High-level Frameworks for the Specification and Verification of Scheduling Problems

Mounir Chadli, Jin H Kim, Kim G. Larsen, Axel Legay, Stefan Naujokat, Bernhard Steffen, Louis-Marie Traonouez

Noname manuscript No.
(will be inserted by the editor)

High-level Frameworks for the Specification and Verification of Scheduling Problems

Mounir Chadli · Jin H. Kim · Kim G. Larsen · Axel Legay · Stefan Naujokat · Bernhard Steffen · Louis-Marie Traonouez

the date of receipt and acceptance should be inserted later

Abstract Over the years, schedulability of Cyber-Physical Systems (CPS) has mainly been performed by analytical methods. These 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 contributions has been to show that such models are flexible enough to embed various types of scheduling policies, which goes beyond those in the scope of analytical tools.

However, the specification of scheduling problems with model-based approaches requires a substantial modeling effort, and a deep understanding of the techniques employed in order to understand their results. In this paper we propose simplicity-driven high-level specification and verification frameworks for various scheduling problems. These frameworks consist of graphical and user-friendly languages for describing scheduling problems. The high-level specifications are then automatically translated to formal models, and results are transformed back into the comprehensible model view. To construct these frameworks we exploit a meta-modeling approach based on the tool generator Cinco.

Additionally we propose in this paper two new techniques for scheduling analysis. The first performs runtime monitoring using the CUSUM algorithm to detect alarming change in the system. The second performs optimization using efficient statistical techniques. We illustrate our frameworks and techniques on two case studies.

Keywords Scheduling · Energy · Hierarchical scheduling · Formal methods · Statistical model-checking · High-level language · Meta-modeling

M. Chadli
Irisa, Rennes, France
E-mail: {mounir.chadli}@irisa.fr

A. Legay, L-M. Traonouez
Inria, Rennes, France
E-mail: {firstname.lastname}@inria.fr

J.H. Kim
University of Pennsylvania, U.S.
E-mail: jhkim07@gmail.com

K.G. Larsen
Aalborg University, Denmark
E-mail: kgl@cs.aau.dk

S. Naujokat, B. Steffen
Technische Universitt Dortmund, Germany
E-mail: {stefan.naujokat,steffen}@cs.tu-dortmund.de

1 Introduction

Cyber-Physical Systems (CPS) are software-implemented control systems that control physical objects in the real world. These systems are 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.

One of the trends in developing a CPS is to integrate many heterogeneous computational components into a single platform in order to maximize the utilization of hardware resources. The components are managed to be completely partitioned, such that errors caused by one component are alienated from the other components. However, high-performance hardware architectures, as well as advanced software architectures, make it much harder to predict the behavior and the timing performances of these real-time systems – in particular the schedulability analysis, that is essential to eval-

uate 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 safety-critical systems.

In order to increase the predictability of these complicated CPS, two approaches are drawing more and more attention: the model-based approach and the probabilistic approach. On one hand, the model-based approach allows more flexibility and complexity in the system design, and it expands the scope of properties that can be analyzed on the system using automated verification techniques, such as model checking and theorem proving techniques. Timed automata (*TA*) [1] for instance is a well known formal model that can be used to perform schedulability analysis of real-time systems [20,16]. Recently, this model has been used to model sophisticated scheduling system with several hierarchical scheduling units that allows to decompose the schedulability analysis of complex CPS [8]. On the other hand, the probabilistic approach allows abstracting unknown and hardly-estimated aspects of system components. Probabilistic timed automata is an extension of timed automata with probabilities [4]. When the model is fully stochastic its behavior can be predicted by statistical methods such as statistical model-checking (*SMC*) [19]. This approach is much more efficient for analyzing complex systems, that are often intractable with exhaustive methods.

High-level frameworks Currently, many models and tools are successfully used to analyze properties of CPS. But they are domain-specific, which means they cannot easily be applied to other systems. Moreover, these models and tools require high technical knowledge about the theoretical formalisms used to design models and write properties, which most system engineers do not master. In this paper we demonstrate a flexible and formal analysis engineering approach for analyzing scheduling properties of CPS.

We first introduce new model-based frameworks to describe scheduling systems. These frameworks are based on stochastic hybrid automata to model stochastic real-time tasks, scheduling mechanisms or energy consumption. They constitute a model bank to describe complex scheduling systems, such as hierarchical scheduling or multi-processor energy aware.

We then encapsulate the formal frameworks into Cinco [34,35], a generator for domain-specific modeling tools. Cinco allows to specify the features of a graphical interface in a compact meta-model language, and it generates automatically from this meta-model specification a domain-specific analysis tool with a graphical interface. Inside this analysis tool we can specify scheduling systems and the properties they must satisfy. We can launch analysis of the properties, which generates automatically the timed automata models using the components of our model-bank, and it calls the tools Uppaal [5] and Uppaal SMC [18] to perform the analysis. This approach allows to completely hide the formal models being used from the system designer, who can concentrate on the structure and the parameters of the scheduling system.

The last challenge is to give significant feedbacks to the user in the most user friendly manner. Indeed, results of formal verification from academic tools like Uppaal can be difficult to interpret, all the more when the models used by these tools have been automatically generated. Cinco provides an API for model transformations that allows to program actions that can update the model. We have used this functionality to parse the results of the analyses output by Uppaal and to show graphically the most relevant information.

Scheduling problems We consider in this paper three major problems to solve on scheduling systems. The first problem is concerned with the correctness of the design and its performances. The main correctness property is the schedulability, whereas performance evaluations can be the measure of maximum response times or energy consumption. To solve these problems we will resort to model-checking and statistical model-checking.

The second problem is concerned with the optimization of the system. In [11] we have addressed this problem in the context of hierarchical scheduling systems, in order to determine an optimal compositional framework. In this paper we propose a new optimization technique for multi-processor scheduling systems. It determines optimal mappings from tasks to processors in order to minimize energy consumption and/or response time. To determine the minimal number of simulations on which we can based a decision, we rely on statistical tests (ANOVA and Tukey HSD). We use these tests in new optimization algorithms that perform iteratively simulations of the scheduling system, check with ANOVA and Tukey HSD if the results are significant, and output as results the optimal mappings.

The last problem is to monitor the system to detect emerging behaviors or expected events. To solve this problem we propose a runtime monitoring technique that detects if an expected event has occurred and determines the time of the occurrence. It is based on the CUSUM algorithm that we have already applied to statistical model-checking problems in [29]. Our algorithm checks a long simulation trace of the system at regular intervals against a formal property. According to the

result it updates a CUSUM likelihood ratio. When this ratio exceeds a sensitivity threshold an alarm is raised to notify that the event has occurred. In this paper we present a new implementation of the algorithm to detect variations in the energy consumption of a multi-processor scheduling system.

Results Using our meta-modeling approach we have developed two high level frameworks. One is dedicated to hierarchical scheduling systems; the other is dedicated to multi-processor energy aware scheduling systems. We have applied these frameworks on two case-studies. The first one describes an avionic scheduling system with mixed critical tasks. Using our hierarchical scheduling framework we determine if different sub-systems with various criticality can be run on the same platform. The second one is a proof of concept of a multi-processor scheduling system organized in two layers: a platform layer and an application layer. We demonstrate on this case-study our new techniques for optimizing energy consumption and change detection.

Summary of the contributions The paper is a journal extension of the short conference paper presented in [11], with several new major contributions:

- New formal models for specifying complex scheduling systems. These models extend the models for hierarchical scheduling systems presented in [11] with models for multi-processor scheduling systems and energy measure.
- Two high-level frameworks for specifying and verifying scheduling problems. Both are automatically generated using a meta-modeling approach.
- Two new techniques for solving scheduling problems. The first optimizes multi-processor scheduling systems. The second performs runtime monitoring to detect expected events.
- Two case-studies that demonstrate the high-level frameworks and the verification techniques.

Organization of the paper The rest of this paper is organized as follows: Section 2 discusses related works on scheduling problems. Section 3 presents background theory about the classes of formal models used in the paper and general analysis techniques that we apply. Section 4 introduces the different scheduling problems studied in the paper and how they are formalized using model-based approaches. Section 5 proposes techniques and new algorithms to solve the scheduling problems previously defined. Section 6 presents high level frameworks for the modelling and analysis of these scheduling problems. Section 7 and Section 8 present experiments to specify and solve scheduling problems using the high level frameworks. Section 9 concludes the paper.

2 Related Work

Our model-based analysis framework for hierarchical scheduling systems is based on [8,7,24]. Extending the models of Timed Automata (TA) and Stopwatch Automata (SWA) in [8,7], we present a model of hierarchical scheduling systems, based on the stochastic sporadic tasks of [24] but with dynamic stochastic updates of real-time attributes.

Scheduling problems with energy costs are studied in [36]. This work studies an energy-flexible flow shop scheduling problem, that is a multi-objective optimization problem whose goal is to minimize both overall completion time and global energy consumption. It employs stochastic local search techniques. We address a similar problem in our framework for multi-processor scheduling systems. Instead of execution modes and machines switch-off, the configuration options that we study are assignments of tasks to processors and we use statistical model-checking combined with the ANOVA technique to estimate energy cost and response time.

There also exists approaches that perform scheduling via timeline-based planning. The work of [13] proposes such an approach and uses timed game automata (timed automata with controllable and uncontrollable actions) to find strategies for the timeline-based planning problem. Timed games and satisfiability modulo theory are also used in [12] to solve control problems with temporal constraints. Our model-based approach is also based on extensions of timed automata but we mostly rely on statistical model-checking for finding solutions to the scheduling problems. This allows us to consider more complex scheduling systems, with sporadic tasks, hierarchical scheduling or energy constraints, that would not be solvable using exhaustive techniques such as model-checking or timed games.

Another work presented in [10] uses a logic-based approaches. The planning problem is encoded in a high level action notation modeling language [41] and then translated into linear temporal logic modulo rational arithmetic formula. In our work, we introduce new high level graphical notations for complex scheduling problems. These notations are specific to the scheduling framework being studied (either hierarchical or multi-processor scheduling systems). These domain-specific notations allow to have a simpler and more accurate description of a scheduling system than using existing formalisms. Moreover we can rely on the tool generator Cinco for easily generating a domain-specific tool that implements a graphical editor for these notations.

3 Background

We present in this section the formal models and the basic techniques used in this paper. The formal models are extensions of timed automata supported by the tools Uppaal or Uppaal SMC. They range from classical timed automata to hybrid automata and may include stochastic information. The main formal methods used to analyze these models are model-checking and statistical model-checking.

3.1 Hybrid Automata

A hybrid automaton [22] is a finite state automaton extended with continuous variables that evolve according to continuous dynamics associated to each discrete state (called a location). Let X be a set of continuous variables. A variable valuation is a function $\nu : X \to \mathbb{R}$. We write $\mathbb{R}^X$ for the set of valuations over X. Valuations over X evolve according to *delay functions* $F : \mathbb{R}_{\geq 0} \times \mathbb{R}^X \to \mathbb{R}^X$, where for a delay d and a valuation ν, $F(d, \nu)$ is a the new valuation. Delay functions are assumed to be time additive ($F(d_1, F(d_2, \nu)) = F(d_1 + d_2, \nu)$. To allow communication between these automata we assume that the set of actions Σ is partitioned into disjoint sets of input and output actions ($\Sigma = \Sigma_i \uplus \Sigma_o$).

Definition 1 A hybrid automaton (*HA*) is a tuple $\mathcal{H} = (L, l_o, X, \Sigma, E, F, I)$. L is a finite set of locations, $l_0 \in L$ is an initial location, X is a finite set of continuous variables. $\Sigma = \Sigma_i \uplus \Sigma_o \uplus \{\tau\}$ is a finite set of actions partitioned into inputs (Σ_i), outputs (Σ_o) or internal (labelled with τ). E is a finite set of edges of the form (l, g, a, ϕ, l'), where l and l' are locations (resp. the *source* and the *destination*), g is a predicate on $\mathbb{R}^X$ (called the *guard*), $a \in \Sigma$ is an action label, ϕ is a binary relation on $\mathbb{R}^X$ that defines the *clock updates*. For each location l, $F(l)$ is a delay function and $I(l)$ is an *invariant* predicate.

The semantics of $\mathcal{H}$ is a timed labeled transition system, whose states are pairs $(l, \nu) \in L \times \mathbb{R}^X$ with $\nu \models I(l)$, and whose transitions are either, delay transitions $(l, \nu) \xrightarrow{d} (l, \nu')$ with $d \in \mathbb{R}_{\geq 0}$ and $\nu' = F(l)(d, \nu)$, or, discrete transitions $(l, \nu) \xrightarrow{a} (l', \nu')$ if there is an edge $(l, g, a, \phi, l') \in E$, such that $\nu \models g$ and $\phi(\nu, \nu')$. An execution of $\mathcal{H}$ is an alternating sequence of delay and discrete transitions. As classical transition systems can do, *HA* can be combined in networks of *HA* via parallel composition. This is done by synchronizing inputs and outputs in a broadcast manner. This means that when an *HA* executes one output, all those *HA* that can receive it must be synchronized. We denote input actions with a channel name followed by **?** and output actions with the channel name followed by **!**.

The above definition deliberately left open the syntax for the delay functions F, the guards g, the update predicates ϕ and the invariants I. Their concrete definition depends on the class of hybrid automata that is considered.

Timed automata (TA) [1] is the most restrictive class of *HA* we use. In this model, continuous variables are real-time clocks that all advance at the same speed. This means that for any clock $x \in X$, the delay functions $F(l)$ defines an implicit rate $x' = 1$. Guards and invariants are defined by conjunction of simple integer bounds of the form $x \sim k$, where $\sim \in \{<, \leq, >, \geq, ==, !=\}$, and $k \in \mathbb{N}$. The update predicates ϕ only allow simple assignments of the form $x = e$, where e is an expression that only depends on the discrete part of the current state.

