A Model-Based Framework for the Specification and Analysis of Hierarchical Scheduling Systems

Mounir Chadli, Jin Hyun Kim, Axel Legay, Louis-Marie Traonouez, Stefan Naujokat, Bernhard Steffen, Kim Guldstrand Larsen

▶ **To cite this version:**

Mounir Chadli, Jin Hyun Kim, Axel Legay, Louis-Marie Traonouez, Stefan Naujokat, et al.. A Model-Based Framework for the Specification and Analysis of Hierarchical Scheduling Systems. FMICS-AVoCS, Sep 2016, Pise, Italy. pp.133 - 141, 10.1007/978-3-319-45943-1_9 . hal-01241681v2

HAL Id: hal-01241681
https://hal.science/hal-01241681v2

Submitted on 25 Oct 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Model-Based Framework for the Specification and Analysis of Hierarchical Scheduling Systems

Mounir Chadli[1], Jin Hyun Kim[1], Axel Legay[1], Louis-Marie Traonouez[1], Stefan Naujokat[2], Bernhard Steffen[2], and Kim G. Larsen[3]

[1] Inria Rennes, France
{firstname.lastname}@inria.fr
[2] Technische Universität Dortmund, Germany
{stefan.naujokat,steffen}@cs.tu-dortmund.de
[3] Aalborg University, Denmark
kgl@cs.aau.dk

Abstract. Over the years, schedulability of Cyber-Physical Systems (CPS) has mainly been performed by analytical methods. Those techniques are known to be effective but limited to a few classes of scheduling policies. In a series of recent work, we have shown that schedulability analysis of CPS could be performed with a model-based approach and extensions of verification tools such as UPPAAL. One of our main contribution has been to show that such models are flexible enough to embed various types of scheduling policies that go beyond those in the scope of analytical tools. In this paper, we go one step further and show how our formalism can be extended to account for stochastic information, such as sporadic tasks whose attributes depend on the hardware domain.

1 Introduction

Cyber-Physical Systems (CPS) are software-implemented control systems that control physical objects in the real world. These systems are being increasingly used in many critical systems, such as avionics and automotive systems. They are now integrated into high performance platforms, with shared resources. This motivates the development of efficient design and verification methodologies to assess the correctness of CPS. Among the panoply of existing techniques to respond to these challenges, one distinguishes between those that rely on an analytic approach, using tools like CARTS [10], from those that rely on formal models and tools such as Uppaal [2], or SpaceEx [8].

In this paper, we mainly focus on schedulability for CPS. Over the years, schedulability has mostly been performed by analytical methods [12]. Those techniques are known to be effective but limited to specific classes of scheduling policies and systems. In a series of recent work, we have shown that schedulability analysis of CPS could be performed with a model-based approach and extension of verification tools such as Uppaal. One of our main contribution has been to show that such models are flexible enough to embed various types of scheduling policies that go beyond those in the scope of analytical tools. In addition, we

proposed a hierarchical approach that allows us to reduce the complexity of this computation [3,4]. This approach is well-suited to perform worst-case analysis of scheduling systems, and even average performance analysis via a stochastic extension of timed automata. This extension allows us to make hypothesis about time at which tasks are scheduled. However, high-performance hardware architectures, as well as advanced software architectures, have more unpredictable behaviors. This makes the verification of these real-time systems much harder, in particular the schedulability analysis, that is essential to evaluate the safety and reliability of mission-critical systems. For this reason, designers are still reluctant to use lower-price hardware components with higher capabilities, such as multi-core processors, for these mission-critical systems.

