement of resource usability using the maximal offset of task's initial period. The cases marked with a grey background are verified using symbolic model checking.

our findings in that we calculated the minimal budget to be 45. This leads us to conclude that there must be an error in the implementation of CARTS while the underlying theory is correct. We reported this anomaly and it has been confirmed by the developers of the CARTS tool that CARTS has an implementation error.

In this paper, we apply two approaches to expand the scalability of model checking techniques: a compositional method and statistical model checking.

A compositional approach allows narrowing down the problem of the system size to the problem of a component size of the system. In other words, instead of analyzing the whole system, it allows analyzing a component of the system and composing the analysis of individual components.

Statistical model checking, a relatively light-weight model checking technique, allows us to instantly inspect a given system by using simulation-based techniques based on the Monte-Carlo method. A scheduling system of periodic tasks without considering unexpected interrupts, as we observed, have a uniform task arrival pattern, the simulation analysis of a scheduling system can thus guarantee the correctness of the schedulability check as long as a simulation time encompasses the comprehensive macro-period [43]. Moreover, the resource model PRM that we adopt in this paper assumes the worst-case supply of resources for child component, thus we can produce more deterministic resource supply patterns that might scale down the space of the system to explore.

For the above reasons, the analysis using symbolic model checking for obtaining 100% certainty did not take much time. Except the last case of Figure 25, all the verifications finished within a few milliseconds. The following is one of the verification results for the last case in the third row of Figure 25 that we obtained from the symbolic model checking of Uppaal.

```
A[] not error
Verification/kernel/elapsed time used: 0.328s / 0s / 0.326s.
Resident/virtual memory usage peaks: 9,672KB / 31,196KB.
Property is satisfied.
```

In most cases of Figure 25, the analysis time using the symbolic model checking did not exceed much the above analysis time.

10. Conclusion

The problem that we are addressing in this paper is the configuration and schedulability analysis of hierarchical scheduling systems. We have presented a methodology that incorporates Statistical Model Checking and Symbolic Model Checking for the statistical and exact analysis of the schedulability. The analysis is compositional and highly configurable. We proposed a new generic resource model for multi-core hierarchical scheduling systems, which can capture the behavior of any classical periodic resource model. We have investigated two specific techniques for enhancing the resource utilization: a new resource model and offset manipulation.

We instantiated our approach as a framework using parameterized stopwatch automata (SWA) and the two tools Uppaal SMC and Uppaal for the analysis. However our methodology can be instantiated to fit any modeling formalism that supports both statistical and exact analysis techniques. We also provided a faster method for estimating the minimal budget of a supplier, instead of performing a binary search of potential budgets. To validate our method, we have compared our results to that of a state of the art tool. The results from the two tools are almost identical.

The main contribution of the paper is that systems, which cannot be proven schedulable using classical analytic approaches, can potentially be proven schedulable using our approach.

A perspective of this work could be a study of the impact of the two techniques, we proposed for the enhancement of resource utilization, on the systems energy efficiency [10].

References

[1] R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[2] R. Alur, S. La Torre, and G. J. Pappas. Optimal paths in weighted timed automata. In Benedetto and Sangiovanni-Vincentelli [8], pages 49–62.

[3] 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, 2003.

[4] ARINC 653. Website. https://www.arinc.com/cf/store/documentlist.cfm.

[5] A. Basu, S. Bensalem, M. Bozga, B. Delahaye, and A. Legay. Statistical abstraction and model-checking of large heterogeneous systems. *International Journal on Software Tools for Technology Transfer*, 14(1):53–72, 2012.

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

[7] G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. W. Vaandrager. Minimum-cost reachability for priced timed automata. In Benedetto and Sangiovanni-Vincentelli [8], pages 147–161.

[8] M. D. D. Benedetto and A. L. Sangiovanni-Vincentelli, editors. *Hybrid Systems: Computation and Control, 4th International Workshop, HSCC 2001, Rome, Italy, March 28-30, 2001, Proceedings*, volume 2034 of *Lecture Notes in Computer Science*. Springer, 2001.

[9] T. Bøgholm, H. Kragh-Hansen, P. Olsen, B. Thomsen, and K. G. Larsen. Model-based schedulability analysis of safety critical hard real-time java

programs. In *Proceedings of the 6th International Workshop on Java Technologies for Real-time and Embedded Systems*, JTRES '08, pages 106–114, New York, NY, USA, 2008. ACM.