Stopwatch automata (SWA) [9] extend *TA* by allowing to stop and resume clocks. The rates of the variables are therefore either $x' = 1$ (for running clocks) or $x' = 0$ (for stopped clocks).

Priced timed automata (PTA) [6, 2] allow the continuous variables to be either clocks as in *TA*, or cost-variables with a rate $x' = e$, where e is an expression that only depends on the discrete part of the current state. These cost-variables cannot be used in guards, updates and invariants of the *PTA*, which implies that they cannot affect the behavior of the model.

Hybrid automata (HA) is the most general case. It allows to use ordinary differential equations to define delay functions F and invariants I.

3.2 Modelling Hybrid Automata in Uppaal

Uppaal is one the most famous tool for modelling and analyzing timed automata and their hybrid extensions. The tool has been developed for more than 20 years by a collaboration between Uppsala University in Sweden and Aalborg University in Denmark. It allows to design models that belong to one of the four classes of hybrid automata presented previously. It additionally provides many syntactic constructions that help the design of complex models. In the following of the paper we will heavily use these constructions for designing models of scheduling systems. We will succinctly explain the syntax and semantics of our models, but we cannot present in the paper the full syntax of Uppaal models,

and therefore we redirect the reader to the documentation of the tool (at `http://www.uppaal.org/`) for a more precise description. Some of the main capabilities offered by the tool are:

- *Data variables.* In addition to clocks, the tool allows to use data variables (integer, float, arrays, and structures). They can be updated during transitions, and tested in guards or invariants. Synchronization channels can also be defined in arrays.
- *Functions.* The tool allows to write functions using a syntax similar to the C language. They can be used in guards, invariants, and updates of variables. When synchronizing transitions on a channel, the update functions of all the transitions involved in the synchronization are performed.
- *Templates automata.* Hybrid automata can be defined as templates with input parameters. This allows to instantiate several automata in a model using the same template (for instance several tasks with different parameters).

In the paper we will show several examples of hybrid automata by using screen captures from automata designed in UPPAAL. In these figures the transitions have guards in green, synchronization actions in light blue (τ actions are omitted), updates in blue. Locations have a name and an invariant (possibly with clock rates) in purple.

Example 1 We present in Fig. 1 four examples of the different types of models. All these models implement a simple real-time task with various functionalities, depending on the type of model being used.

The model in Fig. 1a implements a task with no preemption using a timed automata. It has a clock x to measure the length of the period and a clock y to measure the execution time. It starts its execution when receiving the event schedule?. It sends an event done! as soon as the clock y has reached the best case execution time (bcet) and before reaching the worst case execution time (wcet). Otherwise it goes to the location MissingDeadline with an internal transition when the clock exceeds the deadline. Finally it returns to location JobDone to wait for the next execution round and it sends the signal ready! to the scheduler.

The model in Fig. 1b implements a preemptive task using a stopwatch automata. It refines the previous model with a stopwatch on clock y: the clock is stopped in location Ready (denoted y'=0), otherwise it is assumed that its execution rate is 1. The task can be preempted by the scheduler when it receives the signal not_schedule?, in which case it returns to location Ready.

The model in Fig. 1c additionally computes the energy consumed by the running task using a priced timed automata with a variable e to measure the energy. The energy can only increase in location Executing at a rate given by the constant POWER.

Finally, the model in Fig. 1d is additionally aware of the frequency FREQ at which the processor is running. This frequency defines the rate at which the task executes by setting y'==FREQ in the invariant of location Executing.

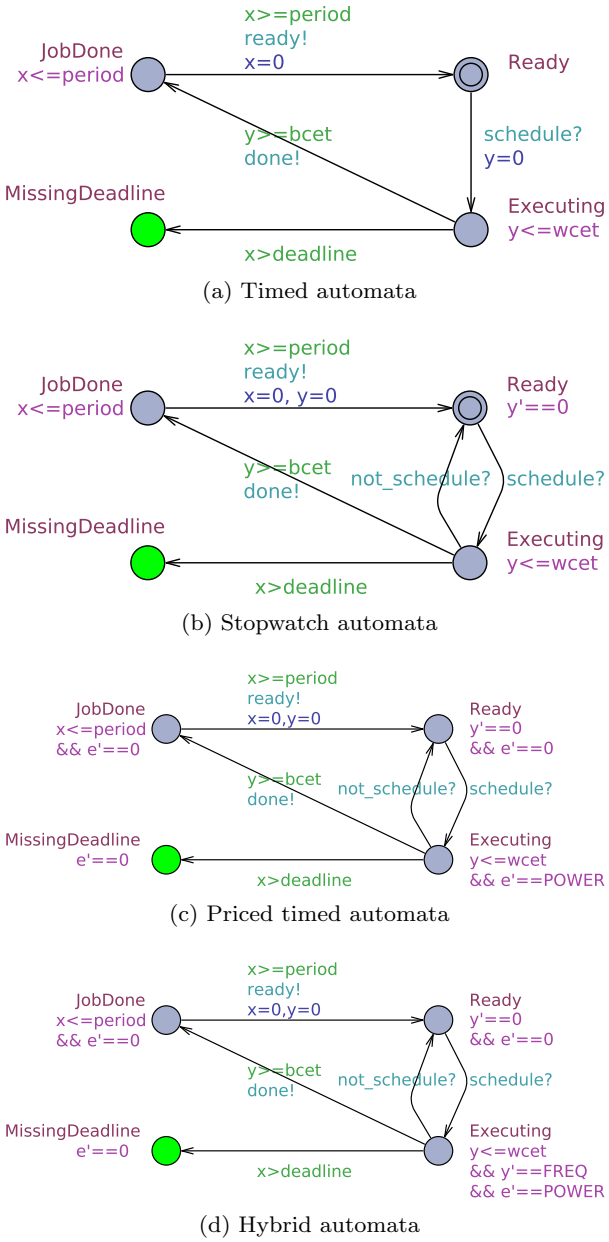


Fig. 1: Implementations of a simple real-time task with timed, stopwatch, priced and hybrid automata

3.3 Stochastic Hybrid Automata

Hybrid automata (and their sub-classes) may be used with a stochastic semantics [17,15] that refines all non-deterministic choices with probability distributions. This impacts the choice of delay, output and next state. For each state $s = (l, \nu)$ of an *HA* $\mathcal{H}$ we assume there exist the following probability distributions:

- the *delay density function* μ_s over delays in $\mathbb{R}_{\geq 0}$, that defines when the component will perform an output,
- the *output probability function* λ_s, that assigns probabilities to each available outputs $o \in \Sigma_o$,
- the *next-state density function* η_s^a, that provides stochastic information on the next state $s' = (l', \nu') \in \mathbb{R}^X$ given an action a.

Adding stochastic information Stochastic hybrid automata are analyzed with Uppaal SMC. Without additional information the tool is also able to run classical *TA*, *SWA*, *PTA* or *HA* with a stochastic semantics, that applies uniform distributions to delays in states with bounded delay, to outputs and to next states. Additionally the user can provide the rate of an exponential distribution for each location with unbounded delay, and discrete probability distributions between different outputs and next states.

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 have lower criticality. They can include sporadic tasks generated by users requests. In that case, average user demand will be efficiently modelled with a probability distribution. Similarly, timing executions may vary due to the content being display and can be measured from the system.

If analyzed with Uppaal model-checker, stochastic information from a stochastic hybrid automaton is discarded to consider only the underlying non-deterministic model.

Example 2 Stochastic hybrid automata with discrete probability distributions are useful to initialize the parameters of a model with random values, e.g., to specify that the period or the deadline of a task depend on some random information. They can be designed in Uppaal using a special node with one incoming transition (possibly with guard), and several outgoing transitions (displayed with dashed lines) that perform different updates and reach different locations, each associated to a probability weight (the probability of the transition is then the ratio of the probability weight over the sum of all the weights).

For instance, the simple automaton in Fig. 2 allows to select two values for the period of the task: 10 with probability 2/3 or 15 with probability 1/3. In what follows, we will call this automaton a *dispatcher*.

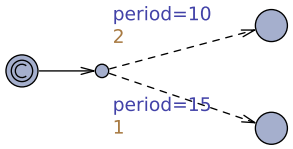


Fig. 2: Stochastic dispatcher implemented with a stochastic *TA*

In a network of stochastic *HA* 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.

3.4 (Statistical) Model-Checking

Model-checking is an automated verification technique that explores all the possible executions of a model to verify if it satisfies a property written in a formal logic. Model-checking can thus be used to assess the schedulability of a system, for any of its executions. This corresponds to the so-called worst-case analysis. However, even basic model-checking problems (reachability) are undecidable for *SWA* and *HA*. For these models it is only possible to perform exhaustive analysis with an over-approximation of the reachable states. The alternative is to exploit the stochastic semantics of *HA* and to resort to simulations and *statistical model-checking* (*SMC*).

SMC allows to reason on the average scenario, and to quantify the results with a probability measure. The principle is to combine formal verification and techniques from the statistic area in order to compute the probability that a system achieves a given objective.

There exists several *SMC* algorithms, see [28] for details. In this paper, we focus on the Monte-Carlo algorithm. This algorithm performs N executions ρ and then estimates the probability γ that the system satisfies a logical formula φ using the following equation: $\tilde{\gamma} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}(\rho \models \varphi)$, where $\mathbf{1}$ is an indicator function that returns 1 if φ is satisfied and 0 otherwise. The number of simulations N defines the precision of the results. It guarantees that the estimate $\tilde{\gamma}$ is close enough to the true probability γ, such that if $N = \lceil (\ln 2 - \ln \delta)/(2\varepsilon^2) \rceil$ the probability of error is $Pr(|\tilde{\gamma} -$

$\gamma \mid \geq \varepsilon) \leq \delta$, where ϵ and δ define the confidence interval and the confidence level, respectively.

Depending on the verification technique, we will use two types of formal properties. On one side, 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

$$\varphi ::= \mathsf{A[]}P \mid \mathsf{A<>}P \mid \mathsf{E[]}P \mid \mathsf{E<>}P$$

This branching logic allows to express properties over all the possible paths, using the path operators A and E, 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 instance for schedulability analysis, an error state is one where a task has missed a deadline.

On the other side, statistical model-checking queries express properties over a single trace, using the Bounded Linear Temporal Logic (*BLTL*). These queries only use state operators [] and <>, associated to a probability operator Pr and a time bound. The following query for instance "Pr[<=maxTime](<> error)" asks to compute the probability of reaching an error state before maxTime. Additionally, Uppaal SMC allows to write simulate queries that only examine traces without computing a probability.

4 Formalizing Scheduling Problems with Hybrid Automata

Scheduling problems consist in finding a time schedule that distributes a shared resource among several consuming devices. A common application is the scheduling of a set of *tasks* that need to access a *processor* to perform some computation. In a scheduling system, a *scheduler* implements a scheduling algorithm to find a good schedule. This scheduler can have different and antagonist goals, like minimizing the *response time* (the time needed to perform the work of a task), or on the contrary minimizing the energy needed for the computation. When considering a real-time environment, like an embedded control system, tasks have a deadline that must be satisfied absolutely. In that case, the main goal of the scheduler is to find a schedule such that the tasks always satisfy their deadlines.

In the context of real-time systems, tasks are often defined with a small number of characteristics. They are often periodic, such that a new *job* is released at every period. Each job consists in a computation that is only represented by an execution time (the time needed by the processor to complete the job). In that sense a job is a simple abstraction of the real computation that is performed. This job must be completed before the deadline (usually smaller than the period, so before the release of a new job).

Analytical scheduling methods determine if a set of tasks are schedulable by a given scheduling algorithm, using a scheduling test that is a function on the parameters of the tasks. Though effective, these techniques are limited to specific classes of scheduling policies and systems. An alternative is to use model-based approaches, with formal models of the components of a scheduling system (tasks, scheduler), and formal techniques such as model-checking and statistical model-checking. A recent series of papers [16,7,8,26] show that model-based approaches, implemented with timed automata and their extensions, are flexible enough to embed various types of scheduling policies, that go beyond those in the scope of analytical tools.

Model-based approaches also enable to use stochastic tasks whose real-time attributes, such as deadline, execution time or period, are characterized by probability distributions. This is particularly useful to describe mixed-critical systems and to make assumptions on the hardware domain. These systems combine hard real-time periodic tasks, with soft real-time sporadic tasks. Analytical scheduling techniques can only reason about worst-case analysis of these systems, and therefore always return pessimistic results. Using stochastic verification techniques like *SMC* we can instead analyze the system in a average scenario and provide more accurate measures.

4.1 Formal Models of Scheduling Components

The formal models of our scheduling systems are inspired by [7,8].

Tasks Tasks are implemented with a *SWA* shown in Fig. 3. From the Init location, a first job is initialized with real-time attributes obtained from the function setTaskAttribute(...). This job is queued for execution at location DlyPOoffset. There it requests the scheduler to assign a CPU, which is granted by a synchronisation on the channel req_sched[tstat[tid].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

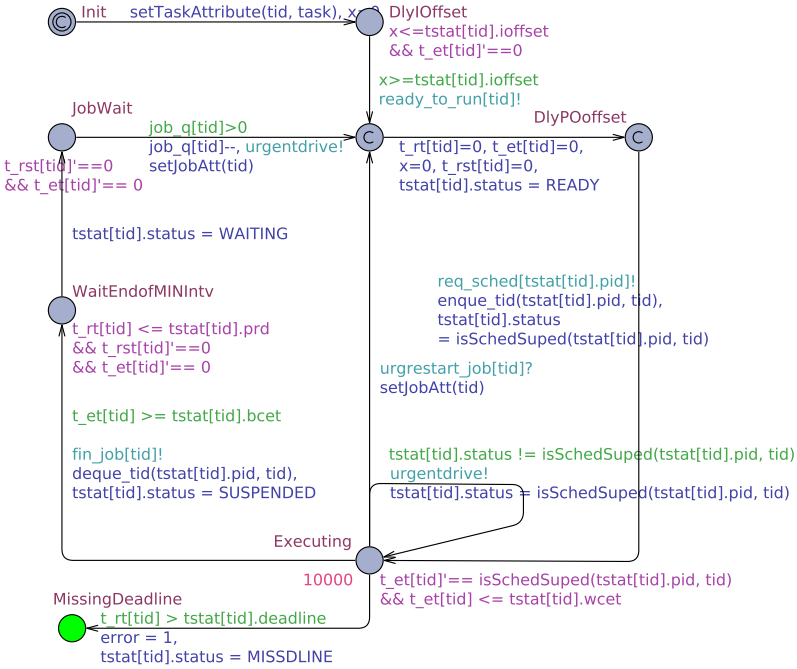


Fig. 3: *SWA* model of a stochastic task

execution time. This releases the CPU resource using function deque_tid(...). The *SWA* waits the end of the minimal inter-arrival time (WaitEndofMINIntv) and then instantiates a new job.