In this paper, we propose a stochastic extension of our scheduling framework that allows us to capture tasks whose real-time attributes, such as deadline, execution time or period, are also characterized by probability distributions. This is particularly useful to describe mixed-critical systems and make assumptions on the hardware domain. These systems combine hard real-time periodic tasks, with soft real-time sporadic tasks. Classical scheduling techniques can only reason about worst-case analysis of these systems, and therefore always return pessimistic results. Using tasks with stochastic period we can better quantify the occurrence of theses tasks. Similarly, using stochastic deadlines we can relax timing requirements. Finally stochastic execution times model the variation of the computation time needed by the tasks. These distributions can be sampled from executions or simulations of the system, or set as requirements from the specifications. For instance in avionics, display components will have lower criticality. They can include sporadic tasks generated by users requests. Average user demand will be efficiently modelled with a probability distribution. Timing executions may vary due to the content being display and can be measured from the system. This formal verification framework is embedded in a graphical high-level modeling tool developed with the Cinco meta tooling suite [9]. It is available at `http://cinco.scce.info/applications/`.

2 Background

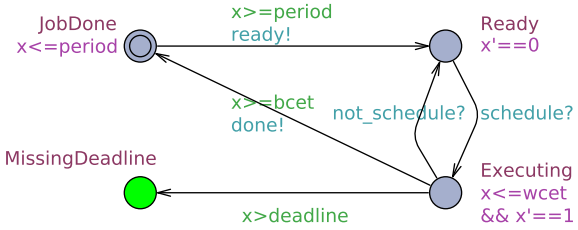
Given a set of clocks C, a function $v : C \to \mathbb{R}_{\geq 0}$ is called a *clock valuation*. A stopwatch is a vector $s : C \to \{0, 1\}$ that distinguishes between a set of running clocks and a set of frozen clocks. For a delay $d \in \mathbb{R}_{\geq 0}$ and a stopwatch s, let $v + s \cdot d$ denotes the clock valuation assignment that maps all $x \in C$ to $v(x) + s(x) \cdot d$. A *clock constraint* on C is a finite conjunction of expressions of the form $x \sim k$ where $x \in C$, $\sim \in \{<, \leq, >, \geq\}$, and $k \in \mathbb{N}$. Let B(C) denote the set of all clock constraints on C. A clock valuation v satisfies the clock constraint $g \in \mathcal{B}(C)$, written $v \vDash g$, *iff* g holds after all the clocks in g have been replaced by their value $v(c)$.

A **Stopwatch Automata (*SWA*)**[6] is a tuple $(L, l_0, \Sigma, C, \to, I, S)$, where L is a finite set of locations, $l_0 \in L$ is the initial location, Σ is an alphabet of actions, C is a finite set of real-time clocks, $\to \subseteq L \times \mathcal{B}(C) \times \Sigma \times 2^C \times L$ is the

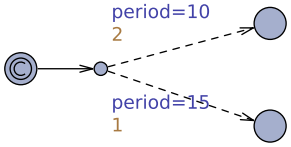
set of discrete transitions, $I : L \to \mathcal{B}(C)$ associates an invariant constraint to each location, $S : L \to \{0,1\}^C$ is the location stopwatch. A state $s = (l, v)$ of a *SWA* consists in a location l and a clock valuation v.

An execution of the *SWA* is an alternating sequence of discrete and continuous transitions. Continuous transitions update the clock valuation in a location by the same value $d \in \mathbb{R}_{\geq 0}$ for all running clocks in $S(l)$, provided that $v + S(l) \cdot d \vDash I(l)$. Discrete transitions switche from one location to another, if there exists a transition $(l, a, g, r, l') \in E$ with $v \vDash g$, and it resets the clocks in r to 0. We distinguish between internal and communication transitions. There are two types of communication transitions, the input ones (noted with ?) to receive a message and the output ones (noted with !) to send a message. As classical transition systems can do, *SWA* can be combined in networks of *SWA* by synchronizing inputs and outputs in a broadcast manner. This means that when a *SWA* executes one output, all those *SWA* that can receive it must be synchronized.