[10] A. Boudjadar, A. David, J. Kim, K. Larsen, U. Nyman, and A. Skou. Schedulability and energy efficiency for multi-core hierarchical scheduling systems. In *Proceedings of the Intl Congres on Embedded Real Time Software and Systems ERTS^2*, 2014.

[11] 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, pages 61–78. Springer International Publishing, 2013. LNCS Volume 8348.

[12] J. Boudjadar, A. David, J. Kim, K. Larsen, U. Nyman, M. Mikučionis, and A. Skou. Widening the schedulability hierarchical scheduling systems. In *Proceedings of Formal Aspects of Component Software (FACS), 2014*, volume LNCS 8997, 2015.

[13] P. E. Bulychev, A. David, K. G. Larsen, M. Mikučionis, D. B. Poulsen, A. Legay, and Z. Wang. UPPAAL-SMC: Statistical model checking for priced timed automata. In H. Wiklicky and M. Massink, editors, *QAPL*, volume 85 of *EPTCS*, pages 1–16, 2012.

[14] 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.

[15] 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.

[16] E. Clarke, J. Faeder, C. Langmead, L. Harris, S. Jha, and A. Legay. Statistical model checking in biolab: Applications to the automated analysis of t-cell receptor signaling pathway. In M. Heiner and A. Uhrmacher, editors, *Computational Methods in Systems Biology*, volume 5307 of *Lecture Notes in Computer Science*, pages 231–250. Springer Berlin Heidelberg, 2008.

[17] C. J. Clopper and E. S. Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):404–413, 1934.

[18] A. David, D. Du, K. G. Larsen, A. Legay, M. Mikučionis, D. B. Poulsen, and S. Sedwards. Statistical model checking for stochastic hybrid systems. In E. Bartocci and L. Bortolussi, editors, *HSB*, volume 92 of *EPTCS*, pages 122–136, 2012.

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

[20] A. David, K. G. Larsen, A. Legay, M. Mikučionis, 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.

[21] A. David, K. G. Larsen, A. Legay, M. Mikučionis, and Z. Wang. Time for statistical model checking of real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 349–355. Springer, 2011.

[22] R. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. In *Real-Time Systems Symposium, 2005. RTSS 2005. 26th IEEE International*, pages 10 pp.–398, Dec 2005.

[23] Z. Deng and J. W. s. Liu. Scheduling real-time applications in an open environment. In *in Proceedings of the 18th RTSS*, pages 308–319. Society Press, 1997.

[24] 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.

[25] A. Easwaran, M. Anand, I. Lee, L. T. X. Phan, and O. Sokolsky. Simulation relations, interface complexity, and resource optimality for real-time hierachical systems, 2009.

[26] J. Frey. Fixed-width sequential confidence intervals for a proportion. *The American Statistician*, 64(3):242–249, 2010.

[27] T. Henzinger. The theory of hybrid automata. In *Logic in Computer Science, 1996. LICS '96. Proceedings., Eleventh Annual IEEE Symposium on*, pages 278–292, Jul 1996.

[28] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[29] T. Hrault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate probabilistic model checking. In B. Steffen and G. Levi, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 2937 of *Lecture Notes in Computer Science*, pages 73–84. Springer Berlin Heidelberg, 2004.

[30] K. G. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, T. Hune, P. Pettersson, and J. Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In G. Berry, H. Comon, and A. Finkel, editors, *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 493–505. Springer, 2001.

[31] 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.

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

[33] J. Martins, A. Platzer, and J. Leite. Statistical model checking for distributed probabilistic-control hybrid automata with smart grid applications. In S. Qin and Z. Qiu, editors, *Formal Methods and Software Engineering*, volume 6991 of *Lecture Notes in Computer Science*, pages 131–146. Springer Berlin Heidelberg, 2011.

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

[35] A. K. Mok, X. A. Feng, and D. Chen. Resource partition for real-time systems. In *Proceedings of RTAS '01*, pages 75–84. IEEE Computer Society, 2001.

[36] D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons Edition, 2006.

[37] ORIS. Oris tool website. http://www.oris-tool.org/.

[38] 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.

[39] 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.

[40] M. Åsberg, T. Nolte, and P. Pettersson. Prototyping and code synthesis of hierarchically scheduled systems using TIMES. *Journal of Convergence (Consumer Electronics)*, 1(1):77–86, December 2010.

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

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

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

[44] Y. Sun, G. Lipari, R. Soulat, L. Fribourg, and N. Markey. Component-based analysis of hierarchical scheduling using linear hybrid automata. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on*, pages 1–10, Aug 2014.

[45] K. Tindell. *Adding time-offsets to schedulability analysis*. University of York, Department of Computer Science, 1994.