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

4.2 Stochastic Scheduling Systems

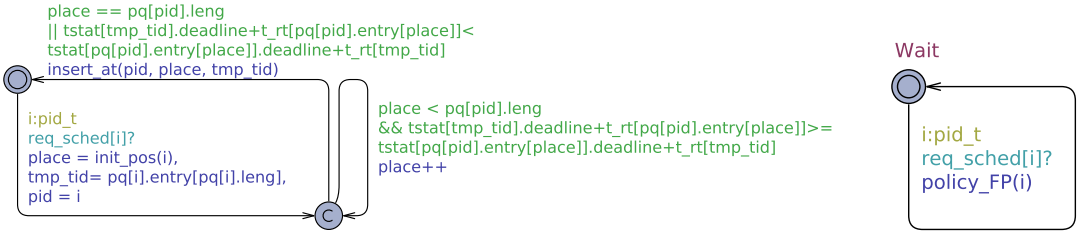
In [11] we have extended these models to use stochastic tasks whose real-time attributes (period, delay, execution time) depend on probability distributions, and are dynamically chosen by a stochastic dispatcher. This stochastic feature is of interest to model the variation of execution time with respect 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 the deadline and the period are determined according to the timing requirements of the functionality implemented by a set of tasks. For instance, some video decoder and 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.

In a stochastic task the stochastic attributes are determined by a stochastic dispatcher at each new instantiation of a job (when calling the functions setTaskAttribute and setJobAtt). The stochastic dispatcher is implemented with a stochastic timed automata using discrete probabilistic choices. Fig. 5 presents an example of a dispatcher that configures the three attributes with probabilistic choices between five values.

4.3 Multi-processor Scheduling Systems and Energy Consumption

Besides schedulability, various objectives can be asked upon the scheduler. One of these can be to measure and minimize the energy consumption. This is a great concern in energy limited systems, like cell phones or satellites, and more generally the power consumption of computing devices is an emerging topic.

Adding energy to our timed automata models require to extend the models with continuous variables and costs, using priced timed automata and hybrid automata. For measuring energy consumption we consider a multi-processor scheduling system with processors of different capabilities (frequency). Based on CMOS technology, the power consumption is dominated by dynamic power dissipation P_d when the processor is used



(a) EDF (b) FP

Fig. 4: *SWA* models of schedulers

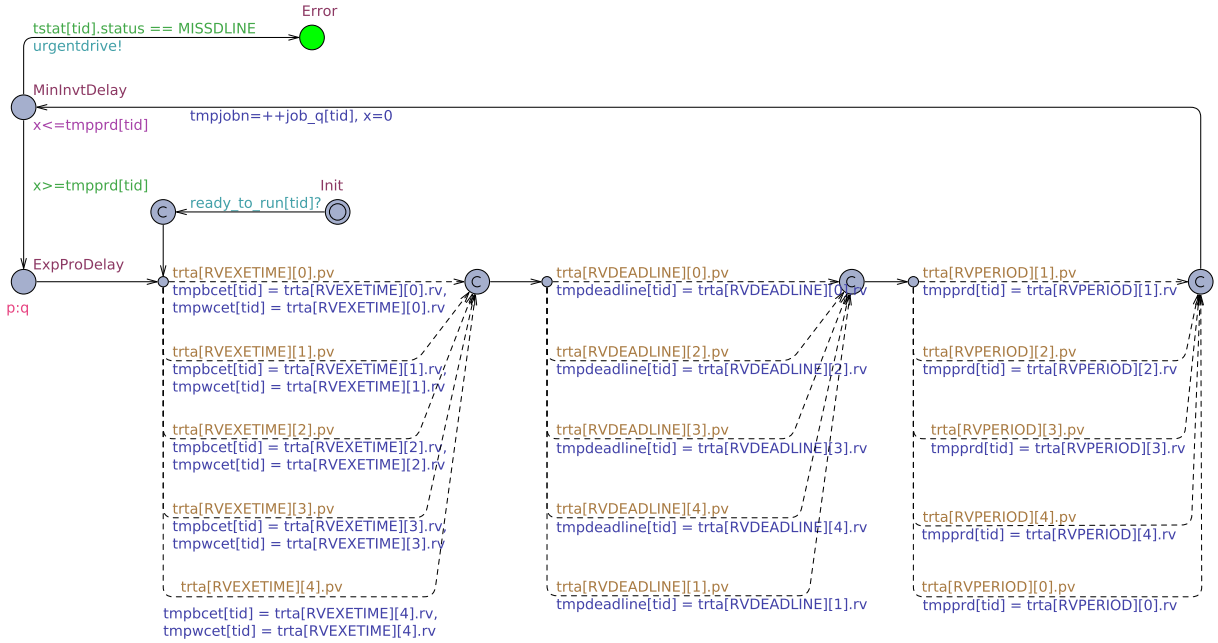


Fig. 5: *PTA* of the stochastic dispatcher

by some task, given by the following formula:

$$P_d = C * V^2 * f$$

where C is the capacitance, V the voltage and f the frequency. The processor speed is almost linear to the voltage:

$$f = k \cdot \frac{V - V_t}{V}$$

where k is a constant and V_t is the threshold voltage. We therefore get an approximated power consumption:

$$P_d = k' * f^3$$

k' being a constant ($C*k$). In our study we want to compare different configurations of the system according to the trade-off between speed (higher frequency) and energy consumption (lower frequency). We will therefore consider that the different configurations have the same constant characteristics by setting $k' = 1$ and only compare the energy consumption using the formula:

$$P_d = f^3$$

Then, our formal models for multi-processor scheduling systems define a set of processors, each having a frequency and its own scheduler. Processors can use different scheduling algorithms. Tasks are statically assigned to one processor.

To measure the energy consumption of the system, we add to our formal models a simple *PTA*. It defines a cost variable energy whose rate is energy'=totalPow, where totalPow is the power of all the running processors. If we increase the speed of a processor (the frequency f) we increase the energy consumption, but in return the task using the processor can run faster. We take this into account in our task model. The stopwatch clock t_et[tid] becomes a continuous variable that progresses at a rate t_et[tid]'=f, where f is the frequency of the processor that executes the task.

4.4 Hierarchical Scheduling Systems

One of the trends in developing CPS is to execute many heterogeneous real-time components into a single high-performance platform. This does not only reduce costs, but also improves performances and maximizes the utilization of hardware resources. However, these heterogeneous components must be partitioned, such that errors caused by one component are alienated from the other components. For instance, heterogeneous operating platforms in avionics and automotive systems manage various and different integrity-level applications. They are integrated using a high-performance hardware platform, supported by multi-core processors, advanced memory, and multi-level cache architectures.

This has motivated research on hierarchical scheduling systems (HSS), where a global scheduler is used to distribute a shared resource among several local schedulers. This mechanism can be duplicated in a multi levels system, effectively building a hierarchy of schedulers organized in a tree structure. On one hand, analytical methods have been proposed for HSS [39,38]. Though they are easy to apply once proven correct, proving their correctness is a difficult research topic, and they only provide a coarse abstraction that grossly overestimates the amount of resources needed.

On the other hand, there exists model-based techniques [16,7,8]. Since the complexity of the entire HSS is too large to be analyzed with formal methods, we rely on compositional approaches that allow to analyze each local scheduler independently [40].

In our formal framework, a HSS is a set of scheduling units organized in a tree structure. Each 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. To perform a compositional analysis of the system, we provide each scheduling unit with a resource supplier that abstract the behavior of the parent scheduling unit.

Resource Supplier The resource supplier is responsible for supplying a scheduling unit with the resource allocated from a parent scheduling unit. We adopt the *periodic resource model* (PRM) [40]. 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

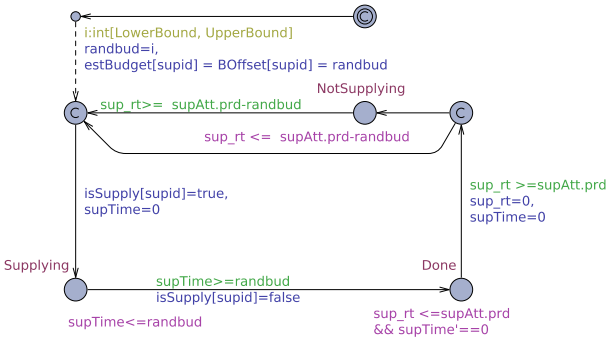


Fig. 6: Periodic Resource Model supplier with stochastic budget

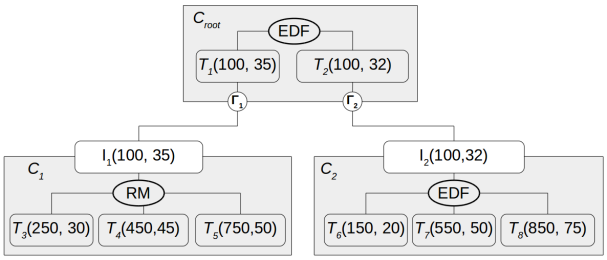


Fig. 7: Example of Hierarchical Scheduling System

variable isSupply that is set to true during the supply period.

We also use the probabilistic supplier model presented in [8]. This probabilistic supplier is implemented with the *SWA* of Fig. 6. Instead of using a fixed budget Θ, it uses a range of values specified between an interval [LowerBound,UpperBound]. At each execution the value is selected randomly over this range by the stochastic transition. This will allow to perform a parameter sweep with SMC by selecting randomly a value of the budget, and it will help determining the optimal budget.

Example 3 We present in Fig.7 a small example of HSS with three schedulers: a top scheduler C_{root}, with an EDF policy, and two bottom schedulers C_1 and C_2, with Rate Monotonic (RM) and EDF policies, respectively. The top scheduler schedules two tasks T_1 and T_2 that distribute the resource to the interfaces I_1 and I_2 of the lower schedulers. These interfaces use the PRM, each with a period of 100, and a budget of 35 for I_1 and 25 for I_2. The lower schedulers schedule three real-time tasks each using the resource they receive from the interfaces I_1 and I_2.

4.5 Scheduling Problems

We finally present the different problems that we want to solve on scheduling systems.

Problem 1: Correctness and performance We want to evaluate several properties of the scheduling system to assess its correctness and measure its performances:

1. **Absence of deadlock:** We check that the formal models have been correctly designed, such that they cannot reach a deadlock state, that is to say a state in which time is blocked and no action is available.
2. **Schedulability:** We determine whether the tasks are schedulable, i.e, none of them misses a deadline. In case of HSS, we check that all the scheduling units are schedulable.
3. **Maximum response time:** We measure the maximum response time of tasks, i.e., the maximum time between a job instantiation and its completion.
4. **Energy consumption:** We measure the average and maximum energy consumed by the system over a period of time.

Problem 2: Optimal configuration of the system Depending on their nature, our scheduling systems may admit different configurations. Then, we may evaluate each configuration according to one or several measures presented in *Problem 1* in order to select an optimal configuration.

1. **Multi-processor scheduling systems:** We consider a multi-processor system, with CPUs having different frequencies, and a set of real-time tasks. Our goal is to assign each task to a CPU. Then we evaluate the configurations of the scheduling system in terms of schedulability, response time and energy consumption.
2. **Hierarchical scheduling systems:** In a HSS, each scheduling unit is analyzed independently using the budget provided by the PRM. To configure the system we determine which budget values make the system schedulable. Our goal is to find minimum budgets, such that all the scheduling units are schedulable.

Problem 3: Change detection We now want to monitor our scheduling system in order to detect emerging behaviors or an expecting event. We consider a property of the system, based on the measures presented in *Problem 1*, e.g., the energy is always lower than a given value. We consider our system as a stochastic process and we evaluate the property at regular steps during an execution. This allows us to compute at runtime the probability to satisfy the property. Then, our goal is to detect an abrupt variation of this probability, which will be the sign that some event happened.

Formally, let S be a set of states and $T \subseteq \mathbb{R}$ be a timed domain. A stochastic process (S, T) is a family of random variables $\mathcal{X} = \{X_t \mid t \in T\}$, each X_t having range S. An execution of the stochastic process is any sequence of observations $\{x_t \in S \mid t \in T\}$ of the random variables $X_t \in \mathcal{X}$. It can be represented as a sequence $\pi = (s_0, t_0), (s_1, t_1), \ldots, (s_n, t_n)$, such that $s_i \in S$ and $t_i \in T$, with time stamps monotonically increasing, e.g. $t_i < t_{i+1}$. Let $0 \leq i \leq n$, we denote $\pi^i = (s_i, t_i), \ldots, (s_n, t_n)$ the suffix of π starting at position i.

Let φ be a property that can be evaluated to true or false on an execution. We consider a sequence of Bernoulli variables Y_i such that $Y_i = 1$ iff $\pi^i \models \varphi$. We define that the execution π satisfies a change $\tau = Pr[\pi \models \varphi] \geq p_{change}$, where $p_{change} \in]0, 1[$, iff $Pr[Y_i = 1] < p_{change}$ for $t_i < t$ and $Pr[Y_i = 1] \geq p_{change}$ for $t_i \geq t$. The first time t_i when this is detected is the time of change.