In Fig. 1a, we illustrate the concept with an abstract real-time task modeled via *SWA*. The task execution time is measured by a clock x, that can progress in location Executing (we denote $x' = 1$ the fact that the stopwatch is running) but is stopped in location Ready (denoted $x' = 0$). It starts its execution when receiving the event schedule?. It sends an event done! as soon as the clock x has reached the best case execution time (bcet) and before reaching the worst case execution time (wcet), or goes to location MissingDeadline if the clock exceeds the deadline. Finally it returns to location JobDone for the next execution round. The running task at location Executing can be preempted when receiving the event not_schedule?, in which case it returns to the location Ready.



(a) *SWA* for a simple real-time task



(b) *PTA* for a task dispatcher

Fig. 1: (Probabilistic) Stopwatch Automata

Probabilistic Timed Automata (PTA) [1] is an extension of *SWA* that adds discrete probabilities to the transitions. Thus the transition relation is replaced by $\to \subseteq L \times \mathcal{B}(C) \times \Sigma \times Dist(2^C \times L)$, where $Dist(2^C \times L)$ is a discrete probability distribution over clock reset and next location.

This extension is useful to initialize the parameters of a model with random values e.g., to specify that the period or the deadline of a task depends on some random information. The simple *PTA* in Fig. 1b allows to select two values for

the period of the task: 10 with probability 1/3 and 15 with probability 2/3. In what follows, we will call this automaton a *dispatcher*. It is worth mentioning that UPPAAL, the tool we use to model *SWA* and *PTA*, can also model global variables and parameters. Such parameters can be modified either via internal transitions, or via output transitions. In that example the period is a shared variable, initialized by the dispatcher and read by the task. This semantics of *PTA* still exhibits nondeterministic behaviors, because several probabilistic transitions may be enabled at the same time, and the time at which transitions can happen is not randomized. Therefore it can only be used to analyze best case or worst case scenario.

It is however of great interest to analyze scheduling systems with average scenario generated by a fully stochastic semantics. This will allow us to quantify performance analysis. In our early works [7,5], we have proposed a stochastic semantics for networks of priced timed automata, an extension of *SWA*. The stochastic semantics associates probability distributions on both the delays one can spend in a given state, as well as on the transitions between states. In UPPAAL uniform distributions are applied for bounded delays and exponential distributions in the case a component can remain indefinitely in a state. In a network of *PTA* the components repeatedly race against each other, i.e. they independently and stochastically decide on their own how much to delay before outputting, the "winner" being the component that chooses the minimum delay.

Model-checking queries are represented via a subset of the Computational Tree Logic (*CTL*) as defined by the model-checker UPPAAL. More precisely, we consider $\varphi ::= \mathsf{A[]}P \mid \mathsf{A<>}P \mid \mathsf{E[]}P \mid \mathsf{E<>}P$. A and E are path operators, meaning respectively "for all paths" and "there exists a path". [] and <> are state operators, meaning respectively "all states of the path" and "there exists a state in the path". P is an atomic proposition that is valid in some state. For example the formula "A[] not error" specifies that in all the paths and all the states on these paths we will never reach a state labelled as an error. For schedulability analysis, an error state is one where a task has missed a deadline.

Statistical model-checking queries require a time bound. The following query for instance "Pr[<=maxTime](<> error)" asks to compute the probability of reaching an error state before maxTime.

UPPAAL model-checker (MC) [2] is used to verify *SWA* and UPPAAL statistical model-checker (SMC) [5] is used to verify *PTA*. Moreover, if *PTA* are used with UPPAAL MC, all stochastic information is discarded and replaced by nondeterminism (probabilistic transitions are replaced by a corresponding discrete transition for each specific value of the distribution).

3 Formal Model-based Compositional Framework for Hierarchical Scheduling Systems

We first introduce the formal model used to represent scheduling units. This formalism extends the one in [3,4] with probability distributions on task's features.

Then, we show how formal tools such as Uppaal MC and Uppaal SMC can be used to solve queries such as deadlock or schedulability.