Example 4 Consider for instance a stochastic scheduling system as presented previously. We can evaluate at regular time intervals the probability that the energy consumption during the time interval exceeds a given value. This probability may change at runtime if the load of the scheduling system changes, because for instance some new task has been added. With change detection we would like to raise an alarm when the change occurs.

5 Solving Scheduling Problems

In this section we detail the techniques we use to solve the scheduling problems presented in Section 4.5.

5.1 Checking Correctness and Evaluating Performances with MC and SMC

The properties associated to *Problem 1* are translated into formal queries in the format of the tool Uppaal MC and Uppaal SMC.

Absence of deadlock We use the CTL formula `A[] not deadlock` that is checked with model-checking by the tool Uppaal.

Schedulability In our formal models we check schedulability 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 deadlines.

Then, schedulability is analyzed by Uppaal SMC using the following probabilistic query:

`simulate nbSim [<=runTime] {error} : 1 : {error}`

It asks to perform nbSim simulations of length runTime t.u., until one reaches a state labelled with error. If such a state is found, then the system is not schedulable.

Uppaal SMC performs a quick evaluation of the schedulability. If the system is not schedulable it may find quickly a counterexample execution. However, for an exhaustive result, we rely on model-checking with Uppaal using the CTL formula A[] not error. If the system contains stopwatches the analysis is performed with an over-approximation: if the result is true then the system is surely schedulable; if the result is false it may not be schedulable.

Maximum response time We measure this property using Uppaal SMC with the following query:

$$\mathsf{E}[<= \mathsf{runTime};\mathsf{nbSim}](\mathsf{max}:\mathsf{t\_resp}[2])$$

It runs nbSim simulations of runTime 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 model with a variable t_resp[2]).

Energy consumption We first measure the average energy consumed over a period of time. We use the following query:

$$\mathsf{E}[<= \mathsf{runTime};\mathsf{nbSim}](\mathsf{max}: \mathsf{PlatformEnergy.energy})$$

PlatformEnergy is the *PTA* that measures the energy using a cost variable energy. Uppaal SMC runs nbSim simulations of runTime t.u. and it computes the average value of the energy at the end of these simulations.

We can also check if the energy is always lower than a maximum value. We use the following probabilistic BLTL formula:

$$\mathsf{Pr}[<= \mathsf{runTime}]([]\ \mathsf{PlatformEnergy.energy} <= \mathsf{maxEnergy})$$

where runTime is the time length for the simulations and maxEnergy is an energy bound. With Uppaal SMC we compute the probability that the property is satisfied.

5.2 Optimization of a Multi-processor Scheduling System with ANOVA

We consider a set of CPUs, $\mathcal{C} = (CPU_1, CPU_2, \ldots, CPU_k)$ and a set of real-time tasks $\mathcal{T} = (T_1, T_2, \ldots, T_l)$. A multi-processor scheduling system is configured by specifying a mapping $\gamma : \mathcal{T} \mapsto \mathcal{C}$.

For each possible mapping, we would like to evaluate first, if the system is schedulable, and second, the average energy consumption and/or the maximum response time of a task $T_i \in \mathcal{T}$. The Uppaal query that we use to evaluate the energy consumption is:

$$\varphi_e = \mathsf{simulate}\ \mathsf{nbSim}[<= \mathsf{runTime}] \{\mathsf{PlatformEnergy.energy}\} : 1 : \mathsf{false}$$

and to evaluate the maximum response time of a task with id i:

$$\varphi_t = \mathsf{simulate}\ \mathsf{nbSim}[<= \mathsf{runTime}]\{\mathsf{max\_resp[i]}\ \} : 1 : \mathsf{false}$$

Finally we would like to compare the different configurations in order to select a schedulable configuration that has a minimum energy consumption and/or a minimum response time. If we want to achieve both objectives we are faced with a multi-objective optimization problem. A simple solution would be to analyze each configuration with SMC experiments in order to compute values for the energy consumption and the response time. However this require a lot of simulations per configuration to be able to compare them, as the confidence intervals should not overlap. Fortunately, there exists a more efficient statistical technique to solve this problem that is called *analysis of variance* (ANOVA). The test has already been used to perform optimization with *SMC* [14]. We propose in this paper new algorithms based on this test.

ANOVA is a statistical test used to compare several probability distributions. We use it in a single factor configuration with a fixed effects model, as presented in [33]. We have k treatments of a single factor (the system configuration defined by a mapping γ) that we wish to compare. For each $1 \leq i \leq k$, the observed response for treatment i is a random variable X_i (the energy or response time) for which we draw n random values $x_{i,1}, \ldots, x_{i,n}$ (computed by running n simulations of the system using the mapping γ_i and a property φ_e or φ_t). We denote $\overline{X_i}$ the mean of the random variable X_i and $\overline{X}$ the total mean all the values. ANOVA tests the null hypothesis that all the means of the treatments are equal, against the alternative hypothesis that at least two treatments have different means.

ANOVA is based on a comparison between the variability observed between the treatments and the variability observed within the treatments using the following F-value:

$$F = \frac{1/(k-1)\sum_{i=1}^{k}(\overline{X_i} - \overline{X})^2}{1/(n-k)\sum_{i=1}^{k}\sum_{j=1}^{n}(X_{i,j} - \overline{X_i})^2}$$

If the null hypothesis is true this F-value should follow a *F-distribution* defined by the degrees of freedom of the

experiment, that are $k-1$ and $n-k$. To determine if the null hypothesis holds a classical hypothesis testing solution is to compute the *P-value* of the test. The P-value is the probability of observing a more extreme F-value than the actual result. It corresponds to the area under the probability density function of the distribution greater than the F-value, as shown in Fig. 8. Therefore, the lower the P-value, the lower the probability that the F-value computed actually follows the F-distribution, and consequently the more likely the null hypothesis should be rejected. To make a decision we compare this P-value to a confidence level α, for instance $\alpha = 0.05$ for a 95% confidence. If P-value $\leq \alpha$ then the null hypothesis is rejected, i.e. some treatments have different means, with a 5% chance of making a Type I error.

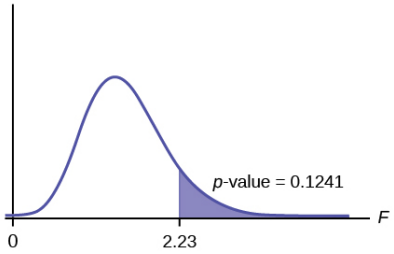


Fig. 8: F-distribution example with the p-value computed for F=2.23.

Tukey HSD If ANOVA shows that the means of the treatments are significantly different, then we would like to determine which treatments differ in order to compare them. In [14] the test was used with treatments that are continuous variables (temperature thresholds). In their context, using ANOVA alone, the authors were able to valid a linear regression over the continuous variables in order to optimize the system

In our context, the treatments (the different mappings) cannot be compared directly with ANOVA. The result of the test is only that at least two treatments differ, but we do not know which ones. Therefore we need an additional test to compare the treatments. This cannot simply be done by a series of pairwise T-test, as it would greatly increase the likelihood of false positive.

There exists however a multiple comparison test called Tukey HSD (Tukey's Honest Significant Difference test) that compares the means of every treatments to the means of every other treatment. It computes the pairwise differences $\overline{X_i} - \overline{X_j}$ with a confidence interval. If the endpoints of the confidence interval have the same sign (both positive or both are negative), then 0 is not in the interval and we conclude that the means are different. If the endpoints of the confidence interval have opposite signs, then 0 is in the interval and we cannot determine whether the means are equal or different. Tukey HSD is based on a *studentized range distribution.* As for the ANOVA test, each comparison of the Tukey test can be associated to a P-value to measure the level of significance.

Note that if the number of mappings is reduced to two, then Tukey HSD should be replaced by a T-test.

Algorithms Using the two statistical tests previously presented, we propose two new algorithms to optimize multi-processor scheduling systems. The algorithms determine dynamically the number of simulations needed to compare the means of energy consumption and/or response time with a sufficient confidence. Algorithm 1 has a single objective (minimizing the energy consumption or the response time), while Algorithm 2 considers both objectives simultaneously.

In these algorithms `Simulate` is a function that performs n simulation of a mapping γ and computes the values specified in the property φ (e.g. energy consumption or the response time). `RunANOVA` runs the ANOVA test on the simulations to determine if the mappings values are significantly different. It returns the P-value of the test. `RunTukeyHSDSingle` runs the Tukey HSD test on the simulations and determines the best mappings, which can be a single mapping, or a set of mappings that cannot be distinguished because there is not enough significance, or because they have the same probability distributions. `RunTukeyHSDMulti` runs the Tukey HSD test and returns True if all the differences have either a significant difference (P-value $\leq \alpha$) or are equal (P-value $\geq 1-\alpha$). `ComputeMeans` computes the means of the values for each mapping given in parameter over all the simulations of the mapping.

Finally we are able to select the "best" configurations. Let $(\gamma_1, \gamma_2, \ldots, \gamma_n)$ be the set of schedulable configurations. We denote $\mathsf{energy}(\gamma_i)$ the average energy consumed over a fix period of time and $\mathsf{resp}(\gamma_i)$ the maximum response time of one the task. If we consider only one objective we select the configuration with the minimum estimated value for $\mathsf{energy}(\gamma_i)$ or $\mathsf{resp}(\gamma_i)$. If we consider both objectives simultaneously we should select a configuration that is Pareto-efficient. Formally, a configuration γ_i is Pareto-efficient if there exists no other configuration γ_j such that $\mathsf{energy}(\gamma_j) \leq \mathsf{energy}(\gamma_i)$ and $\mathsf{resp}(\gamma_j) \leq \mathsf{resp}(\gamma_i)$. We can plot the results on a graph and draw a Pareto-efficiency curve that links the Pareto-efficient configurations.

Example 5 We consider that Algorithm 2 produces the results given in the graph shown in Fig. 9, with values energy and resp for a set of configurations from A

Algorithm 1: Single objective multiprocessor optimization

Input:
Γ: list of mappings
n: initial number of simulations
α: confidence level
$\varphi \in \{\varphi_e, \varphi_t\}$: simulation property.
Output:
$bestMappings$: set of mappings with minimum energy consumption or response time.
min: minimum mean of energy consumption or response time.

Let $conf$ be a Boolean, initialised $conf \leftarrow False$
$bestMappings \leftarrow \Gamma$
foreach $\gamma \in \Gamma$ **do**
  Let $simulations[\gamma]$ be the set of simulations of the mapping γ, initially empty.
while *not* $conf$ **do**
  foreach $\gamma \in bestMappings$ **do**
    $simulations[\gamma] \leftarrow simulations[\gamma] \cup \texttt{Simulate}(\gamma, n, \varphi)$
  P-value $\leftarrow \texttt{RunANOVA}(simulations)$
  if $P\text{-}value \geq 1 - \alpha$ **then**
    $conf \leftarrow True$
  if $P\text{-}value \leq \alpha$ **then**
    $bestMappings \leftarrow \texttt{RunTukeyHSDSingle}(simulations, \alpha)$
    if $|bestMappings| = 1$ **then**
      $conf \leftarrow True$
    foreach $\gamma \in \Gamma \setminus bestMappings$ **do**
      Remove $simulations[\gamma]$ from $simulations$
$min \leftarrow$ Min($\texttt{ComputeMeans}(simulations, bestMappings)$)

to F. Configurations A to D are Pareto-efficient. Configuration E is not Pareto-efficient because energy$(C) <$ energy(E) and resp$(C) <$ resp(E). Similarly, configuration F is no Pareto-efficient because resp$(B) <$ resp(F) and energy$(B) =$ energy(F).

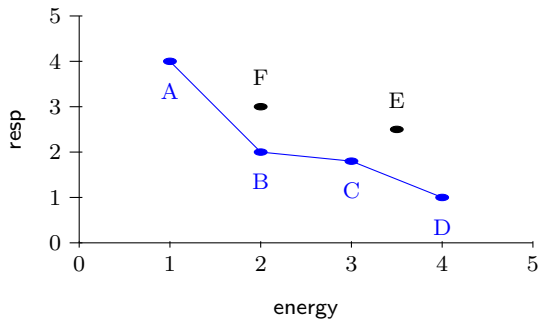


Fig. 9: Pareto-efficiency curve

Algorithm 2: Multi-objectives multiprocessor optimization

Input:
Γ: list of mappings
n: initial number of simulations
α: confidence level
φ_e, φ_t: simulation properties
Output:
$means\_e, means\_t$: means of energy consumption and response time for each mapping

Let $conf\_e$ and $conf\_t$ be Booleans, initialised $conf\_e \leftarrow False$ and $conf\_t \leftarrow False$
foreach $\gamma \in \Gamma$ **do**
  Let $simulations\_e[\gamma]$ be the measures of energy consumption of the mapping γ, initially empty.
  Let $simulations\_t[\gamma]$ be the measures of response time of the mapping γ, initially empty.
while *not* $conf\_e$ *or not* $conf\_t$ **do**
  if *not* $conf\_e$ **then**
    foreach $\gamma \in \Gamma$ **do**
      $simulations\_e[\gamma] \leftarrow simulations\_e[\gamma] \cup \texttt{Simulate}(\gamma, n, \varphi_e)$
    P-value $\leftarrow \texttt{RunANOVA}(simulations\_e)$
    if $P\text{-}value \geq 1 - \alpha$ **then**
      $conf\_e \leftarrow True$
    if $P\text{-}value \leq \alpha$ **then**
      $conf\_e \leftarrow \texttt{RunTukeyHSDMulti}(simulations\_e, \alpha)$
  if *not* $conf\_t$ **then**
    foreach $\gamma \in \Gamma$ **do**
      $simulations\_t[\gamma] \leftarrow simulations\_t[\gamma] \cup \texttt{Simulate}(\gamma, n, \varphi_t)$
    P-value $\leftarrow \texttt{RunANOVA}(simulations\_t)$
    if $P\text{-}value \geq 1 - \alpha$ **then**
      $conf\_t \leftarrow True$
    if $P\text{-}value \leq \alpha$ **then**
      $conf\_t \leftarrow \texttt{RunTukeyHSDMulti}(simulations\_t, \alpha)$
$means\_e \leftarrow \texttt{ComputeMeans}(simulations\_e, \Gamma)$
$means\_t \leftarrow \texttt{ComputeMeans}(simulations\_t, \Gamma)$

5.3 Optimization of a Hierarchical Scheduling System

To optimize a HSS we must determine the minimum budgets for the resource suppliers such that all the scheduling units are schedulable. For this purpose we use the stochastic model of the resource supplier presented in Fig. 6 that specifies a range of possible budgets Θ. Then we use Uppaal SMC to randomly select a value within this range and check whether the scheduling unit is schedulable with this value.

We use the following probabilistic BLTL formula:

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

It computes the probability distribution of all the possible budget values that are not schedulable. With Up-

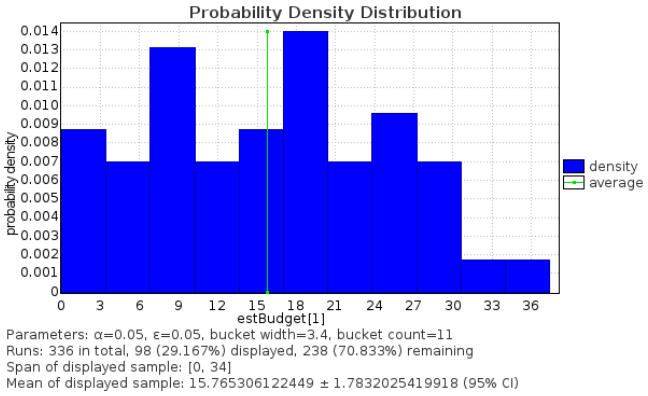


Fig. 10: Probability density distribution for the budgets for the scheduling C1.

PAAL SMC we can plot the probability density distribution in a graph, as shown in Fig. 10. 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.

Example 6 We consider the HSS example presented in Fig. 7. We analyze scheduling unit C_1 to compute the possible budgets for the resource supplier of this scheduling unit (such that the unit is schedulable). In Fig. 7, this budget was arbitrarily set at 35 over a period of 100. We would like to determine if this value is sufficient and if it can be lowered.

We set the range of budgets between 0 and 100. Using UPPAAL SMC we analyze the probabilistic BLTL formula presented above and we compute the probability density distribution shown in Fig. 10. It tells us that all the budgets lower than 34 have a non zero probability of being not schedulable. Therefore the minimum budget needed for the scheduling unit is 35 over a period of 100.

5.4 Change Detection with CUSUM

CUSUM [37,3] is a statistical algorithm used for monitoring change detection. The principle is to monitor the evolution of a probability measure at successive positions during a single execution of the system. The algorithm then detects the position where it drastically changes from original expectation. We have previously adapted the CUSUM algorithm to monitor a BLTL property over an execution trace of a stochastic process and to detect the position in the trace when the probability to satisfy the property changes significantly [29].

Let $\pi = (s_0, t_0), (s_1, t_1), \ldots, (s_n, t_n)$ be an execution of the stochastic process and φ a BLTL property to monitor during this execution. As defined in *Problem 3*, Y_i are Bernoulli variables such that $Y_i = 1$ iff $\pi^i \models \varphi$. We have that $p_k = Pr[Y_i = 1 | i <= k]$ is the probability of satisfying φ from (s_0, t_0) to the state (s_k, t_k). CUSUM will decide between the two following hypothesis:

- $H_0 : \forall\ k,\ 0 \leq k \leq n, p_k < p_{change}$, *i.e.*, no change occurs.
- $H_1 : \exists\ l,\ 0 \leq l \leq n$ such that the change occurs at time t_l: $\forall k,\ 0 \leq k \leq n$, we have $t_k < t_l \implies p_k < p_{change}$ and $t_k \geq t_l \implies p_k \geq p_{change}$.

We assume that we know the initial probability $p_{init} < p_{change}$ of $Pr[\pi \models \varphi]$ before the change occurs. One solution is to estimate this probability with the Monte Carlo algorithm using an ideal version of the system in which not change occurs.

Like the Sequential Probability Ratio Test (SPRT) [43], the CUSUM comparison is based on a likelihood-ratio test: it consists in computing the cumulative sum S_k of the logarithm of the likelihood-ratios s_i over the sequence of samples $Y_1, \ldots Y_k$. The change is detected as soon as S_k satisfies a stopping rule.

$$S_k = \sum_{i=1}^{k} s_i \quad s_i = \begin{cases} \ln \frac{p_{change}}{p_{init}}, & \text{if } X_i = 1 \\ \ln \frac{1-p_{change}}{1-p_{init}}, & \text{otherwise} \end{cases}$$

The typical behavior of the cumulative sum S_k is a global decreasing before the change, and a sharp increase after the change. Then the stopping rule's purpose is to detect when the positive drift is sufficiently relevant to detect the change. It consists in saving $m_k = \min_{1 \leq i \leq k}\ S_i$, the minimal value of CUSUM, and comparing it with the current value. If the distance is sufficiently great, the stopping decision is taken, *i.e.*, an alarm is raised at time $t_a = \min\{t_k : S_k - m_k \geq \lambda\}$, where λ is a sensitivity threshold.

CUSUM Calibration

The efficiency of the CUSUM algorithm depends on several parameters. First, it is important to note that the likelihood-ratio test assumes that the considered samples are independent. This assumption may be difficult to ensure over a single execution of a system, but heuristic solutions exist to guarantee independence. One of them is to introduce delays between the samples. In that case Monte Carlo SMC analyses can evaluate the correlation between the samples, and help to select appropriate delays.

Second, the CUSUM sensitivity depends on the choice of the threshold λ. A smaller value increases the sensitivity, *i.e.*, the false alarms rate. A false alarm is a change detection at a time when no relevant event actually occurs in the system. Conversely, big values may delay the detection of the change. The false alarms rate of CUSUM is defined as $E[t_a]$, the expected time of an alarm raised by CUSUM while the system is still running before the change occurs. Ideally, this value must be the biggest as possible ($E[t_a] \rightarrow +\infty$). The detection delay is defined as the expected time between the actual change at time t and the alarm time t_a raised by CUSUM ($E[t_a - t \mid t < t_a]$). Ideally, this value has to be as small as possible.

To calibrate the sensitivity a solution is to use two versions of the model: one in which the change never occurs and one in which it always occurs. Running the CUSUM on the first model allows to determine the minimum sensitivity such that no detection occurs. Then, the CUSUM is run on the second model to determine the detection delay.

6 High Level Languages for Scheduling Systems

Modeling a scheduling system requires a deep understanding of the various involved formal models and verification tools. They need to be adequately configured by the user to run the experiments and analyze the results. This comprises, in particular, the following five steps:

1. Select the right formal models, from the model bank presented in Section 4.
2. Input values to the parameters of these models, e.g., periods, deadlines and execution times of tasks.
3. Write the formal properties that must be analyzed.
4. Launch formal verification using Uppaal and Uppaal SMC.
5. Analyze and interpret the results.

These tasks should be simplified to help the adoption of formal methods by system engineers. We propose an approach that uses high-level domain-specific languages instead of generic formal languages with powerful, but complex, syntax and semantics. These languages focus on the necessary elements that engineers are used to manipulate to design their systems, while other unnecessary parameters of the formal models are hidden. The language is embedded in a user-friendly tool, which launches formal verification using automatic code generation of the formal models from the high-level description languages. Finally, the tool provides relevant and readable results of the formal verification on the components of the high-level language, instead of the components of the underlying formal models.

We have designed two high-level frameworks for scheduling systems, using Cinco [34], a generator for domain-specific modeling tools. The first framework is dedicated to hierarchical scheduling systems. The second is dedicated to the design and analysis for multiprocessor scheduling systems with energy constraints. The frameworks can be downloaded from Cinco's website[1].

6.1 Domain-Specific Code Generator: CINCO

Cinco is a generative framework for the development of domain-specific graphical modeling tools. It is based on the Eclipse Modeling Project [21], but with a strong emphasis on simplicity [32], so that the user (i.e. the developer of a tool generated with Cinco) does not need to struggle with the underlying powerful but complicated EMF metamodeling technologies [42] directly. This is achieved by focusing on graph model structures (i.e. models consisting of various types of nodes and edges) and automatically generating the required Ecore metamodel as well as the complete corresponding graphical editor from an abstract specification in terms of structural constraints. In a sense, this approach turns constraint-based variability management [23,27] into a tool generation discipline, where a product line is just characterized by the tools' modeling capacities.

6.1.1 Meta-Modeling

Central to every Cinco product is the definition of a file in the Meta Graph Language (MGL). It defines what kind of modeling components the model consists of and what attributes they have. Every modeling component is either a node type, a container type (i.e. a special node that can hold other nodes) or an edge type. It is also possible to define which kind of nodes can be connected by which kind of edges and express cardinality constraints on those connections.

Example 7 For instance, Listing 1 presents of a portion of an MGL file with the definition of a container node to represent a resource supplier in an HSS. The definition precises some attributes (policy, period, budget ...). It needs exactly one input transition and one or more output transitions. Furthermore, it can contain other nodes of type Query.

The second important file is a specification in the Meta Style Language (MSL), which is used for defining

[1] `http://cinco.scce.info/applications/`

```
@style(supplier,"${policy}","${resource}","${tid}",
    "${period}","${budget}")
container Supplier {
  attr Policy as policy
  attr Resource as resource
  attr EInt as tid
  attr EInt as priority
  attr EInt as period
  attr EInt as budget
  attr EInt as deadline
  incomingEdges (Transition[1,1])
  outgoingEdges (Transition[1,*])
  containableElements (Query)
}
```

Listing 1: Part of the MGL file that specifies the supplier of a HSS.

shapes (rectangle, ellipse, polygon, image, text, etc.) and appearances (colors, line style, line width, etc.) for nodes and edges. To change the look of the model depending on runtime information (e.g. the value of a node's attribute) one can either use the attribute directly within a text shape or implement an *appearance provider* that is invoked by the framework and may contain Java code that decides on the appearance by arbitrary external or internal factors.

Example 8 Listing 2 contains the style definition for the previous supplier node. It is displayed with a yellow rounded rectangle, and a smaller white rectangle inside, showing some parameters as text, as shown in Fig. 11.

```
nodeStyle supplier(5) {
  roundedRectangle r {
    appearance extends default {
      background (222,222,117) // light yellow
    }
    size(150,150)
    corner(10,10)
    rectangle p {
      appearance extends default {
        background (255,255,255)
      }
      position relativeTo r (CENTER,TOP)
      size(60,20)
      text {
        position relativeTo p (CENTER,MIDDLE)
        value "%1$s | %2$s"
      }
    }
    text {
      position relativeTo r (CENTER,MIDDLE)
      value "Supp%3$s: (%4$s,%5$s)"
}}}
```

Listing 2: Part of the STYLE file that configures the display of the supplier node.

Those specifications are already enough for Cinco to generate the complete graphical modeling tool. But Cinco also provides mechanisms to integrate code that

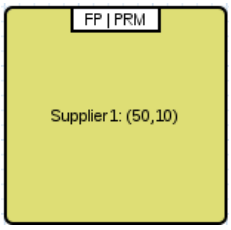


Fig. 11: Example of supplier display generated by the style configuration.

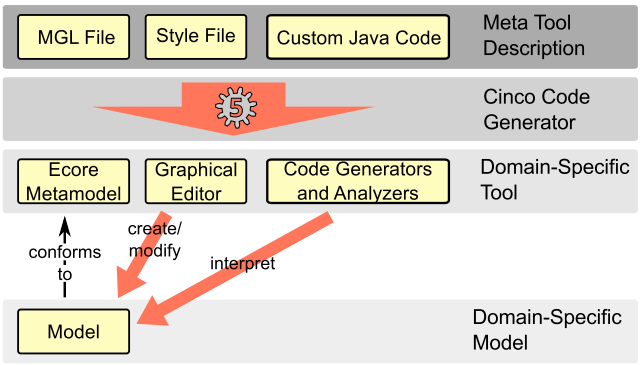


Fig. 12: Main principles of domain-specific tools generation with Cinco.

interprets or transforms the models. It automatically generates APIs specific to the model type and seamlessly integrates code implemented against it into a ready-to-run modeling tool, which is a realization of the one-thing-approach [31].

The main principles for the generation of a domain-specific tool with Cinco are depicted in Fig. 12. From the MGL and Style definitions, Cinco generates an Ecore metamodel as well as a corresponding graphical editor for the domain-specific tool. The user can then create a model in the tool that conforms to the given specification. This model can be analyzed by custom Java code, embedded in the tool during the automatic generation by Cinco.

6.1.2 Domain Specific Tool

Besides the tool meta-modeling, the second important feature of Cinco needed to develop a domain specific tool is the possibility to enhance the graphical editor by adding custom code. This code can call an API generated by Cinco to interact with the meta-model.

Fully pointing out all of Cinco's concepts and capabilities for the development of sophisticated domain-specific modeling tools is clearly beyond the scope of this paper. We therefore only briefly point out the aspects most relevant for our scheduling systems mod-

eling tools. Please refer to [34,35] or the website[2] for more detailed introductions.