3.1 Automata-Based Models for a Scheduling unit

In our framework, a scheduling unit is composed of a set of real-time tasks, a scheduler, that implements a scheduling algorithm, and a queue, that manages jobs instantiated by tasks. Additionally we provide each scheduling unit with a resource supplier that allocates the resource (CPU time) for a given amount of time. As explained in [11] and illustrated in Section 3.2 such a resource supplier can be used to perform scheduling of complex systems in a hierarchical manner.

Tasks and stochastic dispatcher: We use two types of tasks: 1. a classical task model as presented in [4], implemented with *SWA*; 2. a new stochastic task model whose real-time attributes (period, delay, execution time) depend on a probability distributions, and are dynamically chosen by a stochastic dispatcher. This stochastic feature is of interest to model the variation of execution time with respect to the computation logics and the capability of the execution environments (CPU, memory, I/O and caches, etc). Such real values can be obtained by sampling the execution times from the real world system (and this objective is out of scope of this paper). Observe that other task's parameters, such as deadline and period, are determined according to the timing requirements of the functionality implemented by a set of tasks. For instance, some video decoder/encoder would update the deadline and period of tasks according to the frequency of input streams. For those reasons, they can also be represented by probability distributions.

Fig. 2 shows the *SWA* for the stochastic task model. From the Init location, a job is initialized with real-time attributes obtained by setTaskAttribute (and assigned by the dispatcher as explained below). This job is queued for execution at location DlyPOoffset. There it requests the scheduler to assign a CPU, which is granted by the synchronisation on the channel req_sched(pid), and reaches location Executing. Its execution can be stopped and resumed according to the availability of the CPU resource. This is implemented by a stopwatch clock t_et[tid]. The clock progresses only when the CPU is available, that is when the function isSchedSuped returns 1. Finally, the job exits from location Executing when it has completed its execution time. This releases the CPU resource using function deque_tid(tstat[tid].pid, tid). The *SWA* waits the end of the minimal inter-arrival time (WaitEndofMINIntv) and then waits for a new job instantiated by the stochastic dispatcher (JobWait).

The stochastic dispatcher, presented in Fig. 3, configures the real-time attributes of the tasks at each individual execution round.

Scheduler: The scheduler *SWA* (Fig. 4) implements the scheduling policy of the scheduling unit. We use two types of scheduling policy: *earliest deadline first* (EDF) and *fixed priority* (FP). These schedulers synchronize with the task model on the channel req_sched.

Resource Supplier: The resource supplier is responsible for supplying a scheduling unit with the resource allocated from another scheduling unit. We

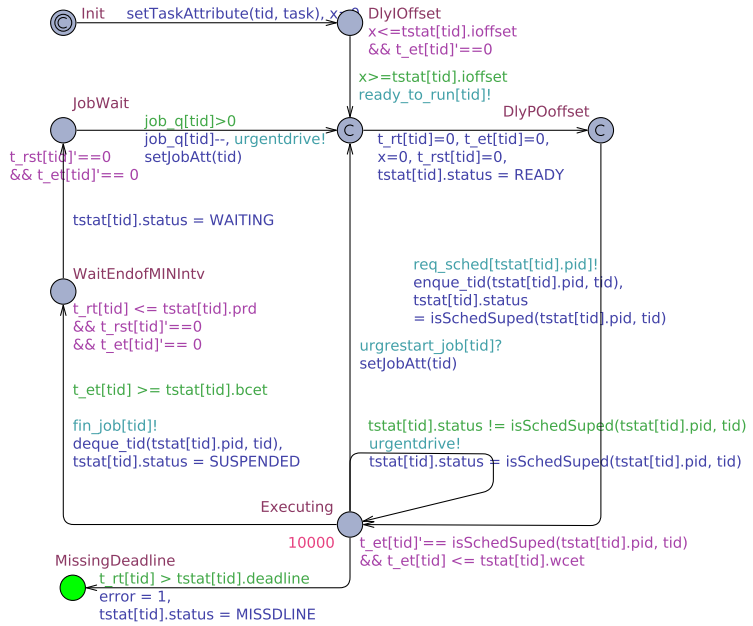


Fig. 2: *SWA* model of a stochastic task

adopt the *periodic resource model* (PRM) [11]. It supplies the resource for a duration of Θ time units every period Π. To speed up the schedulability analysis using model checking techniques, it only generates the extreme cases of resource assignment: either the resource is provided at the beginning of the period (from 0 to Θ) or at the very end (from $\Pi - \Theta$ to Π). The choice between the two assignments is non-deterministic. The PRM automata communicates with the task model through a shared variable isSupply that is set to true during the supply period. We also use the probabilistic supplier model presented in [4]. Instead of using a fixed budget Θ, it uses a range of values specified between an interval. This will allow to perform a parameter sweep with SMC by selecting uniformly a value of the budget and help determining the optimal budget.

3.2 Formal Analysis of Hierarchical Scheduling Systems

A **hierarchical scheduling system (HSS)** is a set of scheduling units organized in a tree structure. It allows to dispatch a common CPU resource to different scheduling units with the help of resource suppliers. As presented in [11], HSSs can be analyzed in a compositional manner by analyzing each scheduling units individually. Below, we present different types of real-time properties in the format of the tool Uppaal MC and Uppaal SMC.

Absence of deadlock: We check that the formal models have been correctly designed, because they cannot reach a deadlock state in which time is blocked

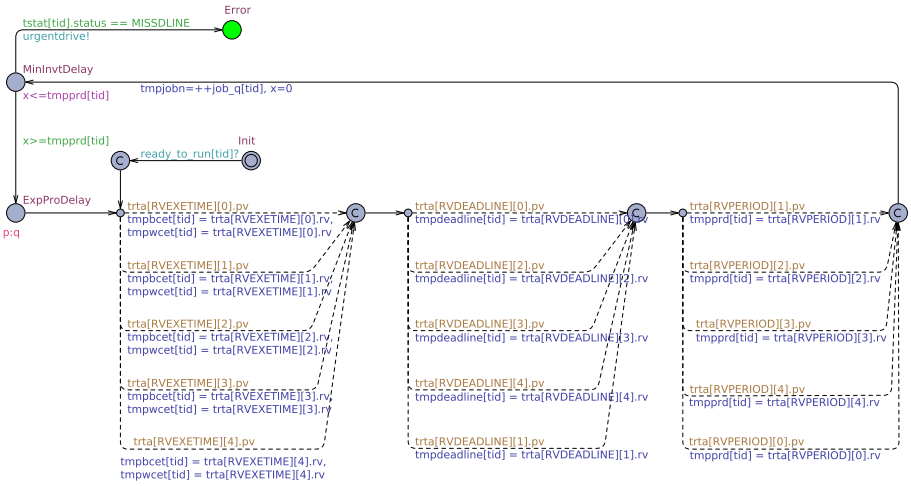


Fig. 3: *PTA* of a stochastic dispatcher that configures the three attributes with a probabilistic choice between five values.

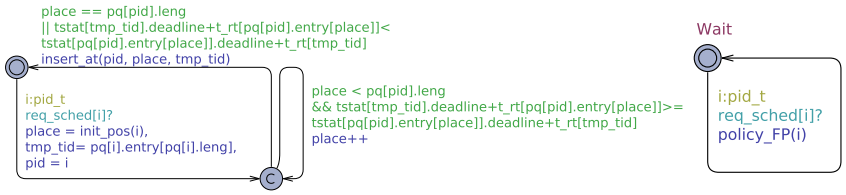


Fig. 4: EDF (left) and FP (right) schedulers

and no action is available. We use Uppaal MC to analyze this query with the following CTL formula A[] not deadlock.

Schedulability: The main objective of analyzing a scheduling unit and an HSS is to determine whether the tasks are schedulable. In our formal models we analyze this property by searching for error states in tasks, that correspond to the tasks missing their deadline. All these error states are identified by a single Boolean variable error, set to true when a task misses a deadline.