The easiest way to enhance the graphical editor is by adding a *custom action* to a node type, which is then available via the nodes' context menu or on double-click. For a custom action two methods need to be implemented: `canExecute` and `execute`. Both receive the node on which the action should be performed as parameter. While the first decides whether the action is available (i.e. not disabled/greyed out in the context menu), the second one actually performs it. The generated enhanced API for the metamodel simplifies the implementation of those methods, as one can easily access related modeling elements in a semantic and type-safe way, e.g. by accessing all successors (i.e. target nodes of outgoing edges) of a certain type.

Furthermore, Cinco makes it especially easy to perform changes to the edited model. Usually, with the common Eclipse approaches, the visual representation as well as the underlying model structure need to be changed separately. The *transformation API* that Cinco generates for every model type handles the synchronous and consistent modification of both parts automatically, so that it becomes very straightforward to program transformations for the model, as the generated API provides the same actions the tool user can perform within the editor, e.g. change attributes, add new elements, connect them with edges, or move/resize/delete them.

6.2 High-Level Framework for Hierarchical Scheduling Systems

As presented in Section 4.4, HSS are best represented by a tree structure. This is the format we adopt for our graphical specification of HSS. Fig. 13 presents an example of an HSS designed in our framework. The nodes of the tree correspond to the components of the scheduling units (tasks, suppliers). In the rest of the section we detail the available components of our high-level language and their configuration parameters.

Resource suppliers TopSupplier(policy), in blue, is the root of the HSS tree. It supplies the resource to all the scheduling units. Its only parameter is the scheduling policy. Supplier(policy,period,budget), in yellow, are intermediate suppliers (e.g., Supplier1 in Fig. 13) that receive the resource from an upper level and supply real-time tasks or lower level suppliers. Their parameters are a scheduling policy, a period and a budget within this period. To estimate the necessary budget of a scheduling unit we use a probabilistic supplier, in red, (e.g., PSupplier2 in Fig. 13) whose budget is chosen randomly between values given in an interval. It is denoted ProbSupplier(policy,period,budget), where budget is an interval of the form [LowerBound,UpperBound].

Tasks Tasks are the leaf of the HSS tree. They represent the time spent for executing some computation. A task is denoted Task(period,deadline,bcet,wcet,priority) and represented in the model with a green rounded box. As presented in Section 4.1, we propose a new model of stochastic task whose attributes may be probability distributions. This type of task is denoted STask(period, deadline,execution,priority) and represented by a green rectangle. Here period, deadline and execution are discrete probability distributions. Instead of having a worst case and best case execution time, we input a probability distribution of execution times.

Queries Queries are associated to the suppliers. The following queries, that correspond to the formal properties presented in Section 5.1, are available: deadlock query, schedulability, maximum response time, and budget estimation. In Fig. 13 for instance, PSupplier2 is assigned a budget estimation query and Supplier1 a schedulability query. Queries that have been verified are colored automatically by the tool, in green if they are satisfied, or in red if they are not satisfied.

6.3 High-Level Framework for Multi-Processor Scheduling Systems

For the design of CPS with multi-processor we consider a two-layer approach as proposed in [25]. The first layer models the hardware platform, with a scheduling system composed of real-time tasks and CPUs. The second layer models the application that is composed of a set of actions. The link between the two layers is implemented by a mapping from actions to tasks, that specifies for each action of the application on which task it is intended to run. In our current framework this mapping is static and determined before an execution.

This design allows a separation of concerns that facilitates the verification of formal properties:

- Scheduling properties are verified on the platform layer only.
- Logical properties of the application are verified on the application layer only.
- Energy consumption or execution time properties need to consider both layers simultaneously.

[2] `http://cinco.scce.info`

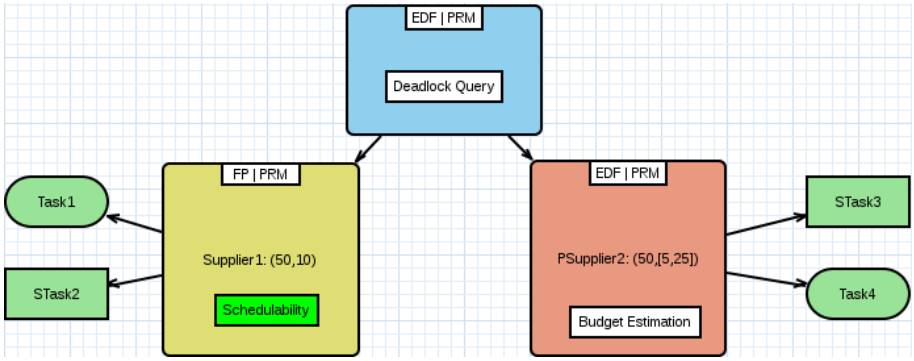


Fig. 13: HSS with 3 scheduling units

We have implemented with CINCO a high-level framework that allows to design a two-layer multi-processor scheduling system.

Platform Layer The platform layer is composed of a set of processors and a set of real-time tasks. Each processor has its own scheduling mechanism and is parameterized by its frequency. The frequency defines the speed of the processor and its energy consumption when running. Real-time tasks can be either hard real-time, with a deadline, a period and execution times, or soft real-time, with only period and execution times. Tasks are statically assigned to a processor. A model of a platform layer designed in our framework is presented in Fig. 14. This model is translated into a set of timed automata, using the models presented in Section 4.

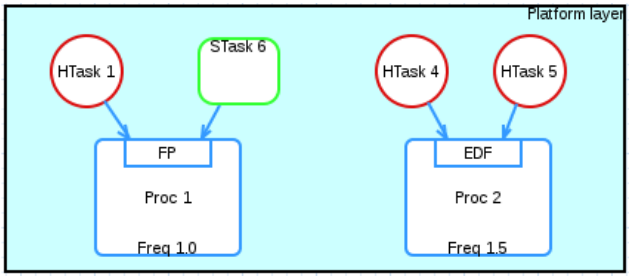


Fig. 14: Platform layer with 2 processors, 3 hard real-time tasks and 1 soft real-time task

Application Layer Applications running in CPS are unlimited, with no fixed design. To demonstrate the use of our framework we consider a simple design methodology for writing applications with stochastic behaviour. Our application is composed of a set of components. Each component consists in a sequence of actions. Each action has a delay mechanism, implemented with either a uniform or an exponential probability distribution, and minimum and maximum execution times. Actions are also parameterized with an energy consumption parameter (between $[0, 1]$) that defines how much power the action will take from the CPU. The semantics of this language is to execute each component in parallel by running their actions iteratively. A component that has completed its last action will continue in a loop with the execution of the first action. An example of application is presented in Fig. 15. These models are translated in a set of stochastic timed automata.

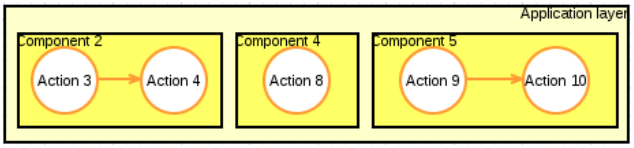


Fig. 15: Application layer with 3 components and 5 actions

Mapping between application and platform The mapping between the two layers is done by linking each action of the application to a real-time task, as presented in Fig. 16.

Queries: In this framework we consider different type of queries, some of them associated to the platform and some of them associated to the application. On the platform layer we verify schedulability queries and we determine optimal mapping between tasks and processors with ANOVA, as presented in Section 5.2. On the application layer we measure average energy consumption and we use CUSUM to detect changes in the application behavior.

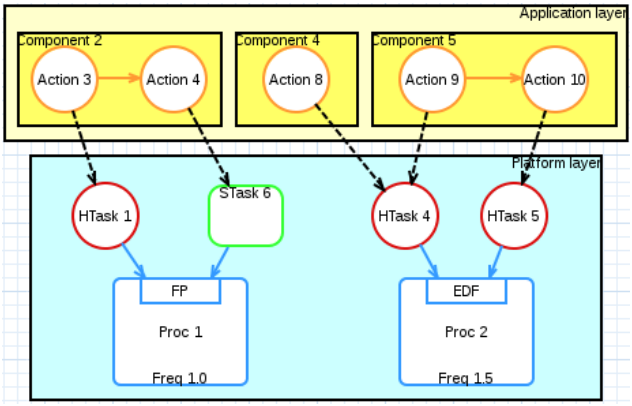


Fig. 16: Mapping between application layer and platform layer

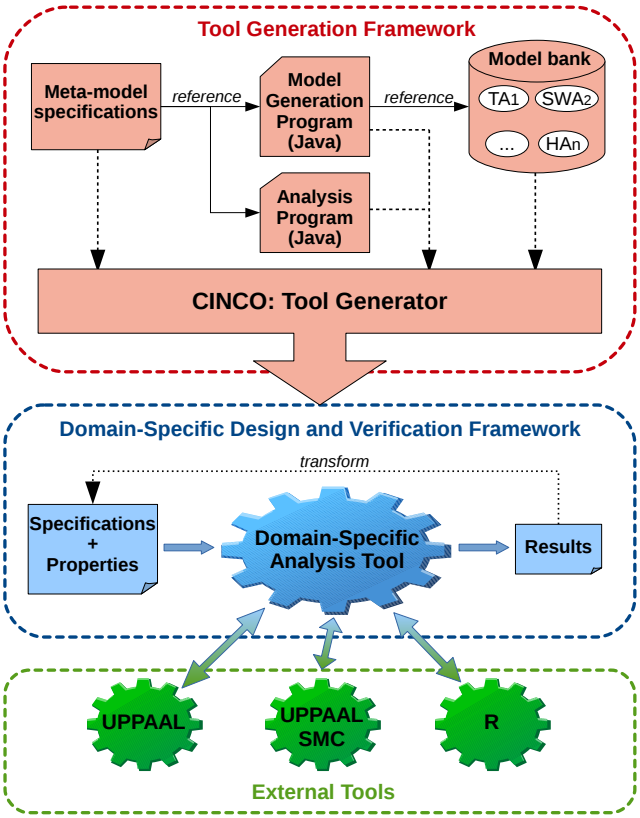


Fig. 17: Tool chain for generating and using domain-specific analysis frameworks

6.4 Implementation of the Frameworks and Tool Chain

We have implemented two domain-specific analysis frameworks: one dedicated to HSS and one dedicated to multi-processor scheduling systems. The tool chain involved in the generation of these frameworks and then in their usage is described in Fig. 17.

The frameworks are developed in Java and with Cinco. The graphical interface of the framework is specified with the meta-modeling languages of Cinco, presented at the beginning of this section. Then we have developed Java programs for generating complete formal models from the high-level specifications. These generators use existing formal models from a model bank (the models presented in Section 4). Finally we have developed Java custom analysis programs. These programs are linked to the code generated by Cinco such that they can be started directly from the graphical interface, either by a right-click menu or double-click actions. These programs solve the problems listed in Section 4.5 using the techniques presented in Section 5. In the background, they launch Uppaal and Uppaal SMC via the command line interface to perform formal verifications, and they use the tool R for statistical analysis. The textual results of these verifications are then analyzed by our programs to determine the relevant results of the analysis. Then, the transformation API of Cinco is used to visualize the results on the model's high-level abstract view, either by creating pop-up windows or by making modifications on the model designed in the interface.

Using the meta-model specification, our custom Java code and the model bank, Cinco automatically generates a domain-specific framework, that includes the Java code and the Ecore specifications of an Eclipse graphical interface. This domain-specific framework allows to design scheduling problems, using the high-level graphical languages presented before. It then launches analysis by calling external tools (Uppaal, Uppaal SMC and R). It produces results and consequently can transform the original high-level specification.

We detail below the basic steps performed by our analysis programs to solve the different scheduling problems.

Correctness and performance The program that solved these problems first generates a Uppaal model from the model designed in the graphical interface of the framework. It also generates a text file with the Uppaal query needed for the analysis. It then launches Uppaal or Uppaal SMC and analyses the results. The following results are displayed in the interface:

- The absence of deadlock is shown in a pop-up window. The color of the query is turned to green or red according to the result.
- The schedulability analysis produces a pop-up window with the result. The color of the query is turned to green or red. Additionally, if the result is false the color of the task that has missed a deadline is turned to red.

– The measures of maximum response times or energy consumptions are displayed in pop-up windows and in the queries.

Optimization of Hierarchical Scheduling Systems The program that solves this problem generates the UPPAAL model of the HSS with a probabilistic supplier, as the one presented in Fig. 6. It analyses the schedulability query with UPPAAL SMC. This generates a probability distribution, as the one presented in Fig. 10. The program analyses this distribution to determine the minimum budget. It displays the result in a pop-up window.

Optimization of Multi-processor Scheduling Systems We have implemented Algorithms 1 and 2. Our program generates a set of UPPAAL models, each corresponding to a configuration of the system with a mapping from tasks to processors. It then runs the optimisation algorithm. This algorithm launches some simulations with UPPAAL SMC and extracts the numerical results. The results are written in some temporary files. that are analysed with the statistical tool R to perform the RunANOVA, RunTukeyHSDSingle, RunTukeyHSDMulti and ComputeMeans procedures. According to the results the program determines if more simulations are needed, or outputs the result.

For the single objective problem the program directly shows the optimal mapping by drawing it on the interface using the transformation API of CINCO.

For the multi-objectives problem the program opens a pop-up window and draws into the Pareto diagram. This window allows to select one of the Pareto-efficient mapping that is then drawn on the interface.

Change detection The program that performs change detection implements the CUSUM algorithm. It first generates the UPPAAL model and it will run the CUSUM algorithm on this model several times. For each execution, it generates with UPPAAL SMC a simulation trace that corresponds to the total length of the experiment. It then splits this execution into a set of samples and it analyses each sample to evaluate the query and update the CUSUM ratio. If the value of ratio exceeds the sensitivity threshold it outputs a detection with the detection time in a pop-up window.

7 Experiment with Hierarchical Scheduling Systems

We apply our framework for HSS to model and verify an avionic scheduling system. We consider the specification of avionic tasks presented in [30]. This is a mixed-critical system with multiple tasks of various criticality running together. We arrange these tasks in a hierarchical scheduling system by grouping tasks from similar functions and critically (Navigation, Targeting, Weapon control and Controls and displays). Each function is associated to a scheduling unit. The three scheduling units of the most critical functions (Navigation, Targeting and Weapon control) are further grouped under a "Hard-Subsystem" scheduling unit. This results in the hierarchical scheduling systems presented in Fig. 18.

The goal of our study is to determine if the complete system is schedulable and to find appropriate parameters for each scheduling unit, such that they are all schedulable.

High-level model We design the HSS in our domain-specific tool generated by Cinco, using the high-level language presented in Section 6.2. Sporadic tasks are modeled with stochastic task nodes and are associated to probability distributions. To estimate their necessary budget, each scheduling unit is modeled using a probabilistic supplier.

Verification procedure We analyze each scheduling unit, starting from the bottom, with the budget estimation query. We configure the scheduling unit, by selecting several values for the period of the probabilistic supplier. The period must be lower than the minimum period of the tasks being supplied. Then, we configure the minimum and maximum budget for the estimation between $[1, period]$. The tool computes the minimum budget such that the tasks are schedulable. The ratio $budget/period$ gives us the load factor of the scheduling unit. Our goal is to find the lowest load factor among the choice of possible values for the period.

When all the bottom units have been analyzed we can replace them with normal supplier using the minimum budget that has been computed. We then repeat the procedure to compute the minimum budget for the upper scheduling units.

Results We present in Fig. 19 the results obtained from the analysis of the 3 bottom scheduling units (Navigation, Targeting, Weapon control). The graph plots the load factor of the scheduling unit using the minimum budget computed with SMC for several values of the periods. From these results we select the points with the lowest load factor and the highest period. The values that we choose are listed in Table 1.

We can now replace these probabilistic Suppliers with normal suppliers and confirm the schedulability of the units using the schedulability query, that is checked either with model-checking or SMC.

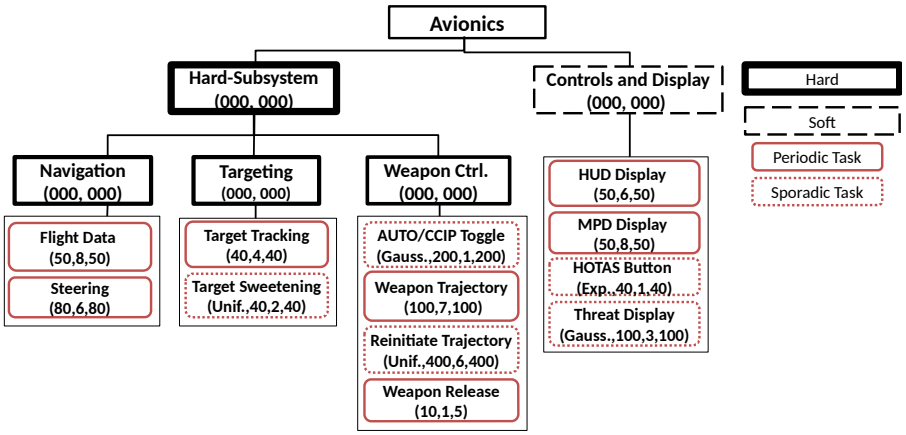


Fig. 18: Hierarchical scheduling of avionic tasks

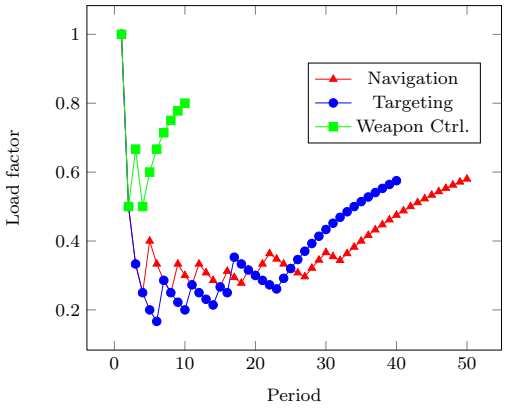


Fig. 19: Budget estimation for Navigation, Targeting and Weapon control

Unit	Period	Budget	Load factor
Navigation	8	2	0.25
Targeting	6	1	0.17
Weapon Ctrl.	4	2	0.5
Hard-Subsystem	4	4	1
Controls and Display	3	1	0.33

Table 1: Minimum budget for the scheduling units

We then determine the period and the budget for the Hard-Subsystem unit. Its period must be lower than 4, the chosen period of the Weapon control unit. Since the combined load factor of the 3 lower scheduling unit is 0.92, only a budget of 4 over 4 can scheduled the Hard Subsystem unit, which we verify with the schedulability query.

We also determine the necessary budget for the Controls and display scheduling unit. We found the best budget to be 1 over a period of 3.

From our results we conclude that the two upper scheduling units (Hard Subsystem and Controls and Display) are each schedulable. However since the load factor of the Hard Subsystem is already 1, it cannot be scheduled with the second unit using the same resources.

8 Experiment with Multi-processor Scheduling Systems

This section present an example of a multi-processor scheduling system designed and analyzed in our framework. We first describe the model and then we present the experiments performed in our framework to solve the problems presented in Section 5.

8.1 Example

The proposed example is composed of two layers, following the modeling framework presented in Section 6.3, a Platform layer and an Application layer.

The Platform Layer is composed by 3 periodic hard real-time tasks and 2 processors. The tasks parameters are configured according to the following order: Task(period,deadline,bcet,wcet,priority), and are respectively $T_1(10, 10, 3, 4, 9)$, $T_2(20, 20, 5, 6, 8)$ and $T_3(30, 30, 6, 8, 7)$. The 2 processors are P_1, with a 1.5 MHz frequency and a FP scheduling policy, and P_2, with a 1.0 MHz frequency and an EDF scheduling policy. We initially distribute T_1 and T_2 on processor P_1, while T_3 is running alone on processor P_2.

The Application Layer consists of 3 components, each composed by a succession of actions as presented in

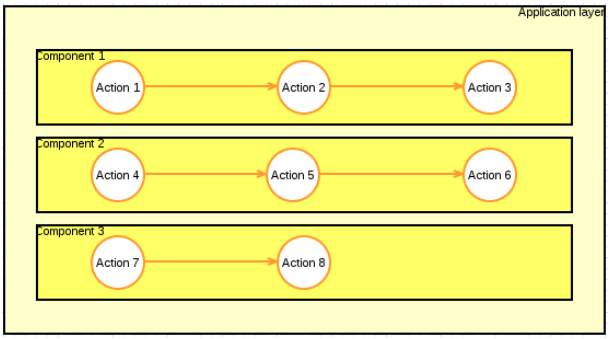


Fig. 20: Application layer of our case-study model

Fig. 20. Component C_1 is composed of actions A_1, A_2 and A_3, whose execution times are respectively 4, 3, 5. These actions are executed on task T_1. Component C_2 is composed of actions A_4, A_5 and A_6, whose execution times are respectively 4, 5, 5. These actions are executed on task T_2. Component C_3 is composed of actions A_7 and A_8, whose execution times are respectively 5 and 6. These actions are executed on task T_3.

Each action has an energy parameter that defines how much energy it takes when running on a processor, with a maximum value of 1 meaning that it takes the full power of the processor.

Finally, random delays with uniform distributions are set between the execution of each actions. As explain in Section 6.3 the execution of each component is cyclic: it runs sequentially each action, and then starts again at the first action. Action A_8 is additionally delayed, such that is starts only after 50 or 100 executions of action A_7. Using the change detection problem and CUSUM we will try in our experiments to detect the beginning of execution of this action.

8.2 Checking Correctness and Evaluating Performances

Experiments Using SMC we perform the following experiments on the initial model:

1. Schedulability analysis.
2. Measure of energy consumption, considering the platform only and both the platform and the application.
3. Measure of the maximum response time for each task.

We use 100 simulations and a runtime of 60 t.u. This runtime allows to execute the model over the smallest common multiple of the periods of our tasks (the hyperperiod).

Results The results of these experiments are presented in Table 2. We give for each result the time taken by the analysis. If the result is a measure we give its estimated value and the confidence interval that corresponds to the SMC analysis.

Analysis	Result	Time (s)
Schedulability	True	4.47
Energy consumption (platform)	69.408 ± 0.46	1
Energy consumption (application)	37.69 ± 0.72	4.2
Maximum response time of T_1	3.86 ± 0.025	3.8
Maximum response time of T_2	9.36 ± 0.055	3.78
Maximum response time of T_3	4.89 ± 0.061	3.78

Table 2: Correctness and performances analyzed with Uppaal SMC

8.3 Optimization with ANOVA

Experiment This second experiment consists in finding optimal mappings between tasks and processors, such that the system is schedulable and has optimal performances. Therefore we start by removing in our model the mapping used in the previous section. Then we use the ANOVA method with the multi-objectives Algorithm 2 proposed in Section 5.2. Our two objectives are to minimize the energy consumption of the scheduling system and to the maximum response time of one of the tasks. The result is a Pareto efficiency diagram.

For this experiment with will use SMC with 100 simulations to determine schedulability, and Algorithm 2 with ANOVA and Tukey HSD techniques with a 95% confidence.

Results Table 3 presents the results of executing Algorithm 2. We perform 3 executions of the algorithm (Exec. 1, Exec. 2 and Exec. 3) that are differentiated according to the task for which we want to minimize the maximum response time. One execution takes approximately 40 seconds. We determine that there are 8 mappings schedulable, simply named mapping-i with i from 1 to 8. Then for each execution we give in column E the energy consumption of the processors, and in column $t(T_i)$ the maximum response time of a task T_i.

Let us now consider that task T_2 is our critical task for which we want to minimize the maximum response

Mapping	Exec. 1		Exec. 2		Exec. 3	
	E	$t(T_1)$	E	$t(T_2)$	E	$t(T_3)$
mapping-1	98.1	2.57	98.1	6.21	97.8	7.15
mapping-2	115	2.58	115	6.24	115	11.7
mapping-3	77.2	2.57	77.3	5.74	77.5	12.2
mapping-4	95.3	2.57	95.1	5.76	94.9	7.18
mapping-5	70.1	3.81	70.2	3.58	70.1	9.99
mapping-6	88.5	3.80	88.5	3.74	88.8	8.29
mapping-7	50.9	3.87	50.8	9.34	50.7	19.3
mapping-8	69	3.86	68.7	9.33	69.2	4.91

Table 3: Optimization of the mapping between tasks and processors according to energy consumption and maximum response time of tasks T_1, T_2 or T_3

time. From the results given in columns 4 and 5 we can plot in our framework a Pareto diagram in a pop-up window, as shown in Fig. 21. From this window we can select one of the Pareto-efficient mapping that will then be automatically applied to the model.

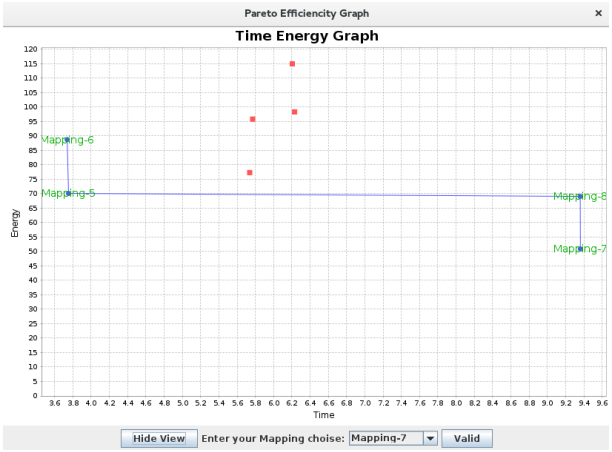


Fig. 21: Pareto Efficiency diagram for optimizing energy consumption and maximum response time of task T_2

8.4 Change Detection with CUSUM

Experiment In our third experiment we analyze each of the 4 optimal mappings found in the previous experiment and shown in Fig. 21. We use the CUSUM algorithm presented in Section 5.4 to detect the beginning of execution of action A_8. While action A_7 that precedes A_8 consumes only 80% of the CPU power, A_8 when it starts consumes the full power. This difference should increase the probability that the maximum energy consumption during a sample exceeds a given level. We consider a sample time of 60 time units, that corresponds to the hyper-period of the executing platform. We will observe at each sample of an execution the probability to exceeds the maximum energy consumption. This probability should raise when action A_8 starts executing. We will monitor the variation of this probability during an execution of 300 samples, i.e. 18'000 time units. We will the CUSUM algorithm to detect a change of probability and measure the detection time. We repeat the CUSUM 100 times and we compute the detection time as the average detection time over all the execution of the CUSUM.

To configure the CUSUM algorithm we first need to determine the initial probability. In this example we choose to estimate this probability by executing the optimal model, that is the model without action A_8 that provokes the change. The second parameter that we need to configure is the deviation from the initial probability when the change occurred. This parameter is estimated by computed the energy consumption on a model in which that action is already running at the beginning of the execution.

After fixing these two parameters, we proceed to the calibration of the CUSUM algorithm. This step consists in computing the sensitivity threshold λ. It is done by executing CUSUM on the optimal model, without the action responsible for the change, and using the initial probability and the deviation computed before. The threshold λ will be the minimal value such that no detection is observed for all simulations.

Results We run CUSUM on the set of Pareto-efficient mappings of Fig. 9. The analysis of one model takes approximately 20 minutes. The results for each mapping are presented in the following tables. In these tables, the first column (Energy) is the energy level used for the detection, the second column (Init. prob.) is the initial probability , the third column (Deviat.) is the probability deviation, the fourth column (λ) is the sensitivity threshold λ, the fifth and sixth columns (T.Detect) are the detection times, in the cases when action A_8 starts after 50 or 100 executions of A_7.