The schedulability can be analyzed by Uppaal SMC using the following probabilistic query: Pr[<=runTime](<> not error). It computes the probability that not error state is found within runTime t.u. This probability must be 1 to assert that the scheduling unit is schedulable. We can also performed the exhaustive analysis of the *SWA* model with Uppaal MC. In that case, we use the CTL formula A[] not error.

Maximum response time: Performances of the HSS are measured by analyzing the maximum response time of tasks, that is to say the maximum time between a job instantiation and its completion. We measure this property using Uppaal SMC with the following query: E[<=100000;1000](max:t_rst[2]). It asks

for 1000 simulations of 100'000 t.u. and it computes the average value over these simulations of the maximum response time of the task with ID 2 (the response time of task 2 is measured in the variable t_rst[2]).

Budget estimation: We use the probabilistic supplier to specify a range of values between an interval for the budget Θ. Then we can use UPPAAL SMC to randomly select a value within this range and check whether the scheduling unit it schedulable with this value. We use the following probabilistic query: Pr[estBudget[1]<=runTime](<>globalTime>=runTime and error). It computes the probability distribution of all the possible budget values that are not schedulable. By looking at the support of this distribution we can determine the minimum budget whose probability is zero, that is the minimum budget necessary to schedule all the tasks of the scheduling unit.

Simulate queries: Additional parameters of the model can be observe during random simulations with UPPAAL SMC. Results can be displayed in a graph.

References

1. Beauquier, D.: On probabilistic timed automata. Theor. Comput. Sci. 292(1), 65–84 (Jan 2003)
2. Behrmann, G., David, A., Larsen, K.G., Håkansson, J., Pettersson, P., Yi, W., Hendriks, M.: UPPAAL 4.0. In: QEST. pp. 125–126 (2006)
3. Boudjadar, A., David, A., Kim, J., Larsen, K., Mikucionis, M., Nyman, U., Skou, A.: Hierarchical scheduling framework based on compositional analysis using uppaal. In: FACS. LNCS, vol. 8348, pp. 61–78. Springer (2013)
4. Boudjadar, A., David, A., Kim, J., Larsen, K., Mikučionis, M., Nyman, U., Skou, A.: Widening the schedulability of hierarchical scheduling systems. In: FACS, LNCS, vol. 8997, pp. 209–227. Springer (2015)
5. Bulychev, P.E., David, A., Larsen, K.G., Mikucionis, M., Poulsen, D.B., Legay, A., Wang, Z.: UPPAAL-SMC: statistical model checking for priced timed automata. In: QAPL. EPTCS, vol. 85, pp. 1–16 (2012)
6. Cassez, F., Larsen, K.G.: The impressive power of stopwatches. In: CONCUR. LNCS, vol. 1877, pp. 138–152. Springer (2000)
7. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., van Vliet, J., Wang, Z.: Statistical model checking for networks of priced timed automata. In: FORMATS. LNCS, vol. 6919, pp. 80–96. Springer (2011)
8. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: CAV. LNCS, Springer (2011)
9. Naujokat, S., Lybecait, M., Kopetzki, D., Steffen, B.: CINCO: A Simplicity-Driven Approach to Full Generation of Domain-Specific Graphical Modeling Tools (2016), to appear
10. Phan, L.T.X., Lee, J., Easwaran, A., Ramaswamy, V., Chen, S., Lee, I., Sokolsky, O.: CARTS: a tool for compositional analysis of real-time systems. SIGBED Rev. 8(1), 62–63 (Mar 2011)
11. Shin, I., Lee, I.: Periodic resource model for compositional real-time guarantees. In: RTSS. pp. 2–13. IEEE Computer Society (2003)
12. Shin, I., Lee, I.: Compositional real-time scheduling framework with periodic model. ACM Trans. Embedded Comput. Syst. 7(3) (2008)