Table 4 presents the results obtained for mapping-6, that executes T_1 on processor P_1 and T_2, T_3 on processor P_2. With this mapping we can measure experimentally with Uppaal that action A_8 starts after approximately 1470 t.u. when its start parameter is 50, and 2950 t.u. when its start parameter is 100.

Energy	Init. prob.	Deviat.	λ	T. Detect (50)	T. Detect (100)
48	0.665	0.27	7.2	2745	4282
50	0.227	0.432	7.4	2278	3814
52	0.042	0.215	8.4	3109	4396

Table 4: Change detection results for mapping-6

Table 5 presents the results obtained for mapping-5, such that T_1 and T_3 are executed on processor P_1, and T_2 is executed on processor P_2. In this mapping action A_8 begins after approximately 1480 t.u. for a start of 50, and 2970 t.u. for a start of 100.

Energy	Init. prob.	Deviat.	λ	T. Detect (50)	T. Detect (100)
44	0.867	0.059	5.3	7277	8646
46	0.492	0.255	5.4	9381	9546
48	0.135	0.124	5.5	4961	6425
50	0.026	0.03	5.7	10725	11761

Table 5: Change detection results for mapping-5

The third mapping is mapping-8 such that T_1 and T_2 executes on P_1 and T_3 executes on P_2. The results are presented in Table 6. In this mapping action A_8 begins after approximately 1510 t.u. for start of 50, and after approximately 3010 t.u. for a start of 100.

Energy	Init. prob.	Deviat.	λ	T. Detect (50)	T. Detect (100)
39	0.48	0.475	9.0	2362	3851
41	0.289	0.518	4.7	1932	3416
43	0.118	0.489	7.0	2339	4004
45	0.033	0.271	8.0	2557	4070

Table 6: Change detection results for mapping-8

The last Pareto-efficient mapping is mapping-7 that executes all tasks on P_1. Results for this mapping are presented in Table. 7. In this mapping the action A_8 begins after approximately 1480 t.u.for start of 50, and after approximately 2980 t.u. for a start of 100.

Energy	Init. prob.	Deviat.	λ	T. Detect (50)	T. Detect (100)
23	0.976	0.013	4.0	3687	3714
25	0.832	0.067	20.8	16211	16730
27	0.612	0.086	19.8	13067	13683
29	0.388	0.119	13.5	7379	8587
31	0.188	0.106	3.8	6836	8170
33	0.038	0.103	8.3	7348	8092

Table 7: Change detection results for mapping-7

Discussion In these experiments, we are mainly interested in the detection delay, that is the delay between the true occurrence of the event and its detection by our CUSUM algorithm. Since our models are stochastic and our experiments are based on statistics there is inevitably some variance in the results. First we have configured our algorithm in order to limit to the minimum the occurrences of false alarm. As we can see in the results there is no detection before the true occurrence of the event. There is however some detection delay. Since our algorithm is based on the measure of energy consumption, the event that we monitor (the start of action A_8) needs some time to produce effects on the energy consumption. Indeed the change produced by this event is quite subtle (a change from 80% CPU power to 100% CPU power, when the action is running). Nevertheless the algorithm always manages to raise a detection.

Looking more closely at the results from the different mappings, we can observe that the best results are obtained from mapping-8, a model in which action A_8 (that runs on task T_3) is executed alone on processor P_2. This result can be explained by the fact that A_8 running alone on P_2 is not perturbed by the preemption from other tasks, and therefore tends to produce more deterministic effects on the energy consumption. In Table. 3 we can see that mapping-8 also provides the best maximum response time for task T_3.

9 Conclusion

We have presented a software engineering approach that generates model-based analysis tools for the schedulability analysis of CPS. This approach is based on one side on a set of formal models for describing complex scheduling problems, and on the other on meta-models of high-level specification languages to easily specify these scheduling problems.

Our approach generates automatically domain-specific analysis tools based on the Cinco framework. These tools allow to specify scheduling problems using graphical components, and they can launch formal analyses by calling model-checking tools such as Uppaal and Uppaal SMC. We have also presented new statistical model-checking algorithms that perform optimization or runtime monitoring. These algorithms are based on statistical tests like ANOVA and CUSUM. They are implemented and embedded into our analysis tools.

Using this approach we have proposed two domain-specific tools, one for hierarchical scheduling systems, and one for multi-processor scheduling systems with energy constraints. We have experimented these tools on two case-studies.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994). DOI 10.1016/0304-3975(94)90010-8

2. Alur, R., La Torre, S., Pappas, G.J.: Optimal paths in weighted timed automata. Theor. Comput. Sci. **318**(3), 297–322 (2004). DOI 10.1016/j.tcs.2003.10.038
3. Basseville, M., Nikiforov, I.V.: Detection of Abrupt Changes: Theory and Application. Prentice-Hall, Inc. (1993)
4. Beauquier, D.: On probabilistic timed automata. Theor. Comput. Sci. **292**(1), 65–84 (2003). DOI 10.1016/S0304-3975(01)00215-8
5. Behrmann, G., David, A., Larsen, K.G., Håkansson, J., Pettersson, P., Yi, W., Hendriks, M.: UPPAAL 4.0. In: Third International Conference on the Quantitative Evaluation of Systems (QEST), pp. 125–126 (2006). DOI 10.1109/QEST.2006.59
6. Behrmann, G., Fehnker, A., Hune, T., Larsen, K.G., Pettersson, P., Romijn, J., Vaandrager, F.W.: Minimum-cost reachability for priced timed automata. In: Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC), pp. 147–161. Springer (2001). DOI 10.1007/3-540-45351-2_15
7. Boudjadar, A., David, A., Kim, J.H., Larsen, K.G., Mikuionis, M., Nyman, U., Skou, A.: Hierarchical scheduling framework based on compositional analysis using Uppaal. In: Proceedings of the 10th International Symposium on Formal Aspects of Component Software (FACS), Revised Selected Papers, *LNCS*, vol. 8348, pp. 61–78. Springer (2013). DOI 10.1007/978-3-319-07602-7_6
8. Boudjadar, A., David, A., Kim, J.H., Larsen, K.G., Mikuionis, M., Nyman, U., Skou, A.: Widening the schedulability of hierarchical scheduling systems. In: Proceedings of the 11th International Symposium on Formal Aspects of Component Software (FACS), Revised Selected Papers, *LNCS*, vol. 8997, pp. 209–227. Springer (2015). DOI 10.1007/978-3-319-15317-9_14
9. Cassez, F., Larsen, K.G.: The impressive power of stopwatches. In: Proceedings of the 11th International Conference on Concurrency Theory (CONCUR), pp. 138–152. Springer (2000). DOI 10.1007/3-540-44618-4_12
10. Cesta, A., Fratini, S., Orlandini, A., Finzi, A., Tronci, E.: Flexible plan verification: Feasibility results. Fundam. Inform. **107**(2-3), 111–137 (2011). DOI 10.3233/FI-2011-397
11. Chadli, M., Kim, J.H., Legay, A., Traonouez, L., Naujokat, S., Steffen, B., Larsen, K.G.: A model-based framework for the specification and analysis of hierarchical scheduling systems. In: Proceedings of the Joint 21st International Workshop on Formal Methods for Industrial Critical Systems and 16th International Workshop on Automated Verification of Critical Systems (FMICS-AVoCS), *LNCS*, vol. 9933, pp. 133–141. Springer (2016). DOI 10.1007/978-3-319-45943-1_9
12. Cimatti, A., Micheli, A., Roveri, M.: Dynamic controllability of disjunctive temporal networks: Validation and synthesis of executable strategies. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence, pp. 3116–3122. AAAI Press (2016)
13. Cimatti, A., Micheli, A., Roveri, M.: Validating domains and plans for temporal planning via encoding into infinite-state linear temporal logic. In: Proceedings of the 31st AAAI Conference on Artificial Intelligence, pp. 3547–3554. AAAI Press (2017)
14. David, A., Du, D., Larsen, K.G., Legay, A., Mikučionis, M.: Optimizing control strategy using statistical model checking. In: NASA Formal Methods: Proceedings of the 5th International Symposium (NFM), pp. 352–367. Springer (2013). DOI 10.1007/978-3-642-38088-4_24
15. David, A., Du, D., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., Sedwards, S.: Statistical model checking for stochastic hybrid systems. In: Proceedings of the First International Workshop on Hybrid Systems and Biology (HSB), *EPTCS*, vol. 92, pp. 122–136 (2012). DOI 10.4204/EPTCS.92.9
16. David, A., Larsen, K.G., Legay, A., Mikucionis, M.: Schedulability of herschel-planck revisited using statistical model checking. In: Proceedings of 5th International Symposium ISoLA, Part II, *LNCS*, vol. 7610, pp. 293–307. Springer (2012). DOI 10.1007/978-3-642-34032-1_28
17. 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: Proceedings of the 9th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS), *LNCS*, vol. 6919, pp. 80–96. Springer (2011). DOI 10.1007/978-3-642-24310-3_7
18. David, A., Larsen, K.G., Legay, A., Mikuionis, M., Poulsen, D.: Uppaal SMC tutorial. International Journal on Software Tools for Technology Transfer pp. 1–19 (2015). DOI 10.1007/s10009-014-0361-y
19. David, A., Larsen, K.G., Legay, A., Poulsen, D.B.: Statistical model checking of dynamic networks of stochastic hybrid automata. ECEASST **66** (2013)
20. David, A., Rasmussen, J.I., Larsen, K.G., Skou, A.: Model-based Framework for Schedulability Analysis Using Uppaal 4.1d. CRC Press LLC (2009)
21. Gronback, R.C.: Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley (2008)
22. Henzinger, T.A.: The Theory of Hybrid Automata, pp. 265–292. Springer (2000). DOI 10.1007/978-3-642-59615-5_13
23. Jrges, S., Lamprecht, A.L., Margaria, T., Schaefer, I., Steffen, B.: A Constraint-based Variability Modeling Framework. International Journal on Software Tools for Technology Transfer (STTT) **14**(5), 511–530 (2012). DOI 10.1007/s10009-012-0254-x
24. Kim, J.H., Boudjadar, A., Nyman, U., Mikucionis, M., Larsen, K.G., Lee, I.: Quantitative schedulability analysis of continuous probability tasks in a hierarchical context. In: 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE), pp. 91–100 (2015). DOI 10.1145/2737166.2737170
25. Kim, J.H., Legay, A., Larsen, K.G., Mikučionis, M., Nielsen, B.: Resource-parameterized timing analysis of real-time systems. In: Hardware and Software: Verification and Testing: Proceeding of the 11th International Haifa Verification Conference (HVC), pp. 190–205. Springer (2015). DOI 10.1007/978-3-319-26287-1_12
26. Kim, J.H., Legay, A., Traonouez, L.M., Boudjadar, A., Nyman, U., Larsen, K.G., Lee, I., Choi, J.Y.: Optimizing the resource requirements of hierarchical scheduling systems. SIGBED Rev. **13**(3), 41–48 (2016). DOI 10.1145/2983185.2983192
27. Lamprecht, A.L., Naujokat, S., Schaefer, I.: Variability Management Beyond Feature Models. Computer **46**(11), 48–54 (2013). DOI 10.1109/MC.2013.299
28. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: Proceedings of the First International Conference on Runtime Verification (RV), *LNCS*, vol. 6418, pp. 122–135. Springer (2010). DOI 10.1007/978-3-642-16612-9_11
29. Legay, A., Traonouez, L.M.: Statistical model checking with change detection. Transactions on Foundations for Mastering Change I **1**, 157–179 (2016). DOI 10.1007/978-3-319-46508-1_9

30. Locke, D., Lucas, L., Goodenough, J.: Generic avionics software specification. Tech. Rep. CMU/SEI-90-TR-008, Software Engineering Institute (1990)
31. Margaria, T., Steffen, B.: Business Process Modelling in the jABC: The One-Thing-Approach. In: Handbook of Research on Business Process Modeling. IGI Global (2009)
32. Margaria, T., Steffen, B.: Simplicity as a Driver for Agile Innovation. Computer **43**(6), 90–92 (2010). DOI 10.1109/MC.2010.177
33. Montgomery, D.C.: Design and Analysis of Experiments. John Wiley & Sons (2006)
34. Naujokat, S., Lybecait, M., Kopetzki, D., Steffen, B.: CINCO: A Simplicity-Driven Approach to Full Generation of Domain-Specific Graphical Modeling Tools. Software Tools for Technology Transfer (2017). To appear
35. Naujokat, S., Traonouez, L.M., Isberner, M., Steffen, B., Legay, A.: Domain-Specific Code Generator Modeling: A Case Study for Multi-faceted Concurrent Systems. In: Proc. of the 6th Int. Symp. on Leveraging Applications of Formal Methods, Verification and Validation, Part I (ISoLA), no. 8802 in LNCS, pp. 463–480. Springer (2014). DOI 10.1007/978-3-662-45234-9_33
36. Oddi, A., Rasconi, R., Cesta, A.: A multi-objective large neighborhood search methodology for scheduling problems with energy costs. In: 2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI), pp. 453–460 (2015). DOI 10.1109/ICTAI.2015.74
37. Page, E.S.: Continuous inspection schemes. Biometrika **41**(1/2), 100–115 (1954)
38. 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 (2011). DOI 10.1145/1967021.1967029
39. Shin, I., Easwaran, A., Lee, I.: Hierarchical scheduling framework for virtual clustering of multiprocessors. In: Euromicro Conference on Real-Time Systems, pp. 181–190 (2008). DOI 10.1109/ECRTS.2008.28
40. Shin, I., Lee, I.: Periodic resource model for compositional real-time guarantees. In: Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS), pp. 2–13. IEEE Computer Society (2003)
41. Smith, D., Frank, J., Cushing, W.: The anml language. In: In ICAPS Poster session (2008)
42. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework (2nd Edition). Addison-Wesley (2008)
43. Wald, A.: Sequential Tests of Statistical Hypotheses. The Annals of Mathematical Statistics **16**(2), 117–186 (1945)